Christian Esteve Rothenberg

Compact forwarding: A probabilistic approach to packet forwarding in content-oriented networks

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Prof. Dr. Maurício Ferreira Magalhães

Campinas, SP
2010

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Esteve Rothenberg, Christian Rodolfo
Es85c Compact forwarding: uma abordagem probabilística para o encaminhamento de pacotes em redes orientadas a conteúdo / Christian Rodolfo Esteve Rothenberg. – Campinas, SP: [s.n.], 2010.

Orientador: Maurício Ferreira Magalhães.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computadores. 2. Comutação de pacotes. 3. Estruturas de dados (Computação). 4. Algoritmos 5. Probabilidade, Teoria da. I. Magalhães, Mauricio Ferreira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês:	Compact forwarding: A probabilistic approach to packet forwarding in content-oriented networks
Palavras-chave em Inglês:	Computer networking, Packet switching, Data structures (Computer), Algorithms, Probability theory
Área de concentração:	Engenharia de Computação
Titulação:	Doutor em Engenharia Elétrica
Banca Examinadora:	Djamel Fawzi Hadj Sadok, Edmundo Roberto Mauro Madeira, Eleri Cardozo, José Ferreira de Rezende
Data da defesa:	15/12/2010
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Christian Rodolfo Esteve Rothenberg

Data da Defesa: 15 de dezembro de 2010

Título da Tese: "Compact Forwarding: Uma abordagem probabilística para o encaminhamento de pacotes em redes orientadas a conteúdo"

Prof. Dr. Maurício Ferreira Magalhães (Presidente): [illegible]

Prof. Dr. Djamel Fawzi Hadj Sadok: Hadj Sadok

Prof. Dr. Edmundo Roberto Mauro Madeira: Edmundo R M Madeira

Prof. Dr. José Ferreira de Rezende: [illegible]

Prof. Dr. Eleri Cardozo: Eleri Cardozo

Resumo

Esta tese introduz um novo conceito para as redes de conteúdo denominado *compact forwarding*. Este conceito traduz-se na utilização de técnicas probabilísticas no plano de encaminhamento onde o espaço de identificação não é mais relacionado a um host final, mas sim, à identificação de conteúdo(s). A essência do conceito originou-se de uma questão básica, qual seja, onde deve ser colocado o estado associado ao encaminhamento do pacote? Nos elementos de rede ou no cabeçalho do pacote? A tese propõe duas soluções que representam estes extremos, SPSwitch, na qual o estado é colocado nos elementos de rede e, LIPSIN, onde o estado é colocado no cabeçalho do pacote. O denominador comum a essas soluções consiste na utilização de técnicas probabilísticas inspiradas no Bloom filter como elemento base das decisões de encaminhamento. A utilização de estruturas de dados derivadas do Bloom filter traz um custo adicional necessário à minimização dos erros associados à utilização de uma estrutura probabilística. A tese contribui com várias técnicas para redução desses erros incluindo a análise dos custos associados. Cenários de aplicação são apresentados para validação das propostas discutidas no trabalho.

Palavras-chave: Redes de pacotes, algoritmos, estruturas de dados, Bloom filter.

Abstract

This thesis introduces the concept of *compact forwarding* in the field of content-oriented networks. The main idea behind this concept is taking a probabilistic approach to the problem of packet forwarding in networks centered on content identifiers rather than traditional host addresses. The fundamental question explored is where to place the packet forwarding state, in network nodes or in packet headers? Solutions for both extremes are proposed. In the SPSwitch, approximate forwarding state is kept in network nodes. In LIPSIN, the state is carried in the packets themselves. Both approaches are based on probabilistic packet forwarding functions inspired by the Bloom filter data structure. The approximate forwarding state comes at the cost of additional considerations due to the effects of one-sided error-prone data structures. The thesis contributes with a series of techniques to mitigate the false positive errors. The proposed compact forwarding methods are experimentally validated in several practical networking scenarios.

Keywords: Packet networks, algorithms, data structures, Bloom filter.

Acknowledgements

To my advisor, Prof. Maurício Ferreira Magalhães, I am truly thankful for his encouragement and support throughout this thesis. Beyond being an always present mentor, I thank you Mauricio for the extracurricular aspects that made me feel at home.

I am greatly indebted to Dr. Pekka Nikander and the colleagues at Ericsson Research Nomadiclab. Thank you for showing me the meaning of high-quality corporate research. Having the opportunity of working together was a pivotal point in my PhD research for which I cannot be thankful enough.

Group project colleagues and co-authors, I could not have done this alone. This thesis would not be the same piece without your contributions and influences.

To the thesis committee for the insightful discussion and suggestions.

To Ericsson Research, for the financial support and the opportunity of aligning my work to corporate goals.

To Fundação CPqD, in addition to the professional opportunity, I am especially thankful to Helio Malavazzi and Jose Pedro de Freitas, who trusted and beliefed in me since our first E-Mail and VoIP conversations encouraging me to cross the ocean.

I wish to express my sincere gratitute to all my friends who heartily welcome me from day 1 in Brazil and accompanied through happiness contributing with lots of wonderful experiences.

To my parents, always supportive in the directions I have taken, I have not enough words to express what Anna and José Luis mean to me. Any skills that I may possess is born of their confidence in me and a consequence of having received their love, education, and blind support.

To my parents, Anna e José Luís

Contents

List of Figures

List of Tables

Glossary

API	Application Programming Interface
ARP	Address Resolution Protocol
ARPA	Advanced Research and Projects Agency
AS	Autonomous Systems
BGP	Border Gateway Protocol
CIDR	Classless Inter-Domain Routing
CCN	Content-Centric Networking
CDN	Content Delivery Networks
DCN	Data Center Networks
DHT	Distributed Hash Table
DPI	Deep Packet Inspection
DoS	Denial-of-Service
DNS	Domain Name System
FQDN	Fully Qualified Domain Name
iBF	in-packet Bloom filter
IP	Internet Protocol
ISP	Internet Service Provider
LLDP	Link Layer Discovery Protocol
MAC	Media Access Control
MPLS	Multi Protocol Label Switching
NAT	Network Address Translation
NSFNET	National Science Foundation Network
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
QoS	Quality of Service
RM	Rack Manager
SiBF	Switching with in-packet Bloom filters
SOA	Service Oriented Architectures
STP	Spanning Tree Protocol
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
URL	Unified Resource Locator
VL2	Virtual Layer 2
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network

Publications

1. C. Esteve Rothenberg, F. Verdi and M. Magalhães. "Towards a new generation of information-oriented internetworking architectures." In *ACM CoNext, First Workshop on Re-Architecting the Internet (Re-Arch08)*, Dec. 2008, Madrid, Spain.

2. P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. "LIPSIN: Line Speed Publish/Subscribe Inter-Networkings." In *ACM SIGCOMM'09*, Aug. 2009, Barcelona, Spain.

3. A. Zahemszky, A. Császár, P. Nikander and C. Esteve Rothenberg. "Exploring the Pub/Sub Routing & Forwarding Space." In *IEEE ICC, Workshop on the Network of The Future*, Jun. 2009, Dresden, Germany.

4. C. Esteve Rothenberg, P. Jokela, P. Nikander, M. Särela and J. Ylitalo. "Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters." In *5th European Conference on Computer Network Defense (EC2ND)*, Nov. 2009, Milan, Italy.

5. F. Verdi, C. Esteve Rothenberg, R. Pasquini and M. F. Magalhães. "Novas Arquiteturas de Data Center para Cloud Computing." Book Chapter, *28th Brazilian Symposium on Computer Networks (SBRC)*, May 2010, Gramado, Brazil.

6. C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Zahemszky. "Data center networking with in-packet Bloom filters." In *28th Brazilian Symposium on Computer Networks* (SBRC), May 2010, Gramado, Brazil.

7. A. Zahemszky, B. Gajic, C. Esteve Rothenberg, C. Reason, D. Trossen, D. Lagutin, J. Tuononen and K. Katsaros. "Experimentally-driven research in publish/subscribe information-centric inter-networking." In *Tridentcom* 2010, May. 2010, Berlin, Germany.

8. C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi and M. F. Magalhães. "The Deletable Bloom Filter: A new member of the Bloom family." In *IEEE Communication Letters*, June 2010, Volume: 14 Issue:6, On page(s): 557-559.

9. C. Esteve Rothenberg, "Re-architected Cloud Data Center Networks and Their Impact on the Future Internet." In New Network Architectures, Studies in Computational Intelligence, August 2010, Volume 297, 121-140.

10. M. Särelä, C. Esteve Rothenberg, A. Zahemszky, P. Nikander and J. Ott. "BloomCasting: Security in Bloom filter based multicast." In proceedings of the 15th Nordic Conference in Secure IT Systems (Nordsec) 2010, Oct. 2010, Aalto University, Espoo, Finland.

11. Carlos A. B. Macapuna, C. Esteve Rothenberg and M. F. Magalhães. "In-packet Bloom filter based data center networking with distributed OpenFlow controllers". In 2nd IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010) in conjunction with IEEE GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA.

12. M. Särelä, C. Esteve Rothenberg, T. Aura, A. Zahemszky, P. Nikander and J. Ott. “Forwarding Anomalies in Bloom Filter Based Multicast.” In the 30th IEEE International Conference on Computer Communications (IEEE INFOCOM 2011), April 10-15, 2011, Shanghai, China.

13. S. Tarkoma, C. Esteve Rothenberg and E. Lagerspetz. “Theory and Practice of Bloom Filters for Distributed Systems.” To appear in *IEEE Communications Surveys and Tutorials*, Second issue 2012.

14. C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Wiesmaier. “In-packet Bloom filters: Design and networking applications.” In Computer Networks, In Press, Corrected Proof, Available online 19 December 2010, ISSN 1389-1286, DOI: 10.1016/j.comnet.2010.12.005.

Intellectual Property Results

1. International Patent Application PCT/EP2008/063647, Patent: “Packet Forwarding in a Network,” Applicant: Ericsson AB, Publication number: WO 2010/022799, Publication date: 04.03.2010, Date of Filing: 10.10.2008, Inventors: JOKELA, Petri, ESTEVE, Christian, KJÄLLMAN, Jimmy, NIKANDER, Pekka, RINTA-AHO, Teemu, YLITALO, Jukka,

2. International Patent Application PCT/EP2009/062785, Patent: “Packet Routing in a Network,” Applicant: Ericsson AB, Publication number: WO/2010/142354, Publication date: 16.12.2010, Date of Filing: 01.10.2009, Inventors: ESTEVE, Christian, JOKELA, Petri, NIKANDER, Pekka, SÄRELÄ, Mikko, YLITALO, Jukka.

Chapter 1

Introduction

The Internet architecture has its origins in the 1970's by a small group of network researchers engaged in an academy/military research project funded by the Advanced Research Projects Agency (ARPA) of the US Department of Defense to build robust, fault-tolerant and distributed computer networks [1, 2]. The main goal of the DARPA Internet architecture was the development of an effective technique for multiplexed utilization of existing, heterogeneous, interconnected networks [3], i.e., the provision of a packet-based inter-networking architecture. The resultant Internet Protocol (IP) suite (TCP/IP) enabled an accelerated growth of the Internet including the integration of commercial ISP networks. In 1995, the central NSFNET backbone was transformed into a privatized, distributed backbone architecture. Being completely decentralized and lacking of a central coordinating (enforcing) instance, major architectural changes to the Internet have been hard to adopt. As a consequence, the underpinnings of today's Internet (i.e., hierarchical routing, TCP/IP, DNS) are fundamentally the same as projected over 30 years ago.

At the center of its original design, the IP is the single identifier space that enables global communications by providing a simple "best-effort" service of datagram delivery among network-attached devices. The original Internet addressing scheme mandates every host having an unique IP address with the fundamental functions characterized as follows in RFC 791 [4]: "A name indicates what we seek. An address indicates where it is. A route indicates how to get there." IP addresses combine two functions in one number space as they simultaneously act as routing *locators* (i.e. where you are attached to the network) and *identifiers* (i.e. who you are). This semantic overload of the IP is said to be at the root of many of the limitations of today's Internet architecture [5]. The engineering decisions behind the functionality and format of the IP and the original end-to-end model were a consequence of both the technological trade-offs of the time and the cooperative, experimental environment for which it was originally meant. Indeed, the fixed size, hierarchically structured 32-bit IP address format was a key factor for its technical feasibility, making packets easy to process by the resource-limited packet forwarding elements (i.e. routers), which only needed to inspect the network component of the destination address and could remain ignorant about the host part. As a consequence, routers only needed to exchange information about available routes to different networks.

Important features added to the architecture during the early 1980's include subnetting, Autonomous Systems (AS), and the Domain Name System (DNS) [6, 7]. Since then, various new transport technologies and protocol amendments have been introduced to provide new services and to increase the manageability of the network at a lower cost. For instance, Classless Inter-Domain Routing (CIDR) [8] was introduced to allow a flexible allocation of the original class-based IP address space. The Border Gateway Protocol (BGP) [9] introduced policy-based capabilities to reflect the business relationships among providers and was later extended with features for additional flexibility (e.g., community attribute, MED) and larger scale deployments (e.g., route reflectors and route aggregation). Multiprotocol Label Switching (MPLS) [10], originally called Tag Switching, was developed in the early 1990's to address core IP router performance issues. As time progressed, this packet forwarding technology has evolved into a powerful consolidation platform for IP backbones enabling new data services such as Virtual Private Networks (VPN) and, more recently, Carrier Ethernet solutions. In addition, a number of mechanisms have been developed and turned out to be useful to fight against problems caused by the original Internet design in an open commercial environment. For instance, Network Address Translation (NAT) boxes provide extended address spaces, configuration benefits, and partial protection for unwanted traffic at the cost of fracturing network connectivity [11, 12]. Mobile IP [13] provides means for host mobility by introducing network indirection points (i.e. home agents). From a larger perspective, such mechanisms only lead to a complex intertwined protocol suite between those wanting flexible network control and protection and those wanting freedom and connectivity, for legitimate or illegitimate reasons. As a consequence, the potential utility and innovation at the core of the Internet is put at risk.

While today's commercial use of the Internet unveils limitations with regard to mobility support, security, address space exhaustion, routing system scalability, and content delivery efficiency among others, the Internet is an ever growing success that works reasonably well [14]. Today, over 500 million end-hosts and 30.000 autonomous systems are connected. The advent of Internet-enabled objects, sensors, and mobile personal devices only makes this figure worse. At the same time, the Internet has been criticized to be "ossified" [15] due to the continuously patching approach based on ad-hoc protocol extensions and overlay solutions, which may be a complex and costly solution for the long term. The Internet is an example of what researchers [16] have called "organized complexity" modeled by the trade-offs made by engineered network design in connecting computer networks across a set of links resulting in a "robust yet fragile" network.

Last decade's efforts towards a future Internet architecture[1] have mainly focused on end-host reachability, revisiting concepts (e.g., IP identifier/locator split) to address end-to-end security, mobility and routing issues. All of these proposals are more-or-less *host-centric*. Recent research activities however point to a new way of looking at networking from a *content/information-centric* perspective [19, 20].[2] The Internet has shifted from being a simple host connectivity infrastructure to a platform enabling massive content production and content delivery, transforming the way information is generated and consumed. From its original design, the Internet carries datagrams inserted by sending hosts in a best effort manner, agnostic to the semantics and purpose of the data transport. There is a sense that the network could do more [21] and better given that today's use of the network is about retrieval of named pieces of data (e.g., URL, service, user identity) rather than specific destination host connections [22]. The Internet protocol suite (TCP/IP) is inherently unfair and inefficient for data dissemination purposes (e.g., multiple flows of P2P applications, redundant information over the wires [23]). With this in mind, the content-oriented research thread advocates for enhancements at the inter-networking layer not to be limited to QoS or routing scalability: data *persistence*, *availability* and *authentication* [24] of the data itself may be beneficial network capabilities from design.

Internet pioneer Van Jacobson provides a vision [22] to understand the motivation for a networking revolution; while the first networking generation was about wiring (telephony) and the second generation was about interconnecting wires (TCP/IP), the next generation should be about interconnecting information at large [19]. This shift in the orientation of network architecture design implies rethinking many fundamentals by handling information as a first class object. A key question is to what extent a new paradigm thinking 'out-of-the-TCP/IP-box' for the future network is really necessary, e.g., as packet switching was to circuit switching in the 70's. The reasoning is based on the large scale use of the Internet for dissemination of data. A myriad of devices, including user-attended terminals and long-running automated services, generate and consume content, without caring about the actual data source location as long as integrity, authenticity and timeliness are assured. This shift toward information-oriented networking is also noticeable in the momentum of service oriented architectures (SOA), XML routers, deep packet inspection (DPI), content delivery networks (CDN) and peer-to-peer (P2P) overlay technologies. A common issue is the necessity to manage a huge quantity of labeled data items, which is a quite different task than reaching a particular host in today's Inter-

[1]While Future Internet is a hot topic these days, the first wave on re-thinking the core Internet architecture can be dated back to the early 90's, when an increasing signs of strains on the fundamental architecture motivated the IETF a planned process for the architectural evolution as expressed by Clark *et al.* in RFC 1287 [17] entitled "Future of Internet Architecture." Later in 1995, Shenker argued for a new service model for the future Internet [18] to accommodate the requirements of (multimedia) applications. The convergence on IP resulted in massive work on QoS for packet networks, an issue that is being publicly debated these days under the controversial notion of 'net neutrality'.

[2]For the purposes of this thesis, we can and will interchangeable use the terms *information*, *content*, and *data*, together with *centrism* and *orientation* also used in an arbitrary manner to denote this paradigm shift.

net, where forwarding decisions are made not only by IP routers, but also by middleboxes, VLAN switches, MPLS routers, load balancers, mesh routing nodes and other cross-layer approaches.

Only time will tell whether and how these novel networking concepts evolve and get eventually deployed. History has shown that economics and not purely technological arguments is what ultimately turns prototypes into reality. Recent concerning events (and more to come) may potentially promote and accelerate the adoption of new inter-networking paradigms. Our days economy is Internet-sensitive, service outages due to Denial-of-Service (DoS) attacks or due to limitations of BGP insecure routing carry important worries and expenses (operational plus revenue loses). At the root of the well-known problems of unwanted traffic is the imbalance of powers in the original Internet design, in which the sender has too much control over the network, compared to the receiver. The network makes its best to deliver a packet to the destination, independent if the receiver wants to receive it or not. Different kinds of add-ons have been introduced to fight against these problems, such as firewalls and intrusion detection solutions. The same openness that helped to the successful growth of rich Internet applications is now putting at risk the privacy and security of network-attached corporations and individuals.

More than an endless discussion around ‘clean-slate’ design and deployable network evolution [25], feasibility work is needed along ‘clean-slate thinking’ beyond the TCP/IP heritage to foster innovation through questioning paradigms. This thesis is certainly not the first to turn into data-oriented networking [26] or to leverage the publish / subscribe communication paradigm [27]. Our contributions are less in form of an overarching solution but rather of enablers in the data forwarding stratum for novel networking paradigms. In this sense, we tackle challenges faced by the packet forwarding plane and explore probabilistic methods to solve them, contributing to the feasibility of scalable, content-oriented infrastructures.

1.1 How to read this Thesis

This thesis is meant to be read as follows. The next chapter introduces the research problem on compact forwarding, and gives an overview of the key contributions, including a description of the author’s publications appended in the Annex. Chapter 3 goes through the essential background in the field of our contributions. We contrast the original Internet design with the content-oriented usage of our days and highlight the fundamental differences which motivated taking a probabilistic approach when re-thinking the packet forwarding functions. For each sub area of our technical contributions, we discuss the main foundations and cover relevant related work. Chapter 4 reviews the contributions with more detail discussing how the applied principles appear in the developed solutions. Finally, Chapter 5 concludes the thesis with a series of final remarks and future lines of work.

Chapter 2

Research Problem

A shift in the orientation of network architecture design implies rethinking many fundamentals, for instance, defining a new identifier space for information objects of potentially different granularities (e.g., documents, channels, packets), enabling more expressive communication patterns (e.g., publish/subscribe, find/register), efficient transmissions (e.g., multicast, in-network caching, network coding) and increased resilience (e.g., security, data replication). The overall picture of a global scale communication infrastructure is complex and deserves detailed multi-disciplinary discussions (e.g., global namespaces, inter-networking functions, network management, security, stakeholders, etc.) involving architectural, engineering, and business considerations. We aim at addressing the challenges of novel content-oriented networks by re-thinking the key functionality of the forwarding plane under potentially new control planes (e.g., topology management, routing control), end-to-end communication paradigms (e.g., publish/subscribe), and namespaces (e.g., content identifiers, link identities).

2.1 Motivation and Scope

Given the grand-scale of the research field in function of different forms and characteristics of the inter-networking namespaces, we focus on the problem of trying to forward packets labelled with *flat* (unstructured, random looking) identifiers. For the sake of generality and the objectives of this thesis, we use the term *flat label* for information object identifiers or any other flat forwarding identifier carried in packet headers. Hence, our main abstraction is a flat label which is essentially a bit string representing any higher level information (e.g., content object, network link, multicast tree, host identifier).

The rationale behind focusing of forwarding on flat labels is the recent emergence of architectural proposals relying on flat labels due to their appealing capabilities such as being location-independent and having self-certifying names of hosts (cf. ROFL [28], AIP [29]) or data objects (cf. DONA [24], PSIRP [30], CCN [19]). In addition to the current frenzy of the so-called clean slate network designs, a conservative view of evolution of the Internet routing system also lends to the fact that topology-independence of the addressing/naming scheme becomes a fundamental requirement for e.g., self-configuration, multi-homing, nomadicity, and seamless mobility. Remarkable examples that have made their way to the IETF standardization process include the Host Identity Protocol (HIP) [31] and the Locator/ID Separation Protocol (LISP) [32]. Such efforts try to address the semantic overload of IP by separating host identifiers from network locators and thus introduce flat namespaces. Similarly, IP multicast group addresses are, in effect, flat identifiers that do not easily lend themselves to topological aggregation, resulting in forwarding state requirements that grow linearly with the number of senders or multicast groups.

The common way to make global network designs to scale is to *aggregate the address space* so that state is needed only for each aggregate. This scalability principle is also known as *information hiding* [33]. Noteworthy examples include the public switched telephone network aggregation of the telephone numbering system on geographical location, the Domain Name System (DNS) aggregation of its hierarchical naming system on zones, and the well-known aggregation of IP addresses on address blocks, formerly (pre-CIDR), constituting address classes.

The caveat of flat addresses is that they prohibit CIDR-style address aggregation [34], which is the best current practices for scalable routing and enables the global routing tables to grow sub-linearly with the number of networks on the Internet. Hence, a common challenge encountered by new networking paradigms is the need to take forwarding decisions at wire speed (Gbps) based on a large universe of flat (non-aggregatable) identifiers. Because the decisions need to be taken at high speed (typically in the order of tens to hundreds of nanoseconds), forwarding elements must use high-speed memory (typically SRAM), which is more constrained and expensive than other resources in network elements.

When looking for new means for aggregation to achieve a fast and scalable forwarding plane, *compression* appears as a natural technique to find a shorter representation that holds the same information as the original. The problem is that flat labels being completely random data strings cannot be compressed (cf. Pigeon-hole principle [35]). Therefore, the compact forwarding methods under study will consider the utilization of *lossy compression* techniques and try to address the question of whether a practical and correct forwarding machinery can be built on top of one-sided error mechanisms.

2.2 Relation to Previous Work

The mechanism of a typical packet router can be separated into (i) the control computations (e.g. routing) which take place in the background, and (ii) the fast forwarding path. Our focus falls in the latter. The *general routing problem* in a network consists of finding a *routing protocol*, or *routing function*, or distributed *routing algorithms*, such that, for any pair of source and destination nodes, any message from the source can be routed to the destination [36]. When routing a message from a source to a destination in the network, to decide where to forward the message to, a node relies on the current context information, which includes its local routing table, the destination address, and the message headers. As a result of the routing algorithms, network state in form of *forwarding information base* (FIB) encoded in *forwarding tables* is created by the (background) routing and resource control computations. The resultant in-network memory information enable hardware-assisted fast packet processing operations, which are relatively costly and difficult to change over time.

In an independent manner from the routing algorithms and upper layer control/signaling planes, we limit the scope of our problem to revisiting the field of suitable port-forwarding functions $i = F(x)$ that result in labeled packets being passed to certain output port(s) $\{i\}$. More specifically, we explore functions of the form $F(I, L, H)$, where:

I: Information in the packet header

L: Forwarding node local information (network state)

H: Headers-in-headers function (allows adding security functions, loop mitigation, and flexible forwarding strategies like trial-and-error)

This forwarding scheme is similar to the standard model of Peleg and Upfal [36] and the function *F: Headers-in-port*. As we shall see later, in contrast to previous work, this thesis takes a probabilistic approach to explore new dimensions in the solution space, questioning the traditional triangle of trade-offs in distributed computation theory:

- **Memory space:** Routing table size
- **Stretch:** Path length inflation
- **Adaptation costs:** Convergence measures, i.e., communication cost (routing updates per topology/policy change) plus processing cost to store and process/compute updated memory entries.

While the traditional triangular model works well for host-centric unicast routing and forwarding systems, we find necessary to introduce subtle refinements in order to (i) match our focus on packet forwarding for content-oriented networks with multicast being the basic communication mode, and (ii) account for our probabilistic approach, where we explore solutions that deliver packets over shortest paths but are subject to unnecessary packet duplication along their way. First, we explicitly add the packet header information I in terms of bits. Second, we restrict the memory space to the fast forwarding table size. Finally, we transform the stretch factor into forwarding efficiency to better account for the multicast mode of communication and the bandwidth penalties of approximate (probabilistic) solutions or eventually larger packet headers. Consequently, we can express the orthogonal metrics of forwarding as follows:

- **In-packet information:** Packet header size (i.e. comprising forwarding information)
- **In-network state:** Local forwarding table size
- **Efficiency:** Transport network usage extending stretch (i.e. packets taking longer paths than necessary) with packet duplication (i.e. copies sent over more links than necessary)
- **Adaptation costs:** Convergence measures, i.e., communication cost (signalling per context change) plus processing cost to store and process/compute updated forwarding table entries.

In previous work, the *compact routing problem* has been defined with focus on the implementation of protocols that require a low amount of hardware and amenable to the very-large-scale integration (VLSI) technologies of the 90's [37]. The trade-offs between space and efficiency for routing tables in host-centric networks under deterministic algorithms have been extensively studied over a variety of topologies and routing strategies [36]. The performance mismatch between the increasing transmission and switching capacity and the slower pace processor and memory speeds of IP routers in the 90's lead to considerable research in the design of forwarding table compacting techniques [38, 39]. A large body of work is (still) devoted to new algorithms and data structures for IP lookups and packet classification [40, 41], novel compact representations for structured graphs [42], and techniques for high-speed packet processing [43, 44, 45]. While previous work is concerned with an efficient implementation of standardized protocols and packet headers around the IP stack, our focus is on new forwarding paradigms well-suited for content-oriented architectures. Nonetheless, there is a ground intersection in algorithmic techniques and data structures applicable to the generalized problem of packet forwarding.

Latterly, *compact routing* for the Internet [46] has become an active field of research seeking for Internet routing algorithms such that given the full view of the network topology, the trade-off between routing table sizes and stretch is balanced in the most efficient way. Compact routing algorithms make routing table sizes compact by means of omitting some details of the network topology in an efficient way such that the resulting path length increase (compared to shortest path lengths, i.e., stretch) stays small. In accordance with [46], a routing algorithm is said to be compact if (1) node address and packet header sizes scale polylogarithmically, (2) routing table sizes scale sublinearly, and (3) stretch is a constant (i.e. does not grow with the network size). Only recently, the problem of *compact multicast routing* has been formulated and studied by Abraham *et al.* [47], resulting in the first memory-stretch tradeoff bounds for one-to-many communications. The multicast routing problem seeks to determine the network node memory requirements for a given routing algorithm that guarantees packet delivery to multiple destinations. According to [47], a routing scheme is compact if it is memory efficient and its goodness is measured in terms of stretch, i.e., the total network distance it utilizes compared with the shortest multicast path available.

2.3 Compact Forwarding

Inspired by, but complementary to the field of *compact routing*, we label our approach to the research problem as *compact forwarding*, which we frame as "the study of the trade-offs of in-network and in-packet state of forwarding methods that guarantee the correct delivery of packets in function of forwarding efficiency metrics." Our notion of *compactness* encompasses not only the studies of the minimal information base to perform memory-efficient forwarding operations but refers also to the probabilistic approach taken in our studies based on one-sided error-prone algorithmic techniques and data structures to materialize a forwarding plane for content-oriented networks. By *forwarding-correctness* we understand the process of packets being delivered *at least* to their intended destinations (i.e. the canonical requirement of deliverability of messages [48]) using a *finite* amount of resources. The finite resource constraint aims at discarding solutions based on naive broadcast/flooding techniques or solutions prone to infinite loops. The introduction of *forwarding efficiency* to quantify the bandwidth efficiency of multicast-capable forwarding methods allows the comparison of alternative (probabilistic) approaches in the solution space.

In comparison, compact routing is focused on optimal memory-stretch tradeoffs and restrain the inclusion of full path information in the packet headers (i.e. source routing). Our studies on compact forwarding techniques are orthogonal to the control plane specifics (e.g. routing algorithms) that feed the fast forwarding tables and hence determine the resultant stretch factors and adaptation costs. The goodness of a compact forwarding is measured in terms of – memory and bandwidth – efficiency rather than stretch. Packets delivered using a compact forwarding technique may use optimal paths to reach every destination but may consume extra bandwidth due to unnecessary packet duplications. Hence, we frame our research on the two extreme approaches of compact forwarding as follows:

In-network forwarding state approaches consist of having each forwarding node store a complete routing table. Each node can then perform independent forwarding decisions as it holds an entry for any destination a next-hop/link to which packets for that destination should be forwarded. The well-known drawback of this approach is the resulting routing table sizes, since each of the n vertex need to store $(n-1)$ entries, totaling $O(n^2 log(n))$ memory bits. Optimal (stretch 1) routing schemes for "simple" topologies like trees, rings, complete networks, grids and outer-planar networks are known to require $O(nlog(n))$ bits of in-network routing information and $O(log(n))$-bit headers [36].

Definition 1: We say a forwarding method is *compact* if each forwarding table entry requires less than $log(n)$-bits per routable object in an n-dimension universe.

Clearly, exact match address lookup systems (e.g. Ethernet MAC forwarding) are not compact. Compact implementations of the forwarding information base like decision trees may fall as well into the category of compact forwarding. By extension of the definition above, a prefix-based forwarding method (e.g. IP longest prefix matching) can be said to be *asymptotically compact* if we average the size of each forwarding table entry over the complete universe of routable objects.

In-packet forwarding state approaches (e.g. source explicit routing) consist of each datagram carry in its header a (complete or partially complete) set of directives (e.g. path descriptors) along which the datagram should be forwarded. As a consequence, forwarding nodes only need to maintain local (reduced) forwarding information (e.g. the identity of its neighbors). The caveat is that datagram headers need to be of variable size $f(n)$ and still be processed at wire speed in-packet.

Definition 2: We say a forwarding method is *compact* if the datagram header size is of fixed size with independence of the forwarding directives included.

By datagram header size we mean the number of bits required to take the forwarding decision. In that sense, source routing forwarding schemes based on the concatenation of network identifiers are not compact. Examples include IP source routing options [4] and tunneling/encapsulation techniques such as IEEE 802.1ah, IP-in-IP, or GRE [49].

This thesis builds around the concept of compact forwarding by researching questions like:

- what is a suitable forwarding substrate for content-oriented networks departing from the host-centric paradigm of IP?
- which are the candidate features and data structures of such forwarding planes?

- what are the dimensions and limits of the solution space, i.e., what is the minimum forwarding information base (in-network and in-packet) to move data objects at scale?
- can we do better than the fundamental trade-offs of distributed systems theory by introducing non-deterministic (probabilistic) techniques?
- what considerations and enhancements are needed to build a correct distributed forwarding gear on top of one-sided error prone forwarding decisions?

2.4 Approach and Contributions

Motivated by the needs of networking at an information layer, this thesis explores new approaches to the fundamental trade-offs of packet routing to provide forwarding services with scalability, multicast-friendliness and security in mind. Due to the lack of aggregation capabilities of flat labels and the compact forwarding goal of seeking the minimal information base to deliver packets at scale, we have dived into solutions based on error-prone probabilistic data structures providing lossy compression functionality. By exchanging correctness (traduced in forwarding efficiency penalties) for space/memory time requirements (traduced in reduced information base in packet headers and network nodes), we explore a new dimension in the traditional design trade-off.

Basically, we express the packet forwarding problem as two extreme set membership problems solved by virtue of the popular data structure Bloom filter named after his inventor Burton Howard Bloom [50]. The already 40-year-old probabilistic data structure supports element queries for set memberships and its unique encoding algorithm gives it excellent space/time savings at the cost of correctness. Being a one-sided error-prone lossy summary technique, Bloom filters are subject to return false positives upon querying for the presence of an element, i.e., claiming that an element is present when it was not really inserted. Conversely, false negatives are not possible per design, Bloom filters always return a correct answer to intentionally inserted elements. By virtue of its hash-based construction, the functionality of a Bloom filter is independent from the nature (type, size, structure) of the elements at hand. The accuracy of the membership answers, that is the false positive performance, depends only on the bit per element ratio (i.e. data structure size m divided by the number of inserted elements n) and thus provides compact forwarding decisions independently from the size of the identifier space. As we shall see, the benefits of this probabilistic approach may well pay off the drawbacks in terms of larger bandwidth consumption due to the usage of extra network links and larger packet header sizes.

This dissertation makes three sets of contributions: (i) principles, (ii) algorithmic techniques, and (iii) applications. The first set of contributions includes a collection of generic and technical principles useful for designing scalable forwarding mechanisms motivated by the advent of content-oriented network architectures. The second set includes the conception and application of algorithmic techniques to cope with the limitations of previous work in probabilistic data structures when used to build forwarding mechanisms following those principles. Finally, the third set of contributions is the application of the compact forwarding methods in practical networking architectures, including an Internet-scale publish/subscribe network architecture, inter-domain multicast, and a scalable data center architecture.

Many of the contributions of this thesis fall into the category of filling the gap between theory and practice, i.e., applying theoretical results on probabilistic data structures to solve the performance and scalability problems faced by network architectures moving packets characterized by a large space of flat labels. In that sense, we do not provide a holistic solution to the broader architectural problems, but rather contribute with a set of enablers for the forwarding plane. At the same time, as a consequence of dealing with general purpose probabilistic data structures, the algorithmic contributions and the proposed methods can be applied to solve other related problems in distributed systems.

2.4.1 Publications

The author's publications that underpin this thesis can be found in the annex and will be cited hereafter from [A] to [H]. Figure 2.1 gives an overview of how the publications can be mapped to the different areas of the contributions. While this thesis is an outcome of my research achievements, some clarification on the work done in collaboration is needed. The author's role and contributions to the publications were as follows:

- Publication A (6 p.): C. Esteve Rothenberg, F. Verdi and M. Magalhães. "Towards a new generation of information-oriented internetworking architectures." In *ACM CoNext, First Workshop on Re-Architecting the Internet (Re-Arch08)*, Dec. 2008, Madrid, Spain.
  - Contributions: The author was the architect of the SPSwitch forwarding engine. He was responsible for the design and evaluation of the proposed solution.
- Publication B (12 p.): P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. "LIPSIN: Line Speed Publish/Subscribe Inter-Networkings." In *ACM SIGCOMM'09*, Aug. 2009, Barcelona, Spain.
  - Contributions: The author was a member of the LIPSIN architecture design team, with special focus in the Link ID Tag extensions, the parameter optimization and practical evaluation of the in-packet Bloom filter data structure.

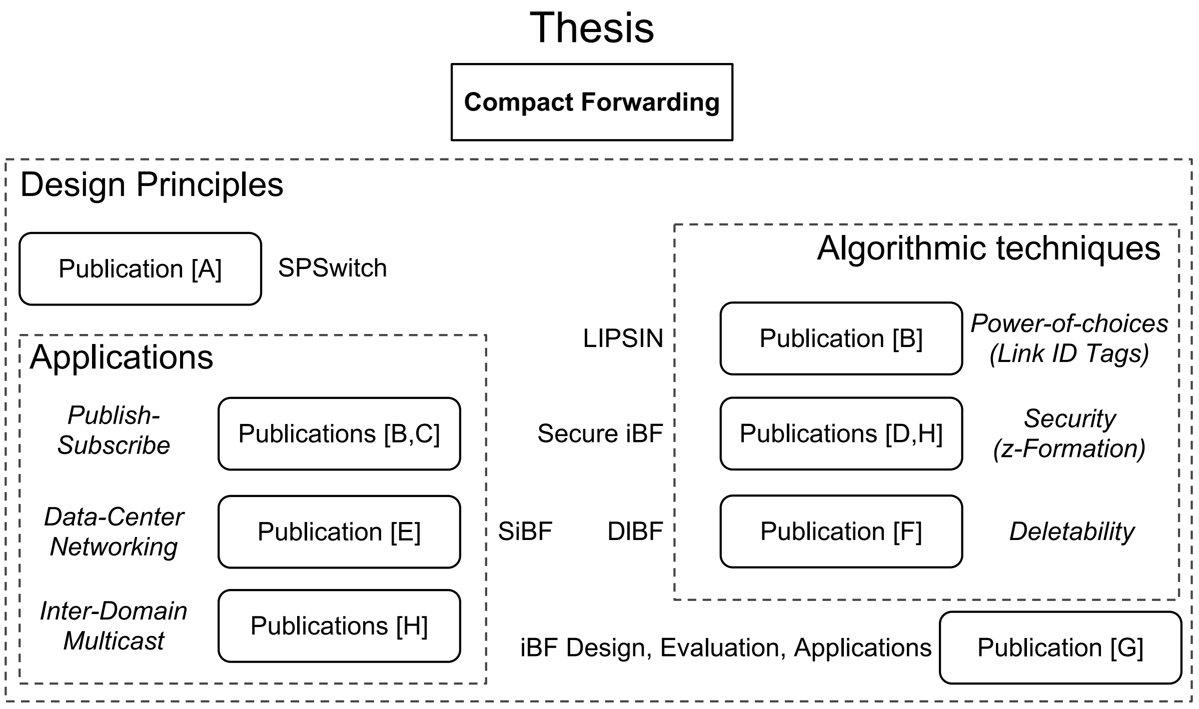


Fig. 2.1: Overview of the main topics and publications of the thesis.

- Publication C (6 p.): A. Zahemszky, A. Császár, P. Nikander and C. Esteve Rothenberg. "Exploring the Pub/Sub Routing & Forwarding Space." In *IEEE ICC, Workshop on the Network of The Future*, Jun. 2009, Dresden, Germany.
  - Contributions: The author contributed to the editorial work of the paper, with emphasis on the edge switching and integration challenges.
- Publication D (6 p.): C. Esteve Rothenberg, P. Jokela, P. Nikander, M. Särela and J. Ylitalo. "Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters." In *5th European Conference on Computer Network Defense (EC2ND)*, Nov. 2009, Milan, Italy.
  - Contributions: The author contributed to the design of the Z-formation forwarding method and was responsible for the probabilistic security analysis.
- Publication E (14 p.): C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Zahemszky. "Data center networking with in-packet Bloom filters." In *28th Brazilian Symposium on Computer Networks* (SBRC), Gramado, Brazil, May 2010.
  - Contributions: The author was the main architect and prototype co-developer of the SiBF data center architecture.

- Publication F (3 p.): C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi and M. F. Magalhães. “The Deletable Bloom Filter: A new member of the Bloom family.” In *IEEE Communication Letters*, June 2010.
  - Contributions: The author was the designer of the DlBF data structure and responsible for the simulation-based evaluation work.
- Publication G (16 p.): C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Wiesmaier. “In-packet Bloom filters: Design and networking applications.” In *Elsevier Computer Networks*.
  - Contributions: The author was the leading author of the work and responsible for the design and practical evaluation of the proposed extensions (performance, security and deletability) to in-packet Bloom filter designs.
- Publication H (16 p.): M. Särelä, C. Esteve Rothenberg, A. Zahemszky, P. Nikander and J. Ott. “BloomCast: Security in Bloom filter based multicast.” In proceedings of the 15th Nordic Conference in Secure IT Systems (Nordsec) 2010.
  - Contributions: The author was a member of the design team and contributed to the security evaluation of the proposed solutions. The author’s contributions to the related work [51] were the practical implementation issues and the simulation-based evaluation of the inter-domain permutating iBFs.

2.4.2 Overview of the Achievements

When designing a routing and forwarding system, one has to consider the balance between the amount of state stored in the network nodes and the amount of information carried in the packet headers. On one extreme, we can compactly store the forwarding information base (state) in network nodes that test the incoming packet labels for presence in a next hop destination set. Along this in-network solution space, we propose a Bloom-filter-inspired port-forwarding engine well-suited for flat identifiers [A]. On the other extreme, we have explored moving the forwarding state to the packets themselves by compactly carrying the forwarding directives (i.e. an explicitly defined source route). This way, forwarding nodes only need to test for membership of their locally maintained link identifiers in order to take the next hop forwarding decision. Along the in-packet forwarding information space, we explore probabilistic methods to provide explicit source routing while keeping fixed-sized packet headers [B,E,H].

Obviously, a balanced approach where some forwarding nodes are stateful and some are stateless is not only possible but advisable when aiming at Internet-scale systems. Both extremes allow us to trade a certain amount of over-deliveries (i.e. duplication of messages over unnecessary links) for simple, resource-efficient forwarding operations. We let up to the specifics of the network architecture the precise selection of the sweet points – probably dominated by the technology constraints at the time – in terms of correctness (i.e. forwarding efficiency), and the amount of in-network and in-packet state where the benefits pay off the drawbacks.

In the remainder of the Chapter, we provide an overview of the author's contributions by briefly presenting the developed concepts and applications of the compact forwarding methods.

In-network compact forwarding on flat identifiers

The goal of our compact forwarding problem is to calculate the set of outports $f(I)$ associated with a packet labelled by a (flat) identifier I. The challenge is that the output is a function of long, randomly looking identifiers (e.g. 256-bit hash-based IDs). Storing a mapping between log(I)-bit identifiers and an output set of (virtual or physical) ports is an expensive proposition. In terms of time, it is expensive as it can take long time because the keys are long. In terms of space, it is clearly expensive due to the size of the flat identifiers. To be compact, the implementation of $f(I)$ should consume less than log(I) bits per entry.

The caveat of flat labels is that, being random data streams, they cannot be compressed, i.e., for the complete identifier space, there exists no shorter representation that holds the same information as the original. By operation of the pigeonhole principle,[1] no lossless compression algorithm can efficiently compress all possible data, and completely random data (e.g. assumed for hash-based identifiers) cannot be compressed. For this reason, many different algorithms exist that are designed either with a specific type of input data in mind or with specific assumptions about what kinds of redundancy the uncompressed data are likely to contain.

Due to their independence from the element size or form, hash functions — an old workhorse of system designers — seem a natural fit to deal with flat identifiers. Unfortunately, perfect hashing techniques are not feasible either due to the dynamics of the unknown set formed by the forwarding identifiers. Moreover, forwarding tables based on hash table implementations that store the key together with the output next hop value(s) $\langle I, f(I) \rangle$ are not compact and imply prohibitive fast for-

[1] Also commonly called Dirichlet's box principle or Dirichlet's drawer principle. The formal statement of the pigeonhole principle is "there does not exist an injective function on finite sets whose co-domain is smaller than its domain" [35], i.e., if n items are put into m pigeonholes with $n > m$, then at least one pigeonhole must contain more than one item. This principle also proves that any general-purpose lossless compression algorithm that makes at least one input file smaller will make some other input file larger. Otherwise, two files would be compressed to the same smaller file and restoring them would be ambiguous.

warding memory requirements given the large identifier space.

Hence, we explore the field of suitable compression schemes for the forwarding state. We avoid lossy dictionaries because of their two-sided error that returns both false positives and false negatives. We start by considering the traditional Bloom filter data structure due to its simplicity and tunable one-sided error rate that trades speed/memory with correctness (false positive rate). We then investigate the required variations on the probabilistic data structure to address the issues of the standard design and meet the goals of a correct packet forwarding machinery.

The SPSwitch - Bloom-filter-inspired port forwarding: The SPSwitch [A] leverages a packet classification technique (d-left fingerprint-compressed hash tables [52]) to function as an abstract switching element with one programmable Bloom filter per output (physical/ virtual links, internal processes). Due to its hashing-based nature, the switching decisions can be taken at O(1) time and accommodate various types of packet identifier spaces (e.g., 256-bit content IDs, flat forwarding labels). Acting as a probabilistic hash table, it returns always the inserted output value and, additionally, in rare cases (false positive rate $\approx O(10^{-6})$) it incurs in extra (non-intended) multicast operations. Trading of over-deliveries for state reduction and line speed operations is justified given the small, multiplicative false positive rate of chained switching operations and the data-oriented paradigm where redundant traffic can be cached and pruned at the edges if no matching subscriptions are installed. At routing domain boundaries [C], making a switching or mapping decision between a large flat identifier space and the next routing and forwarding identifier space needs to be efficient both in space (small high speed memories in forwarding elements) and time (few computation cycles per packet). The SPSwitch aims at solving this problem: with only a few bits per entry (e.g. 40-50 bits) and independently from the identifier space (e.g. 256-bit flat labels), port forwarding operations and label switching can be performed in a fast and resource-efficient way.

With the insights that hash-based data structure may play a fundamental role as efficient data aggregators in network architectures based on non-structured (non-aggregatable) namespaces (e.g., self-certified content names, MAC addresses), we moved towards exploring the other extreme of packet forwarding, namely carrying the routing information state into the packets themselves.

In-packet compact source explicit routing

At the opposite end from the present Internet design lies source routing [53], with its well-known problems related to packet sizes and security [54]. In strict source routing, the packet's path is described, hop by hop, in the packet header. A single forwarding node does not have to know anything else than its neighbours; it just picks the next hop node from the packet header and delivers the packet.

Routable in-packet Bloom filters: By compactly encoding source routes with an in-packet Bloom filter (iBF) [B], we can address one of the main caveats of source routing, namely the overhead of having to carry all the routing information in the packet. As a side benefit of this approach to in-packet compact forwarding, network identifiers are not explicitly revealed to outside observers, neither the sequence or amount of hops involved. The approach is based on the assumption that there are no stable end-to-end *addresses* for the network nodes, for three reasons [B]. Firstly, relying on such addresses would not contribute to the envisioned benefits in fighting unwanted traffic and empowering the receivers. Secondly, in a content-oriented architecture, long-lived node addresses should not be needed. Thirdly, any such (topology-dependent) addresses used as identifiers are detrimental to the ability of supporting mobility and multi-homing. This way, network nodes may no longer need long-lived addresses, and to a large part, they may also remain anonymous to most of the network. However, such node-address-less design generates new kind of problems, especially for routing and forwarding.

The in-packet Bloom filter (iBF) approach solves the forwarding problem without end-to-end addressing, using a link-identifier-based approach that combines elements from source routing and stateful routing, in a flexible way. When used to take forwarding decisions, false positives are translated into packets being transmitted over additional links than the ones originally inserted. As long as the false positive rate is low enough, falsely packet duplications can be considered acceptable due to active caching and the decreasing probability of concatenated false positives over multiple hops. To encode delivery trees, a set of statistically unique directed links can be formed. So, any forwarding tree can be seen as a set of unidirectional links. Then, the iBF describing the delivery tree is placed into the packet header and sent to the network. By checking for certain bit patterns in the header, each forwarding node tests which of its outgoing links are included into the set. Since this is a simple binary `AND` operation, next hop checks can be done parallel in hardware, producing a very fast forwarding plane. It can be shown that this approach leads to fast hardware-amenable forwarding decisions at the forwarding nodes [B,E], reduces the possibilities for malicious nodes for sending unwanted traffic [D], and at the same time has the seeds to scale to Internet-wide dimensions [C].

In general, an iBF [G] is well suited for network applications where one might like to include a list of elements in every packet, but a complete list requires too much space, and, additionally, the elements should remain undisclosed. In these situations, a hash-based representation like a Bloom filter can dramatically reduce space, maintaining a fixed header size, at the cost of introducing false positives. Example network applications beyond multicast forwarding [B,H][55] include, data-path security [56], wireless sensor network security [57], IP traceback [58] and loop prevention [59].

Extensions to probabilistic data structures

Bloom filters may impress by their sheer elegance and performance, and have been widely used in network applications [60]. Sometimes their application to resolve some problems may be classified as indiscriminately used tool and there may be better domain-specific alternatives to Bloom filters under the same parameter space (cf. [61]). Certainly, there is no one size that fits all solution, and the naive application of Bloom filters for a system critical component like packet forwarding deserves careful considerations of whether the effects of false positives can be contained and whether there are alternative or complementary algorithmic solutions.

When applied to the problem of packet forwarding, we encounter the necessity of obeying the policy of no false negatives, which would put the packet delivery at risk. That is, to be *correct*, the forwarding methods should guarantee that packets are being delivered, at least, to their intended destinations. The amount of consumed resources should be bounded (e.g. no infinite loops), and, clearly, solutions with the best efficiency should be favored.

In sake of addressing the effects of the one-sided error methods, we have proposed and validated extensions to achieve a practical, flexible packet forwarding toolbox. The non-deterministic side of Bloom filters means that the resultant forwarding algorithms may need to make a random choice among alternatives when it encounters a choice point at which it cannot know which alternative leads to the desired outcome. It is often the case that the standard Bloom filter data structure is not enough to achieve the desired system performance (e.g., certain false positive rate or guarantees) or functionality (e.g., deletions, counters, security).

We have studied in-depth the design space and proposed algorithmic enhancements to the construction of the hash-based Bloom filter data structure. This way, we contribute to the hypothesis that a correct forwarding machinery can be built on top of false-positive prone decision steps.

Playing with the power of choices: We fight the randomness of hashing algorithms with a multiplicity of choices in the combination of hash functions. By doing so, we empower the application to pick the best candidate for a certain optimization goal (e.g., less false positives, loop-avoidance) [B]. The strategy of having multiple representations for the same element set enables re-inserting determinism in the one-sided error-prone system by having the candidates tested prior to their use [E].

We use the notion of *power of choices* [62] and take advantage of the random distribution of the bits set to 1 to select the iBF representation among the d candidates that leads to a better performance given a certain optimization goal (e.g., lower fill factor, avoidance of specific false positives). This way, we follow a similar approach to the Best-of-N method applied in [63], with the main differences of (1) a distributed application scenario where the d value is carried in the packet header, and (2) the best candidate selection criterion is not limited to the least amount of bits set but includes optimization criteria specific to packet forwarding policies (e.g., loop-freeness, avoid costly links).

Having "equivalent" iBF candidates enables to define a selection criteria based on multiple objectives. To address *performance* by reducing false positives, we can select the candidate iBF that presents the best posterior false positive estimate (*fpa-based selection*). If a reference test set is available to count for false positives, the iBF choice can be done based on the lowest observed rate (*fpr-based selection*). Another type of selection policy can be specified to favour the candidate presenting less false positives for certain "system-critical" elements (*element-avoidance-based selection*) or other iBF optimization goals, for instance, element deletability as explored in [G].

DlBF - The deletable Bloom filter: Under some circumstances, a desirable property of iBFs is to enable element deletions as the iBF packet is processed along the network. For instance, this is the case when some inserted elements are to be processed by only one networking element (e.g., a node / link identity within a source route) or bit space for new additions is required. Unfortunately, due to its compression nature, the bit collisions hamper naive element removal unless we want to introduce false negatives into the system. To overcome this limitation (with high probability), so-called Counting Bloom filters [64] were proposed to expand each bit position to a cell of c bits. Each bit vector cell acts now as a counter, increased on element insertion and decreased on element removal. As long as there is no counter overflow, deletions are safe from false negatives. The caveat is the c times larger space requirements, a very expensive price for the tiny iBFs under consideration.

We have designed the deletable Bloom filter (DlBF) [F], a new Bloom filter variation based on the novel idea of compactly encoding the information of where collisions happen when inserting elements. The DlBF enables false-negative-free deletions at a fraction of the cost in memory consumption. Depending on how much memory space one is willing to invest, different rates on element deletability and false positives can be achieved. The DlBF is well-suited for other use cases where reconstructing the filter upon set membership changes is either infeasible or too costly. For standalone applications, removal of element fingerprints is commonly desirable for functionality or optimization purposes. For distributed applications, a deletable filter can be thinned out as queried elements are processed in order to (i) avoid repeated matches upfront, (ii) reduce false positives, and/or (iii) enable fresh bit space for new additions.

zFormation - Secure Bloom filter constructs: The hash-based nature of Bloom filters provides some inherent security properties to obscure the identities of the inserted elements from an observer/attacker. However, we have identified a series of use cases where extra security means are desirable. For instance, an attacker can deduce by simple iBF inspection whether two packets contain an overlapping set of elements (e.g. network paths). Considering another threat model, an attacker may wait and collect a large sample of iBFs to infer some common patterns of the inserted elements. In any case, if the attacker has knowledge of the complete element space, it can certainly test for presence of every element and obtain a probabilistic answer of what elements are carried in the iBF.

The main idea of the zFormation function [D] is to bind the Bloom filter operations (*insert* and *query*) to an invariant of the packet (e.g., a packet identifier, packet payload, etc.) and a distributed shared time-based secret. Basically, we want to make the inserted elements packet-specific and expirable. By obscuring the actual inserted elements, an iBF becomes meaningful only if used with the specific packet, avoiding the risk of an *iBF replay attack*, where the routing iBF is placed as a header of a different packet. By additionally binding the iBF generation and query operations to a time-variant secret, we can turn the iBF expirable and useless after some period of time. Applying these ideas to the iBF-based source routing architecture we can secure the data forwarding plane against Distributed Denial-of-Service attacks [D]. The resulting forwarding identifiers can act simultaneously as path designators, i.e., define which path the packet should take, and as capabilities, i.e., effectively allowing the forwarding nodes along the path to enforce a security policy where only explicitly authorized packets are forwarded. The compact representation is based on a small Bloom filter whose candidate elements (i.e. link names) are dynamically computed at packet forwarding time using a loosely synchronized time-based shared secret and additional in-packet information (e.g., invariant content or flow identifiers). The capabilities become thus expirable and flow-dependent, but do not require any per-flow network state or memory look-ups, which are traded-off for additional, though hardware-amenable, per-packet computation. Hence, the proposed compact forwarding approach takes the in-network state requirements down to near-zero-state, since the core forwarding decision is based on a pure computational operation rather than based on memory-based forwarding table lookups.

Practical Applications in Network Architectures

Our final set of contributions include the experimental validation in practical network architectures of the hypothesis that one-sided error-prone forwarding algorithms are not only feasible in practice but may carry benefits largely paying the potential effects of false positives.

The idea of iBF-based forwarding was initially conceived for the information-centric networking requirements of an Internet-scale publish/subscribe architectural proposal [B, C]. Multicast-capable iBFs can be formed by collecting enough topology information and then used to form the delivery trees to forward packets from the data sources to their sinks. For instance, the topology information can be gathered on demand by the flow/communication initiation packets (e.g. multicast join messages [H]) or can be managed in a more central approach like the distributed path computation entities of (G)MPLS (cf. [65]). Following the same rationale of a managed network control environment, we have applied the notion of iBF forwarding in data center networks to provide a scalable and flexible packet forwarding service below the IP layer.

SiBF - Switching with in-packet Bloom filters : Motivated by the unprecedented scale, cost, and control requirements of cloud data center networks, we have designed and implemented SiBF [E], a data center network architecture based on the stateless forwarding service provided by iBF encoding source routes and carried in the Ethernet MAC fields. SiBF follows an identifier/locator separated approach where IP addresses act solely as identifiers and oblivious routing is provided by randomly using iBF-encoded routes between the communicating endpoints (e.g. virtual machines). Our design borrows characteristics from a few novel data center network designs, for instance building upon proven interconnection topologies (e.g. Clos networks) and reliance on logically centralized controllers (e.g., Ethane [66], Fabric Manager [67], Directory Service [68], NOX [69]). Compared to related work, one key difference of our work is the provision of a forwarding primitive based on an iBF expedited by what we call a new entity in the data center: the Rack Manager (RM). The RM follows a direct network control approach (cf. 4D [70]) to transparently provide the networking functions (address resolution, route computation) and support services (topology discovery, monitoring, optimization) to unmodified (physical and virtual) servers behind Top-of-Rack (ToR) switches.

Forwarding in SiBF addresses the issue of having a system with two mutually conflicting requirements: (1) flat (non-hierarchical) Ethernet addresses, and (2) aggregation. While our approach initially seems to open another vector of the design space, namely potential efficiency penalties due to false positives resulting in some packets unnecessary using some extra links, the proposed solution is free from false positives by exploiting the power of choices along two dimensions: (1) multiple paths, and (2) multiple iBF representations. The former strategy consists of simply having the iBFs tested for false positives prior to their use, i.e., RMs maintain a ToRsrc-ToRdst routing matrix filled only with false-positive-free iBFs (one iBF per available path).

The proposed solution makes better use of the 96-bit space of source and destination MAC addresses without sacrificing the nice plug and play properties of random Ethernet MAC addresses. To our benefits, the “Bloomed” MAC identifiers do not incur in encapsulation or shim-header overheads. Additionally, the iBF-based fine control over the path travelled by packets enables load balancing schemes to avoid hot spots by bouncing off traffic flows to intermediate switching elements, or explicit control over a sequence of middlebox services (e.g., firewall, SSL offloaders, DPI).

2.5 Summary

This chapter introduced the motivation for this thesis and presented the research problem around the concept of compact forwarding in the context of content-oriented networking paradigms. The scope of the research problem was restricted to suitable compact port-forwarding functions and the relation to previous work was discussed. Finally, the main contributions of this thesis were introduced by giving an overview of the author's publications and the compact forwarding methods therein.

In the next chapter, essential background on the original Internet architecture and its evolution is provided. The discussion on related work includes the ongoing efforts towards content-oriented network architectures, and a review of probabilistic data structures used in network applications.

Chapter 3

Background

This chapter lays the fundamental background on the architectural principles of the Internet, its evolution, the roles of the control and data planes, and the research efforts towards future Internet architectures. Then, the rationale behind content-oriented networking is introduced as a new paradigm with profound implications on naming, routing and forwarding. Remarkable proposals along this trend are presented with special focus on the approaches and challenges of content-oriented packet forwarding. The review of related work would not be complete without surveying the state of the art of probabilistic data structures with special attention to the proposed variations and networking applications of the Bloom filter data structure.

3.1 Principles and Evolution of the Internet Architecture

The Internet was not built in response to popular demand, real or imagined; its subsequent mass appeal had no part in the decisions made in 1973. Rather, the project reflected the command economy of military procurement, where specialized performance is everything and money is no object, and the research ethos of the university, where experimental interest and technical elegance take precedence over commercial application. This was surely an unlikely context for the creation of what would become a popular and profitable service.

"Inventing the Internet" by Janet Abbate, 1999

The Internet — the collection of linked network elements and distributed systems that enable global communications — is an ever growing success that has transformed the way businesses are done and how people socialize. In brief, the big transformation is the emergence of an ubiquitous information platform, democratized in a way that people and machines can generate and consume content in an unprecedented manner, ultimately becoming the enabling global communication infrastructure of what has been recently touted as the fifth Utility, i.e., cloud computing.

Over 500 million end devices and 30.000 autonomous systems are connected today. The number of connected endpoints, is expected to grow at a high pace with the progress of IP-enabled mobile technologies (3G, 4G), the end of the digital divide, the advent of the Internet of Things (sensors, actuators, daily objects), and the proliferation of virtual machines in geo-distributed data centers.

The original Internet architecture was built around a host-to-host communication model, and is perfectly suited for applications, such as file transfer and remote login, that focus on conversations between pairs of well-known and stationary hosts. The basic architectural principles included end-to-end addressing, global routability, and a single namespace of IP addresses that could serve simultaneously as locators and host identifiers. A second namespace of Fully Qualified Domain Names (FQDN) was later added, and the Domain Name System (DNS) was developed to map between such names and addresses.

Astonishingly, the ‘heart’ of the Internet architecture, i.e., the Internet Protocol Suite, is almost the same as what Internet pioneers projected as part of an experimental research project to provide interconnection between a few heterogeneous computer networks. More than 30 years have passed since the specification and implementation of the single network layer protocol [3] that today underpins the converged communication infrastructure hosting these days’ World Wide Computer.

The initially monolithic Transmission Control Program [1] was later divided into a layered architecture consisting of the Transmission Control Protocol (TCP) [71] at the connection-oriented layer and the Internet Protocol (IP) [4] at the connection-less internetworking (datagram) layer (see Figure 3.1). With the insights of running code at scale, the central algorithms of TCP were devised [72] and after several revisions, the latest specification of TCP [73] contains the protocol operations along four intertwined algorithms: Slow-start, congestion avoidance, fast retransmit, and fast recovery.

```
Protocol Layering

                     +---------------------+
                     |     higher-level    |
                     +---------------------+
                     |        TCP          |
                     +---------------------+
                     |  internet protocol  |
                     +---------------------+
                     |communication network|
                     +---------------------+
```

Fig. 3.1: IP Suite Protocol Layering. Source: [71]

Along this journey, many ideas were added and dropped (e.g., variable address lengths). Once the specification was implemented and broadly adopted, it has shown to work good enough [14] to cope gracefully with all the changes below (e.g., link layer speeds and capabilities) and above the IP waist (e.g., real-time voice/video/gaming, content overlays, application-level multicast, etc.). As a consequence of its own success, attempts to change the core of the Internet protocol suite have failed.

Once turned into a commercial artefact to transport money rather than packets, numerous considerations have appeared including security and scalability issues. As a side effect of being commercially-driven and de-centralized, changes in the core require global deployments justified by the right incentives for all players. This process has been described by the National Research Council [74] as *ossification*, i.e., or inability to change, in multiple dimensions: intellectual (pressure for compatibility with the current Internet risks stifling innovative intellectual thinking), infrastructure (ability of researchers to affect what is deployed in the business-driven core infrastructure), and system (limitations in the current architecture have led to shoe-horn solutions that increase the fragility of the system).

The lack of industry motivation to implement risky changes have been noted by networking researchers as the main non-technical challenges associated with deploying various flavors of Quality of Service (QoS), IP Multicast, and IPv6. In the case of the later, the most recent specification for the next generation network layer protocol has been waiting for deployment since 1998. Only recently, some signs of adoption can be highlighted, but still the overall IPv6 traffic in Internet backbone and regional ISPs accounts for only a fraction of the total Internet traffic. Among the show-stoppers of IPv6 lays the success of carrier-grade NAT solutions, which despite clouding the end-to-end significance of IP addresses have proven to be useful in (i) extending the life of the IPv4 address space, (ii) helping in contain security threats, and (iii) assisting renumbering procedures. It can be also argued that the lack of incentives for IPv6 may be also a consequence of the inherent resembleness to IPv4, i.e., carrying many of the IPv4 mistakes or inadequacies to today's use of the network.

Similar difficulties apply to the Border Gateway Protocol (BGP). Established as the *de facto* standard for inter-domain routing, being multi-domain/provider requires almost a global-scale, synchronized protocol update in case of changes that are not backwards-compatible.

In the case of IP multicast, many multicast routing protocols have been proposed since its conception in the early 90's, see comprehensive surveys [75, 76]. A number of theories, both technical and business based, have been proposed to explain why inter-domain multicast has not yet seen deployment [77, 78, 79]. The proposed reasons include the lack of control for who can receive, the knowledge of the number of receivers at the source, and the lack of incentives of upstream providers to reduce the amount of monetized traffic.

At the same time, the beauty of the flexible design of the Internet is that it has enabled numerous evolutionary 'patching' approaches to emerge without having to change the network infrastructure. Indeed, it is being argued that the Internet is a victim from its own success. Many ad-hoc and patches solutions have been and are being developed. These include middleboxes like firewalls, NATs, Mobile IP home agents, and security protocols like IPSEC or TLS. Furthermore, we are assisting to an increase of overlay networks for content distribution like P2P systems and CDNs.

Despite all the non-driving forces, the sense that the Internet suffers from design issues that could be solved on the whiteboard, has and still motivates more fundamental research on global-scale networked systems, specially as new spins to rethink the Internet emerge (cf. content-orientism in Section 3.2). Similarly as how IP was initially conceived as an overlay on top of the telephone system, researchers are trying the Tantalus task of devising whether today's Internet overlays could be the precursors of the so-sought future generation Internet architecture.

3.1.1 The role of the Control and Data Planes

What you need is that your brain is open. — Paul Erdös

Networking systems are commonly decomposed into functional modules, which are organized into groups or "planes." The network architecture defines how these (protocol) functions are placed at different points in the network (e.g., end-systems, access/core routers, servers, clients, overlay nodes, etc.). Functional correctness and efficient coordination between different functions typically requires the maintenance of shared information across time – commonly called "state" – at various nodes and in packet headers. See [80] and [81] for a comprehensive discussion on the principles and guidelines of network architectures. As observed by Rexford *et al.* [82], the broad organization of functions into planes can be dictated by the following time- and space-scales:

Data plane functions are those that operate at line-speed time-scales and involve packet handling primitives (e.g., congestion control, reliability, encryption). For example, the data plane (also-called forwarding plane) performs packet forwarding (e.g., longest-prefix match on destination IP field to decide on the egress interface to the next hop), as well as the access control lists (ACLs) that filter packets based on rules defined on the header fields. Additional functions of the fast data plane include tunneling, queue management, and packet scheduling. In terms of spatial scales, the data plane is local to an interface card of a single router.

Control plane functions happen at a longer time-scale and enable data plane functions. The control plane consists of the network-wide distributed algorithms that compute parts of the state required for the data plane (e.g., routing, signaling, name resolution, address resolution, traffic engineering). For example, the control plane includes BGP update messages and the BGP decision process, as well as the intra-domain routing protocols such as OSPF, its link-state advertisements (LSAs), and the shortest-path routing algorithm (e.g., Dijkstra). As a result of these protocols the forwarding table (FIB) that determines the data plane packet forwarding is generated. Control plane functions include all type of signaling protocols (e.g., MPLS, RSVP, ATM PNNI signaling, telephony/X.25/ISDN) that associate global identifiers (addresses) to local state (e.g., labels, resources). Moreover, end-to-end signaling (e.g., SIP, TCP and IPSEC connection setup) belong also to control plane functions that setup data plane enabling state. Name resolution based on DNS is also a control plane function that maps names to addresses and enables end-to-end data plane activities. Control-plane functions may be data-driven, i.e. triggered by a data-plane event (e.g., ARP, DNS), or be purely control-driven and operate in the background (e.g., OSPF).

Management plane functions deal with monitoring, management and troubleshooting of networks and work at an even larger time-scale than control plane functions – as they typically involve human interactions (e.g., manual configurations). The management plane centralizes and analyzes measurement data from the network (e.g., SNMP, active probes, tomography) and generates the configuration state on the individual routers. For example, the management plane collects and combines Simple Network Management Protocol (SNMP) information, traffic flow records, OSPF LSAs, and BGP update streams. Network management tools that configure OSPF link weights and BGP policies following traffic engineering goals would be part of the management plane. Similarly, a system that analyzes traffic measurements to detect intrusion attempts or denial-of-service attacks and react by accordingly configuring ACLs to block malicious traffic would be part as well of the management plane.

In today's IP networks, the data plane operates at the time-scale of packets (Gbps) and the spatial scale of individual network elements (switches/routers), the control plane operates at the time-scale of seconds – commonly with a partial view of the network (e.g., an OSPF area), and the management plane operates in a centralized fashion at the time-scale of minutes or hours and the spatial scale of the entire network.

Recent re-architecting proposals suggest shifting or consolidating functions from one plane to the other (e.g., the 4D architecture [70, 82, 83]). For instance, functions involving the decision process of the distributed control plane (e.g., BGP routing) could be merged into the management plane for the sake of stability, responsiveness and functionality of routing. As observed by Feamster *et al.* [84], the growth of the Internet has introduced considerable complexity into the global routing infrastructure, with features being added to BGP to support more flexibility at a larger scale. Arguably, this complexity has made routing protocol behaviour hardly understandable, increasingly unpredictable, and error prone [85]. Disaggregation of router functionality, and in particular the separation of control plane functions from forwarding functions, is a current trend in new generation routing architectures. In an IP world, separating routing from forwarding [84] means IP "routers" becoming "lookup-and-forward" switches to forwarding packets as rapidly as possible without being concerned about path selection – a task that can be arguably outsourced to a centralized management environment.

On a related track, enabling some degree of *network programmability* by centralized controllers has been the very sought holly grail of network infrastructure providers. Efforts to this goal can be dated back to the 90's and the efforts in programming telecommunication networks [86], including the OPENSIG community, IEEE 1520, MPOA (Multi-protocol over ATM), GSMP (General Switch Management Protocol) RFC3292, the active network research thread [87], and more recently, ongoing work on the IETF ForCES (Forwarding and Control Element Separation) protocol [88] and the OpenFlow initiative [89]. Basically, the OpenFlow protocol specifies a standard way for controlling packet forwarding decisions in (remote) software while keeping the hardware vendors in charge of the device implementation. This separation of concerns leads to a promising combination (aka software-defined networks) of the programmability of general purpose PCs – implementing an evolvable and customizable control plane – with line-speed commercial networking hardware taking care of the fast data plane functions (e.g., port-forwarding, header re-writing) based on cached control plane decisions.

Another promising line of work advocates for the introduction of a *knowledge plane* [90] as an intermediary plane that holds knowledge of the network resources (such as topologies and more) in order to reason about failures and enabling thereby novel network management capabilities.

3.1.2 Placement of Functions and State

A circuit is just one long packet — "Patterns in Network Architecture" by John Day, 2008

The basic division on where to place networking functions are "end-systems" (i.e., Internet hosts) and "network elements" (i.e., routers, switches, etc.). Certain control-plane functions (e.g., routing protocols) and their associated state variables (e.g., routing tables) are placed in the network (L3) and largely in the network elements. In contrast, end-hosts usually have simple default routes (L3 state) but are best suited to implement per-flow functions like reliability at the transport layer (L4).

This choice of function placement is commonly referred to as the End-to-End (design) principle. The End-to-End argument — originally formulated by Saltzer, Reed and Clark [91] — suggests that specific application-level functions usually cannot, and preferably should not, be built into the lower levels of the system (i.e., the core of the network). The explanation is stated as follows in the original paper:

> "... functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level... "

With the communication version of the End-to-End argument being (for the example of providing reliability):

> "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communications system. Therefore, providing that questioned function as a feature of the communications systems itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement)... "

Hence, the End-to-End argument implies that several functions like reliability, congestion control, session/connection management are best placed at end-systems (i.e., performed on an end-to-end basis), while the network layer remains in charge of functions which it can fully implement (i.e., routing and datagram delivery). As a result, end-points are intelligent terminals in control of the communication while the forwarding layer of the network is kept simple.

The End-to-End argument has been (and still is) subject to multiple mis-interpretation and heated debates. The root of those conflicting views and the issues with the End-to-End principle is probably that it was conceived at a time and for the sake of very different technological and economical considerations compared to what we have today. Over the last decade, new requirements have emerged for the Internet and its applications. In order to meet these various requirements, certain stakeholders have arguably opted for the addition of new mechanism in the core of the network. Examples of those emerging requirements include [92]: (i) operation in an untrustworthy world, (ii) more demanding applications (e.g., real-time audio/video), (iii) ISP service differentiation, (iv) the rise of third-party involvement, (v) less sophisticated users.

Back to the role of the control and data planes and the distribution of functions and state, the End-to-End argument is well reflected in the state required for routing and forwarding being fully distributed and placed at every router and the end-system, i.e., "*... placing functions (and state) at the lowest system level where they can be completely and correctly implemented...*". Noteworthy, routing state maintained at end-systems is minimal (i.e., a default route) compared to the per-flow TCP state at end-systems. In the case of TCP, the system state consists of TCP protocol parameters (e.g., window size, RTT estimate, slow-start threshold) as well as application-level data such as user ID, session ID, and authentication status.

Hence, a connection-oriented protocols like TCP over a connection-less network protocol like IP do not require on “per-connection” or “per-flow” state at routers, i.e., the network is stateless on a per-flow basis. Routing protocols need to maintain routing state in form of routing tables inside the network to facilitate forwarding. Unlike the signaled state of circuit-oriented approaches, routing state is an example of “soft state” maintained by the control plane protocols running in the background (e.g., routing protocols like OSPF). Frequent changes and aging of this soft state information are part of the normal operations. In Ethernet for instance, ARP table entries are timed out unless refreshed. This approach contrasts to the telephony-oriented design of virtual circuits over packet networks such as X.25, ATM, and frame relay, where switches maintain “hard” forwarding state for each active virtual circuit.

The growth and manageability of the routing tables of backbone routers has lead to an increasing concern calling for routing alternatives to alleviate the routing scalability problems by, e.g., reconciling the roles of network locator and identifier of IP as proposed by the Locator/ID Separation Protocol (LISP) [32]. In addition, various algorithms have been proposed in the literature including the dynamic re-assignment of network addresses and techniques for compact IP forwarding tables [38, 39].

A large body of work on so-called *compact routing* schemes [46, 93, 94, 95, 96, 97] has studied the tradeoff between space and time of routing schemes. Space can be generally expressed in terms of packet headers in messages and routing table size at network nodes, while time can be given in terms of the computation required to select next-hop nodes and the length (or cost) of the actual network paths between senders and receivers. This is commonly known as the *space-stretch tradeoff* that dictates the relation between routing table size and route length of any routing scheme [98]. Kleinrock and Kamoun [99] were first in showing how hierarchical node addressing could produce highly scalable routing tables, which is the basis of CIDR and OSPF/ISIS. Peleg and Upfal [36] pioneered the studies on the fundamental tradeoffs for routing tables in general networks. They provided tight upper and lower bounds for the tradeoff on routing table size and stretch factors for universal routing schemes, that is, compact routing schemes applicable to arbitrary networks.

Shortest-path routing approaches represent one extreme approach as it only optimizes the route length while the routing table size grows linearly with the network size. Compact routing refers to design space of routing schemes with optimized space-stretch tradeoffs. Optimal universal compact routing schemes are able to reduce the routing table size down to $O(sqrt(n))$ per node at the cost of a three-fold increase in stretch. Better results [46, 94, 96] can be obtained exploiting the actual structure of operational networks where a power-law degree distribution is common, i.e., few nodes have a very high degree and many have a low degree.

Only recently, Abraham *et al.* [47] have opened the research front on distributed *compact multicast routing schemes* seeking tradeoffs between storage and space for various problems in one-to-many communications. While memory is defined as in the unicast, the stretch factor represents the cost of the compact multicast route and the cost of a Steiner (optimal) tree between the same set of target nodes. The several variants of the problem studied in [47] include: (i) labeled – in which polylogarithmic node names can be assigned, (ii) name-independent – in which node names are arbitrarily chosen, (iii) dynamic – in which nodes dynamically join and leave the multicast service and the goal is to minimize as well the total cost of control messages needed to maintain the tree. The memory requirements of a compact multicast scheme is defined as in the unicast problem. However, stretch is re-defined as the maximum, over all choices of source nodes and sets of destination nodes, of the total weight of edges used by the algorithm to deliver the packet to all target nodes, divided by the weight of the minimum Steiner tree with the same set of destinations. Packet headers are allowed to include a list of all destinations, but are restrained, as in the unicast case, from including full path information in a source route approach.

Indeed, packets themselves are another possible location to place (routing) state in addition to end-hosts or network nodes. The idea of adding information to packet headers to make packet processing easier [100] is as old as the historical debate between connection-oriented and connectionless networking technologies [101].

In a source routing approach [53], the source specifies the partial or complete path that packets are supposed to take through the network. A common implementation (e.g., IP source routing [102], Dynamic Source Routing in wireless ad-hoc networks [103]) consists of packet headers containing a list of addresses, which potentially incurs in high overheads. In signaled architectures like ATM or MPLS, source routing requires an explicit mapping to local state information (e.g., MPLS labels, ATM VCI/VPI) using the signaling protocol (e.g., RSVP, MLD). A noteworthy hybrid approach proposed in the late 90's is IP switching [104], which aimed at taking the advantage of the robustness and scalability of connection-less IP, and the performance (speed, capacity) of ATM switches. The main idea behind an IP switch was a software-based IP router control plane attached to proven switching hardware and the ability to cache routing decisions in switching hardware. To be beneficial, IP switching required a mechanism (algorithm) to associate long-living IP flows with ATM labels.

Noting that state in protocol stacks limits scalability (i.e., servers need to commit per-client resources), Trickles [105] proposes a TCP-like transport protocol that enables pushing encapsulated state from the server to the client, so that system state is kept entirely on one side of the network connection. Additional benefits of this stateless network protocol based on self-describing packets carrying encapsulated per-connection server state include the ability to replicate and migrate services between servers and the avoidance of many types of denial-of-service attacks. The security implications transforming statefull protocols into ‘stateless connections’ by including protocol state in the packets themselves has been described in a general framework by Aura and Nikander [106]. Yet another example of moving state to packets is the per-packet dynamic state QoS approach in [107], where QoS requirements are specified in each packet rather than based on out-of-band signaling together with complex queuing and resource reservations mechanisms at routers.

Trading packet headers for packet processing [100] argues for a series of mechanisms to add information to packet headers to speed up packet processing. More precisely, Chandranmenon and Vargheses propose three mechanisms based on this principle: (i) source hashing – where the source adds a random label acting as a probabilistically unique hash key field (similar to IPv6 Flow Label [108]), (2) threaded indices – where packets carry per hop index for each destination, and (3) a data manipulation header with information required for data processing (e.g., destination buffer names, encryption keys) and dispatch (e.g., destination process IDs).

Following the same principle, Bremler-Barr *et al.* [109] propose a distributed IP lookup based on adding a “routing clue” to each packet. The so-called clue consists of additional bits in the IP header (5 in IPv4, 7 in IPv6) to tell downstream routers where the last IP lookup ended in terms of longest prefix match. This way, the IP lookup work gets distributed. The idea of passing a clue within packets in a way that routers can share what they have learned from a packet with succeeding routers may have other generalizations and applications in different domains (e.g., distributed packet classification for QoS or firewall purposes). In IPv6 [110], a 20-bit Flow Label field [108] has been specified to be used by a source to label those packets with special handling requirements by IPv6 routers, such as non-default quality of service or “real-time” service, or as a pseudo-random flow identifier suitable for use as a hash key by routers to look up the associated flow state. [1]

While source routing has appealing properties in terms of reduced in-network state requirements and explicit path control, there are important shortcomings that have limited the wide adoption of source routing and multi-path routing in multi-domain, connection-less networks. The traditional downsides of source routing include the inefficient coding of source routes (overheads in every packet), the requirement of global routing information at sources, the lack of incrementally deployable strategies, required signaling upon topology changes, and a number of security issues [53, 54].

Despite the extensive adoption of IP communications, the tension between connection-oriented and connection-less networking techonologies is still alive [101]. Connection-oriented technologies are a fundamental underpinning in today’s data network layers below IP. The most recent trends in achieving fast forwarding and enhanced packet transport services include the GMPLS [112] control capabilities to Ethernet data plane and determine the behaviour of the IP layer on top of these new solution(s). GMPLS controlled Ethernet Label Switching (GELS) is a clear example of the demand

[1]See [111] for a recent draft document discussing use cases and issues of the IPv6 flow label.

for flexible and flow-oriented efficient transport over any link layer. Like other technologies in the past, by separating control and forwarding plane, GMPLS introduces more flexibility and important performance gains due to the pure hardware fast label switching technology.

In general, any packet forwarding approach can be classified into four strategies [113]:

1. Modify both router forwarding state and forwarding information in packet headers (e.g., most Active Network proposals [87, 114]).
2. Modify router state but not packet headers (e.g.,“active storage” type of networks [115]).
3. Modify packet forwarding information but not routing state (e.g.,i3 [116], NATs and middle-boxes in general i.e., DOA [12]).
4. Modify neither router forwarding state nor packet state (e.g., original IP).

As discussed by Popa, Stoica and Ratnasamyl [113], each class poses different trade-offs between flexibility and security. For instance, allowing data packets to modify router forwarding state opens significant security risks. At the limit, an application could implement complicated distributed protocols (e.g., routing protocols) whose safety would be notoriously hard to verify. As a consequence, the first two, active-networks-like approaches are commonly disconsidered for public Internet-scale deployments. In contrast, the last strategy (no state modification at all) offers limited flexibility, as users (end-users, network operator) have no control on packet forwarding.

3.1.3 Towards the Future Internet Architecture

There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things.

- Niccolo Machiavelli, The Prince (1532)

Dated back to 1991, Request for Comments 1287 [17] is probably the first holistic effort “Towards the Future Internet Architecture.” Lead by the Internet Engineering Task Force (IETF) – famous for its motto of running code and rough consensus – the Internet community recognized the need for a major discussion of Internet architectural issues.

In addition to contributing the historical debate on the relevance of the TCP/IP with respect to the OSI protocol suite, several important areas for architectural evolution were identified and coarse-grain research agenda was proposed around (1) routing and addressing, (2) multi-protocol architectures, (3) security architecture, (4) traffic control and state, and (5) advanced applications. As a consequence of this call for action, during the next 5 to 10 years multiple protocols were developed to address the acknowledged issues, including extensive work on providing a QoS-oriented service model for IP networks [18].

It is worth to note that many of recent hot topics in network research were already devised at that time. For instance, resembling modern location/identifier split approaches, RFC 1287 contemplates the possibility of including a 64 bit field as a “flat” host identifier together with a mapping service between the host id and the Autonomous System (AS) or the Administrative Domain (AD). Moreover, research directions on further means for aggregation suggested to consider routing on ADs. Other remarkable suggestions include support to in-network store and forward services in the spirit of DTNs or even the provision of a Global File System, in line with the current trend in content-centric networks. More along the requirements of the latter appears when a new definition of the Internet was proposed based on a new unifying concept [17]:

“Old” Internet concept: IP-based. The organizing principle is the IP address, i.e., a common network address space.

“New” Internet concept: Application-based. The organizing principle is the domain name system and directories, i.e., a common - albeit necessarily multiform - application name space.

This early form of name-oriented identifier/locator separation suggests changing the coupling of “connected status” from the traditional IP address (i.e., network numbers) to names and related identifying information contained in the distributed Internet directory (i.e., DNS).

Saltzer [117] was one among the firsts that recognized the requirement of having clear distinctions among network elements; the most common, and least practiced, of these distinctions is between a host identifier and its address. With dynamic bindings at multiple levels, names of objects can become location independent and some naming architectures support different types of mobility (e.g., nodes or services) and the notion of indirection [118] or delegation [119].

During the past decade, so-called future network research (mainly by the academia) has prompted creative architectural proposals, such as LNA [119], FARA [120], Plutarch, Triad [121], i3 [118], SNF, TurfNet, IPNL [122] and NodeID [123], among others. At the core of these new generation network architectures are naming and addressing frameworks that are significantly more flexible, expressive, and comprehensive than the Internet hierarchical IP address space. These naming frameworks are key components that enable advanced inter-networking capabilities, such as multi-homed mobility, dynamic composition of networks, or delay and disruption-tolerant (DTN) communication.

A common approach adopted by most of the new architectures includes the identifier and locator split with the intrinsic benefits for mobility, multi-homing and security, the last due to the coupling between the identifier and the hashing of a corresponding public key [31]. These approaches recover the original Internet end to end transparency with the end host being the most important element in the sender-controlled IP architecture. One of the main consequence of this model is the huge power given to the sender side, that by knowing the destination identifier (i.e., network locator), is able to send unwanted traffic to the receiver side.

Researchers and visionaries around the world have claimed the need to rethink the Internet (r)evolution. The so-called ‘clean slate’ approach has its roots in a research program at Stanford University[2] and places two basic questions:

a) with what we know today, if we were to start again with a clean slate, how would we design a global communications infrastructure?

b) how should the Internet look in 15 years?

Since then, research to circumvent current Internet limitations has been commonly divided into those advocating new architecture designs (*clean-slate*), and those defending an *evolutionary* approach due to incremental deployability concerns [25]. From a pure research perspective, however, *clean-slate design* does not presume *clean-slate deployment* and aims at innovation through questioning fundamentals.

3.2 The Rise of a Content-Oriented Internet

If content is King, then distribution is King Kong.

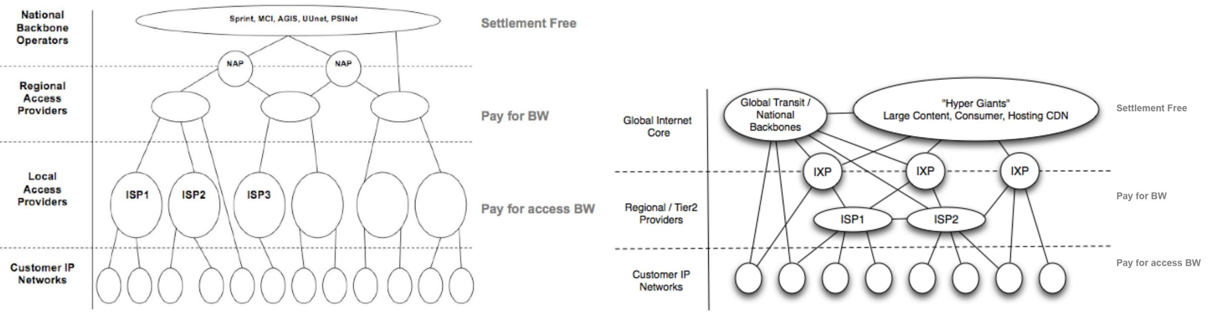
— An old media saying

Networks today were designed for the technologies of the ’70s, when people accessed limited information, on a static network, through a single computer system. Current networking approaches focus on moving packets of data, which are attached to a fixed machine location and unique IP address from source to destination. This is in profound contrast to today’s Internet environment, where people access unprecedented amounts of digital information, through dynamic networks, and with multiple, often mobile devices.

At the same time, the available network access capacity has increased significantly in recent years with more homes worldwide having broadband connections. This increased bandwidth has led to the proliferation of rich multimedia content that are accessed either from Content Distribution Networks (CDN) or through the file-sharing peer-to-peer (P2P) networks. The vast majority of Internet usage today is data retrieval (e.g., HTTP video, direct download, P2P) and not specific host-to-host conversational services (e.g., VoIP, SSH, VPN). In fact, recent studies [124] show that with larger amount

[2]http://cleanslate.stanford.edu/

of data being transferred over HTTP, today, most Internet inter-domain traffic by volume flows directly between large content providers (e.g., Google, Microsoft, Facebook), hosting / CDNs (e.g., Akamai, LimeLight) and consumer networks. As shown in Figure 3.2, this content-driven changes in the peering relationships translate into an evolution of the Internet logical topology. This bypassing of Tier-1 ISPs accompanied by the deployment of content-provider owned wide-area networks and edge cache/front-end servers directly inside ISPs has been also touted as a 'flattening' Internet topology [125].



(a) Traditional Internet logical topology. (b) Emerging new Internet logical topology.

Fig. 3.2: The left figure generally reflects the hierarchical historical BGP topology. The figure on the right illustrates emerging content-oriented Internet traffic patterns. Source: [124]

P2P and CDNs have become so successful because they fill the Internet design gap of optimizing data delivery [126]. The Internet was designed to provide good support for end-to-end host communications. However, today's Internet is mostly used for data dissemination, which is a quite different task than reaching a particular host. As a result, tasks like content distribution have become unnecessarily hard, and have required the deployment of ad-hoc overlay systems like Akamai or BitTorrent that had not been previously foreseen.

Actually, users do not care where the data comes from, as long as timeliness, data integrity and authenticity are ensured. Most Akamai content is served from caches co- located within provider infrastructure and IP address space. The majority of the available data can be classified as long-tail content because it is not relevant to everyone except for certain groups of people. On the other hand, a small part of the data (head-tail content), is relevant for large groups of people demanding tools in the service level (e.g., monitoring, caching, DNS re-direction) to create and consume these different types of data efficiently. From the networking side, CDN approaches to provide some sort of content routing and caching running over TCP/IP represent a huge cost due to the several overlap of functions and the manageability demanded by the several protocol levels.

These changes in the Internet pattern usage and the advances and cost reduction of storage and processing power have lead to the concept of "storage in the network," that suggests to look at the

Internet as a database [127], or to re-design the Internet to act natively as an efficient CDN [126].

In addition, there is a notable evolution in information technology developments that confirm this shift in the focus of network evolution from packets to content. Traffic engineering trends are moving from data flows (e.g. QoS - Quality of Service) to application-level services (e.g. QoE - Quality of Experience). This shift is accompanied by the demand of deep packet inspection (DPI) technologies at edge routers. Moreover, we are assisting to an emergence of application and service oriented architectures (SOA) and a fast growth of the publish/subscribe model and enterprise service buses (ESBs). In this context, XML-based routers [128] have been developed and further reaffirm this content-oriented trend in networking.

Answering to the clean-slate questions, if today we were to design things from scratch, we would probably add content-awareness and massive storage capacity at Internet routers. This is not a discussion on network versus host intelligence but rather a reconsideration of what should be the first-class object in the new Internet. A naming approach based on data rather than end-hosts would enable a democratization of scalable content dissemination and make it part of the Internet core in the same way that connectivity was democratized as IP emerged.

This is precisely the point that Van Jacobson [129] has recently risen up. His content-centric networking vision suggests to shift the point of view of the ongoing approaches to solve the problems of the current Internet. He argues that current networking protocols are inadequate, because they were designed for a conversational network, where two people/machines talk to each other, while today the majority of network traffic comes from a machine acquiring named chunks of data (web pages, multimedia files, E-mails, sensor data, etc.). The user cares about content and is oblivious to its location. For data retrieval, the current Internet architecture (and many host-centric future Internet approaches) is far from convenience and carries both naming- and protocol-level issues.

In TCP/IP, connected is a binary attribute meaning you are either part of the Internet and can talk to everything or you are isolated. In addition, connecting requires a globally unique IP address that is topologically stable on routing timescale (minutes to hours). This makes it difficult and inefficient to handle mobility and opportunistic transport in the Internet [129, 130].

Under a content-oriented paradigm, information is indexed by keys (labels, data names) and retrieved by request. Protocols are declarative (i.e., say what you want, not where/who to get it from). Network nodes (former routers) are caches of content, indexes, and buffers. They cache and forward information, very much in the style of mobile ad-hoc, delay-tolerant, sensor networks, peer-to-peer systems and content delivery networks:

Peer-to-Peer (P2P) Most of the innovations in networking space during the last years have come from P2P systems (new routing algorithms based on distributed hash tables (DHT), swarming protocols, NAT traversal, overlay naming, etc). P2P provides an overlay solution to the networking shortcomings of the Internet in assisting current usage demands, which include support IP multicast, anycasting, content-based naming, efficient content distribution, full host reachability, and so on.

Instead of waiting for an overall infrastructure upgrade involving ISPs around the world, users have realized the power of deploying new services with a simple piece of software that turns personal computers into network elements, resulting in some of the most successful and scalable systems ever deployed like Skype, BitTorrent, Freenet, and more to come.

The beauty of P2P systems is that one can deploy hugely scalable services completely bypassing ISPs and without the need for end-to-end multicast, in a similar way that the Internet created a network that could route packets without having to go through the centralized control of telecom operators.

Future generation Internet architectures may consider adopting into their core design concepts and ideas from the P2P overlay (key-based routing, resource location algorithms, content naming, indexing, etc.). Recent work (VRR [131], ROFL [132]) has demonstrated how the ideas of structured routing overlays can be pushed down into the network layer, thereby potentially replacing IP with new, key-based (flat labels) routing protocols. A similar concept of key-based forwarding is label switching applied in several contexts. For instance, in mobile ad-hoc networks very short-lived local labels are used (LUNAR, Lilith), and in ISP backbones, long-lived local labels such as in MPLS or VLAN tags are common.

Content Delivery Network (CDN) CDNs emerged as an innovative technology to improve the efficiency of static, time-dependent, and rich media content delivery atop large-scale IP-based networks [133]. CDNs are based on smart URL names and DNS redirection services to resolve requests for data to the best candidate server taking into consideration the estimated user location and the observed network performance (e.g., shortest RTT). See [134] for a comprehensive survey and taxonomy of CDNs.

By enhancing the content retrieval experience to end-users through close-by storage capabilities and additional intelligence in the network (i.e., monitoring, enhanced control plane based on DNS resolution requests), CDNs constitute an ad-hoc overlay solution that try to close the gap of the original host-centric Internet design [126] and today's focus on data and service access. The problem of distributing the actual content served within a CDN or federation of CDNs has been extensively studied in a recent PhD dissertation [135]. The practical outcome of this work is called Coral, a free peer-to-peer content distribution network comprising a world-wide network of web proxies and name servers.

Peer-accelerated content delivery technologies like Bittorrent Distributed Network Accelerator (DNA)[3] merge the P2P and CDN worlds to efficiently deliver faster and more reliable downloads from multiple nearby sources in parallel. This way, peer-assisted content delivery combines the efficiency and scalability of peer networking with the control and reliability of traditional CDNs.

Wireless Sensor Networks (WSN) The nature and spirit of sensor networks point out the needs for data-centric approaches [136] instead of traditional address-centric approaches. Data-centric approaches consist in finding routes from multiple sources to a destination that allows in-network consolidation of data, where aggregation of multiple input packets into a single output packet is performed by en-route nodes. In this way, data aggregation may be performed to reduce data transmission by eliminating the redundancy. In the literature, this kind of approaches can be classified as reactive (require flooding of data queries in the entire network) and proactive (storing relevant data by name). Thus, a goal here is to allow queries for data with a particular name to be sent directly to the node storing that named data, instead of flooding the entire network.

Due to its service model, algorithms and mechanisms for content routing (epidemic, direct diffusion, greedy incremental trees, adaptive clustering hierarchy, etc.) may well suit the needs of content-oriented protocols.

Delay-Tolerant Network (DTN) The field of delay/disruption-tolerant network (DTN) [137, 138] looks at enabling communication in the absence of end-to-end connectivity or in the presence of links which are subject to long delays. The idea is to explore the fact that users are nowadays more and more equipped with wireless devices, and that users that are physically close are potential data exchangers. It seems then interesting to exploit the resources of any available wireless communication possibility to deliver and/or storage collected data in networks of intermittent connectivity.

Activities around the IETF DTN WG have consolidated onto several standards documents (e.g., RFC 4828 [138]), promoting content-oriented concepts like data message delivery, opportunistic transport, storage in the network edges, identity-based security and so on.

Publish/Subscribe (pub/sub) Publish/subscribe is a communication paradigm in which the interaction between the information producer (publisher) and consumer (subscriber) is mediated by a set of brokers. Publishers publish events (or publications) to the broker network, and subscribers subscribe to interesting events by submitting subscriptions to the broker network. It is

[3]http://www.bittorrent.com/dna/

the responsibility of the brokers to route each event to interested subscribers.

In content-based pub/sub systems, subscribers can specify constraints on the content of the events, and the broker network is said to perform content-based routing of events. These systems can efficiently deliver messages to large numbers of subscribers and is therefore considered an appropriate technology for large-scale, event-based applications.

In topic-based pub/sub, the basic unit of publication and subscription is a topic, identified by a unique identifier. From an architectural point of view, in a topic-based pub/sub a topic can be thought as an identifier of a channel. Whenever there are events related to the topic, information is delivered over the channel from the event source to the subscribers.

To improve the network layer performance in topic-based pub/sub systems, one approach is mapping pub/sub topics to IP multicast groups, so data can be directly sent to subscribers with a single message on the wire. However, this method, though network efficient, does not help to solve the scalability issues of IP multicast. In case of many concurrent active receiver groups, the routers are forced to maintain huge forwarding states due to the lack of aggregation.

While the application and scope traditional event-centric pub/sub systems differ from a global Internet-scale content-oriented paradigm, they share a core communication model. An attempt to formalize the pub/sub communication model has been presented in [139]. Surveys on the many applications and implementation options of pub/sub systems include [140] [141] and [142].

3.2.1 A new Networking Paradigm

In theory, there is no difference between theory and practice. But, in practice, there is.

— Jan L. A. van de Snepscheut / Yogi Berra.

In contrast to the traditional IP-centric model, content-oriented networking provides a new model for communication where the focus is on data, not nodes [22]. Hence the underlying networking substrate (i.e., network nodes and end hosts) becomes less relevant, possibly to an extent where it can be no longer based on IP-address-like names (cf. with [143]). The name of the node that hosts and converts the original information into the form in which it is finally delivered becomes optional or even dispensable, as long as the received data is timely and correct.

Content-oriented architectures go beyond trying to solve the *host reachability problem* by providing more flexible, expressive, and comprehensive naming and addressing frameworks (e.g., FARA, Plutarch, UIP, IPNL, HIP) mainly aimed at solving the shortcomings of the hierarchical, host-centric IP address space. This new research thread on *interconnecting information* can be observed in recent projects – under the future Internet umbrella – such as PSIRP [144], 4Ward [145], CCN [146], and other activities supported by EU and US funding agencies. As a result, the first content-oriented architectures have started to emerge (e.g., DONA [24], Haggle [147], RTFM [148]). Later in Sec. 3.2.3 we outline the principles of some remarkable proposals.

The unifying approach of service-centric [149] [150], data-oriented [24], content-centric [146], and information-centric networking [144] is to revolve around the data itself and to solve the problem of efficiently delivering a particular piece of data. Being a content-oriented network, the flow of messages is driven by the nodes that have expressed their interest and the information identifiers of the messages. Reachability of destinations is not any more delimited by topological boundaries but by new information-centric means, e.g., *scoped information* (cf. PSIRP vision in Section 3.2.3). Having the data location hidden makes the semantics of what defines a sender or receiver of data less relevant than the data itself, intuitively providing enhanced security (e.g., DoS mitigation) due to a receiver-controlled content-oriented type of communication – also well suited for connectivity challenged underlying networks.

The *publish/subscribe* paradigm [142] is a promising approach to implement a so-sought modern communication API [151] for information-centric systems. The suitability and benefits of moving the pub/sub layer downwards into the networking stack is one of the challenging objectives of content-oriented interest-driven architectures where naming, routing, forwarding and addressing get fresh semantics (see Table 3.1).

Tab. 3.1: Concepts of content-oriented networking versus the original Internet design. Source: [A]

Original Internet	Content-Oriented Networking
Sender	Content producer (publisher)
Receiver	Content consumer (subscriber)
Sender-based control	Receiver-based control
Client/Server communications	Publish/Subscribe Sender and Receiver uncoupled
Host-to-host	Service access / Information retrieval
Topology / Domain	Information scope
Unicast	Unified uni-, multi- and anycast
Explicit destination	Implicit destination
End-to-End (E2E)	End-to-Data (E2D)
Host name (look-up oriented)	Data/Content name (“search” activity)
Secure channels, host authentication	Integrity and trust derived from the data

3.2.2 Naming, Routing and Forwarding

Internet Protocol (IP) routing is based on the destination address inserted by the sender of the message. In contrast, a *content network* is a network that supports some form of content routing, i.e., messages are routed based on their contents rather than an explicit destination IP address appended by sending nodes to the messages.

Many types of content networks have been developed in various contexts such as P2P systems, cooperative Web caching, CDNs, publish/subscribe and content-based sensor networks. Kung *et al.* [152] propose a taxonomy for content networks based on their attributes in two dimensions: *content aggregation* (semantic vs. syntactic) and *content placement* (content-sensitive vs. content-oblivious). In the field of content delivery networks such as Akamai, HTTP request routing mechanisms can be classified according to the variety of request processing (cf. with the CDN taxonomy proposed by Pathan and Buyya [134]): Global Server Load Balancing (GSLB), DNS-based request-routing, HTTP redirection , URL rewriting, anycasting, and CDN peering.

A finer classification of a content network can be made based on how content is identified (granularity, naming, etc.) and the routing and forwarding approach (e.g., overlay, in-network matching, etc.). In general, routing and addressing in content-based networks are fundamentally different from traditional host-centric communication services and group-based multicast services, as shown in Table 3.2.

Tab. 3.2: Main characteristics of the unicast. multicast and content-based modes of communication services. Source: [153].

-	unicast	multicast	content-based
Destination specified by producer	explicit	explicit	implicit
Attribute of consumer used in routing	pre-assigned, unique identity	group identity	expression of interest in content
Information flow	directed	directed	emergent, indirected

For its service model, content-based networking can be related to a number of advanced network services and distributed-system technologies, including IP multicast, distributed publish/subscribe systems, other rendezvous-based communication services such as the Internet Indirection Infrastructure (i3) [116], intentional naming [154], XML routing [155], and basically any information system implementing some sort of message indirection based on the packets' content and not on explicit destination addresses. In some cases, depending on where and how the routing decision is made, content-based routing can also described as an example of *application-layer routing*. For instance, the term “content-addressable network” as proposed by Ratnasamy *et al.* [156] refers to a network providing a lookup service to map keys (i.e., resource identifiers such as file names) to values (usually locations).

As per Carzaniga *et al.* [153], the service provided by a content-based network consists of message delivery to all interested receivers whose predicates match the content of the message – instead of traditional numerical network addresses. Receivers declare their interests to the network by means of predicates, while senders simply inject messages into the network at the periphery. The network is responsible for delivering to each receiver any and all messages matching the selection predicate declared by that receiver. Hence, *content-based routing* supports a more powerful mechanism of routing messages based on message content to those destinations that are known to be interested in that content type. A *content-based routing protocol* maintains a *content-based forwarding table*. This type of content-based forwarding table maps predicates to interfaces, where a predicate associated with each interface represents the union of the predicates advertised by downstream reachable nodes.

Although not explicitly called that way, the problem of content-based routing has been studied quite extensively under multiple applications of distributed systems like publish/subscribe and event notification services [128, 157, 158, 159, 160].

The general problem of content-based routing has been characterized by Carzaniga *et al.* [161], providing a general framework on the structure of the routing state and the corresponding forwarding functions used to realize the algorithm within a particular content-based routing scheme. In content-based publish/subscribe systems, the message content is typically structured as a set of attribute/value pairs (AVPs), and a selection predicate is a logical disjunction of conjunctions of elementary constraints over the values of individual attributes. Routing schemes commonly rely on propagating predicates along the necessary topological information in order to forward the messages across the network to the interested users.

According to Carzaniga and Wolf [162], *content-based forwarding* (CBF) can be defined as a function of three inputs: a message m, a set of broadcast output interfaces B, and a content-based forwarding table $T = p_1, p_2, \ldots, p_I$, where I is the total number of interfaces. The function computes the subset of the broadcast output B that includes all the interfaces in T associated with a predicate matched by m. The focus of CBF lays on the *predicate-matching algorithm*, since this is the novel aspect of the forwarding function in content-based networks where the content of a message m contains a set of AVPs. Formally, CBF can be expressed as: $CBF(m, B, T) = i : i \in B \wedge matches(p_i, m)$

Content networks are usually overlayed on top of IP networks and are not intended as a replacement for IP or other traditional network services like unicast or multicast. Rather, it is intended to implement the specific communication style embodied in publish/subscribe middleware in a way that is superior to current approaches. Though, there is no conceptual obstacle to implement a native content-based networking stack, it is still to be proven whether it would be better (or even feasible) from an engineering standpoint, especially at Internet scales.

Scalability challenges

The major challenge faced by a content-based forwarding plane that takes port-forwarding decisions based on the content identifier (or a more expressive descriptor) is maintaining in-network state for every routable object in a way that the network scales and handles line-speed data rates.

Content-oriented networks can be seen as providing a *generalized multicast* communication service with the main difference that they do not rely on the existence or maintenance of group address spaces, but on the advertisement and propagation of available content based on an interest-driven routing service. Hence, content-based routing and IP multicast service models are similar in that they allow senders and receivers to communicate indirectly through a logical rendezvous point. There are however significant differences in their flexibility and the applicable forwarding strategies, depending on the nature of the content identifier, i.e., whether it is a structured name like a FQDN, a flat label resulting from a hash computation, or a less rigid structure, consisting of arbitrary sets of AVPs.

The basic communication scheme of topic-based publish/subscribe is functionally similar to IP-based source specific multicast (SSM) [163], with IP multicast groups replaced by content identifiers (i.e., topics). IP multicast [164] and topic-based pub/sub systems[165] use almost the same forwarding approaches in addressing more than one receiver in each connection. IP multicast typically creates lot of state in the network if one needs to support a large set of small multicast groups. An IP network is originally an unicast network with multicast as an additional service. However, in pure pub/sub based inter-networking, multicast is expected to be the native routing and forwarding approach.

Many efforts have been put to minimize the forwarding state problem by trying to aggregate state in the network [166] or moving some routing information to the packet headers as in Xcast [167]. Xcast source nodes encode the list of multicast channel destinations into the Xcast header. Each router along the way parses the header, partitions the destinations based on each destination's next hop, and forwards a packet appropriately until there is only one destination left where the Xcast packet is unicasted. In the PoMo architecture [168], the authors suggest a routing/forwarding solution that trades over-deliveries for reduced state and reduced dependence of node network locators. Their approach [169] employs link identities rather than network locators as the pivotal role.

When the content identifier is synthesized into a certain bit string, rather than a full content descriptor such as a XML or AVP schema, the problem of routing such content objects falls into the category of *name-based routing* and the means for aggregation depend on the nature (structure, form, semantics) of the content identifier/name.

Shue *et al*. [170] researched on general name-based versus IP-prefix based packet forwarding concluding that it is within the margins of feasibility. In this case, the name structure of a fully-qualified domain name (FQDN) is the key factor to enable the system scale to Internet-wide sizes. This should not come as a surprise since the scaling capabilities of name-based systems can be compared to the hierarchical IP-based aggregation, with IP subnets being replaced with domain names. However, such a strategy for name-based fast forwarding system is likely to face scalability issues if the granularity of content goes beyond naming hosts to individually naming pieces of content.

In the case of content identifiers generated as a result of hash-based method (e.g., a label resulting from a hash over the content itself), the routing problem falls into the category of *flat routing*, since the content identifier lacks of any topological information or network location semantics useful for routing. Related work aiming at routing on flat labels includes ROFL [132], a proposal for Internet-scale routing on flat host identifiers based on neat DHT constructs. VRR [131] applies the same core ideas on identity-routing in the field of wireless ad-hoc networks. i3 [116] separates the acts of sending and receiving by using a combination of packet identifiers (triggers) and a DHT. Receivers insert a trigger consisting of the data identifier and their network address into the DHT. Triggers reach an indirection point in DHT network and are then routed to the appropriate sender, who in turn satisfies the request by sending a packet containing the same identifier and the requested data. SEATTLE [171] utilizes flat addressing within Ethernet networks based on a one-hop DHT acting as a directory service for reactive address resolution and service discovery.

A fundamental difference of flat identifiers used in content-oriented networks include the different semantics and the contrasting architectural principles (cf. with Table 3.1). For instance, the end-to-data characteristics imply that the same piece of content may be reachable from different sources (i.e., advertised from different network locations), including caches embedded in the forwarding infrastructure. Moreover, if any of the above-mentioned systems were used for content routing, each piece of data would be required to be explicitly published in the DHT along its location before it can be retrieved, a hardly scalable approach considering the potential magnitude of content objects.

Content-Oriented Naming

Although the Internet is now widely used by users and applications to gain access to identifiable services and data, the Internet lacks of a mechanism for *directly* and *persistently* naming data and services. The DNS namespace does not accomplish this goal. DNS names are another example of semantic abuse – similar to IP addresses embodying both identification and location information. Today's DNS design overloads DNS names with multiple semantics (e.g., trademarking and web objects) and rigidly associates them with specific domains or network locations (i.e., in a host-centric manner), difficulting the movement and replication of service instances and data.

Handling information objects as first-class citizens introduces the need of a new, global content-oriented *namespace*. Closely related, in addition to new primitives, some form of *metadata* information is required to enable the self-authentication of the data, fragmentation, scope delimitation, inter-domain policies, in-network management, caching, and so on [144]. Such global namespace around data items enables *caching* capabilities for every type of communications. In comparison, caching over TCP/IP is costly and application-specific. In case of non-mutable information objects caching becomes trivial, whereas for streaming applications, caching can be seen as long in-network buffers. Hence, content-oriented architectures can natively play the role of current CDNs and promise avoiding redundant traffic over network links [172]. Furthermore, a new namespace for information objects could unify multi-, any-, con- and unicast types of communication in addition to enable novel forms of network coding to increase the network's efficiency and resilience.

Named Content Security

Content-based security compared to traditional channel-based security aims at allowing secure content retrieval by name and authentication regardless of where the content comes from. New security and network primitives are required that enable referring to, and authenticating content itself, rather than the host and file containers where it resides [173].

A common approach consists of self-certifying names (e.g., for hosts [28, 174] or content items [24, 175]) where the name itself is cryptographically constructed in a way that it allows to verify whether a given piece of content matches a given name. The simplest form of self-certification is hash-verified data, where the content names are the direct result of its cryptographic (e.g. SHA-1) digest [175, 176]. While this approach allows the receiver to assess validity (i.e. integrity plus authenticity) it does not help with regard to the provenance (i.e. the publisher trusted by the receiver as a content supplier) or relevance (i.e. the content answers the question the receiver asked) [173]. Some degree of provenance can be granted by means of key-verified names, where a piece of content is named with the digest of the public key used to sign that data [175, 177].

The main drawback of both approaches is requiring a secure indirection mechanism to map from the (human-understandable) names understood by users to the self-certifying name for a piece of content. In essence, the original content security problem turns into the problem of securing this mapping. If the mapping gets compromised, the user ends up with the secure, self-verifying name for a wrong or even malicious piece of content, falling back to a situation they tried to circumvent at first by relying on so-called "semantic-free" names [178, 179] and indirection architectures [119, 154].

According to the Zooko's Triangle [180], names can be simultaneously at most two of global, secure, and memorable. For instance, domain names are global and memorable, but as demonstrated by the rise of phishing issues, they are not secure. In contrast, self-certifying names are secure and unique but hard for humans to remember. Petname systems [181] advocate for building a naming system that dynamically translates between different possible kinds of names, i.e., the edges of Zooko's triangle.

The content-centric networking (CCN) approach [173] addresses the named content security issue by separating the roles of the human-relevant content *identifiers* and network-relevant *locators* from the security-critical *authenticator* used to assess the validity of the content. While the above-mentioned self-certifying naming schemes overload names with both roles simultaneously, CCN's approach to content naming is centered on authenticating the linkage between names and content rather than either names or content alone. Retrieved content can be authenticated regardless of how, or from whom, it is obtained, for any arbitrary (even human-readable) name form.

3.2.3 Architectural Proposals

Main Entry: architecture
Function: noun
1. the art and science of designing and superintending the erection of buildings and similar structures
2. a style of building or structure: Gothic architecture
3. buildings or structures collectively
4. the structure or design of anything: the architecture of the universe
5. the internal organization of a computer's components with particular reference to the way in which data is transmitted
6. the arrangement of the various devices in a complete computer system or network

— Collins English Dictionary - Complete Unabridged 10th Edition

In the following, we briefly review a series of remarkable architectural proposals towards enabling content itself to become a first-class object in the Internet.

Layered Naming Architecture (LNA)

The Layered Naming Architecture (LNA) by Balakrishnan *et al.* [119] exploits the introduction of new levels of indirection as a powerful concept to correct the semantic overloads of IP addresses and DNS names. LNA embodies a number of cumulative efforts in re-designing Internet naming and addressing. LNA proposes four layers of names: (1) user-level descriptors (e.g., search keywords, E-Mail identities), (2) service identifiers (SIDs), (3) endpoint identifiers (EIDs), and (4) IP addresses or other forwarding directives. Their goals include (i) making services and data first-class Internet objects, i.e., named independently from network location or DNS domains, (ii) enabling unfettered migration or replication in a mobile and multi-homed environment, and (iii) accommodating network-layer and application-layer intermediaries to be interposed as valid architectural components on the data path between communicating endpoints. The design principles of LNA include:

- Principle 1: Names should bind protocols only to the relevant aspects of the underlying structure; binding protocols to irrelevant details unnecessarily limits flexibility and functionality.
- Principle 2: Names, if they are to be persistent, should not impose arbitrary restrictions on the elements to which they refer.
- Principle 3: A network entity should be able to direct resolutions of its name not only to its own location, but also to the locations or names of chosen delegates.
- Principle 4: Destinations, as specified by sources and also by the resolution of SIDs and EIDs, should be generalizable to sequences of destinations.

The LNA has a strong focus on the naming architecture and disconsiders on purpose changes in forwarding plane due to the deployability issues of changing a large installed routing infrastructure. The bottom line is that a re-design restricted to the naming layer could render important benefits in an independent manner of data plane issues like denial-of-service protection, routing scalability and new services, or QoS enhancements. A new naming approach to enable innovation below the communication API has been argued by the work at UC Berkeley [182] on a NetAPI that lets applications specify communication intents without network-specific bindings as with today's Sockets API and its strong coupling to a destination IP address. Similarly, name-based sockets have been proposed by Vogt [183] to overcome Internet routing scalability issues while relieving applications from IP address management responsibilities.

TRIAD

TRIAD [121] proposes a new Internet architecture following the observation that today's Internet tries to solve what they define as the *content routing problem*. Their goal of *content routing* is to reduce the time needed to access content. This is accomplished by directing a client to one of many possible content servers; the particular server for each client is chosen to reduce round-trip latency, avoid congested points in the network, and prevent servers from becoming overloaded. These content servers may be complete replicas of a popular web site or web caches which retrieve content on demand. User clients express access not to a particular server or IP address but to some piece of content, specified by name (typically a URL).

The network-integrated content routing aims at providing support in the core of the Internet to distribute, maintain, and make use of information about content reachability. This is performed by content routers (CRs) acting as both conventional IP routers and name servers, i.e., participating in both IP routing and *name-based routing*. This integration forms the basis of the TRIAD content layer.

The proposed Name-Based Routing Protocol (NBRP) performs routing by name with a structure similar to BGP. Just as BGP distributes address prefix reachability information among autonomous systems, NBRP distributes name suffix reachability to content routers. Like BGP, NBRP is a distance-vector routing algorithm with path information; an NBRP routing advertisement contains the path of content routers toward a content server.

TRIAD performs content routing in the sense that routes on URLs by mapping URLs to next-hops. Actually, routing is done at the *granularity of server names* (e.g., FQDN) rather than full URLs. Routing is done as a longest-suffix match on FQDNs at gateways (firewalls/NATs) between realms and BGP-level routers between ASes (through theoretically every network router could act as a content-router).

Forwarding state is built up in intermediate content routers as packets are routed, and name suffix reachability is distributed among address realms as in BGP. TRIAD thus relies on name aggregation to scale, and will fail if object locations do not follow the DNS hierarchy closely. In order to overcome this issue, name-level redirection mechanisms are used to handle hosts whose names do not match network topology, becoming essentially a name-resolution mechanism. This observation also applies to IPNL [122], which also proposes some form of routing on FQDNs. TRIAD tackles thus the content-routing problem through a name-based routing approach, where names refer to FQDNs that form part of a larger content name like a full URI.

Data-Oriented Network Architecture (DONA)

DONA [24] explores an alternative content-based architecture, essentially allowing a client to request a piece of data by its name (a flat self-certifying label), rather than the owner's address. To do so, the architecture exposes two fundamental operational primitives, namely:

- FIND : allows a client to request a particular piece of data by its name (not its location).
- REGISTER: using this operation, content providers periodically indicate their intent to serve a particular data object.

To support these primitives, DONA introduces a new class of network entities, the Data Handlers (DHs), which combine the functions of name resolution and data caching. Collectively, DHs assume the responsibility for routing clients' requests to nearby copies of the data.

Basically, DONA replaces DNS names with flat, self-certifying names and a name-based anycast primitive above the IP layer. While names in DONA are a cryptographic digest of the publisher's key and a potentially user-friendly label. The lack of secure binding to the content opens the door to substitution attacks. Another limitation of DONA is that content must first be registered along a hierarchical tree of trusted resolution handlers (RHs) to enable retrieval. As a consequence RHs must maintain a large forwarding table to reach each next hop for every piece of content published in the network. After locating the content, data packets are exchanged using standard IP routing. While this data-oriented anycast approach over IP simplifies deployability, it does not solve many of the intrinsic problems of IP like security, routing table size exhaustion, or mobility. In case of a change of the location of a piece of content, new requests for that piece of content will remain unresolved until the new registration propagates through the network.

Content-Centric Networking (CCN)

Content-Centric Networking (CCN) [19] defines a novel communications architecture built on *named data* - a packet 'address' names content, not location. The new waist' of the stack (i.e., the traditional network layer) is now based on named data, and becomes the only layer requiring global agreement. Figure 3.3 compares the IP and CCN protocol stacks. Like in the traditional layers of the hourglass model of IP, layers of the stack reflect bilateral agreements (e.g., L2 framing protocol between ends of a physical link, L4 transport protocol between data producer and consumer). CCN preserves the design decisions that make TCP/IP simple, robust and scalable, including minimal demands on layer 2 (i.e., stateless, unreliable, unordered and best-effort delivery). As a consequence, CCN can be layered over anything, including IP itself.

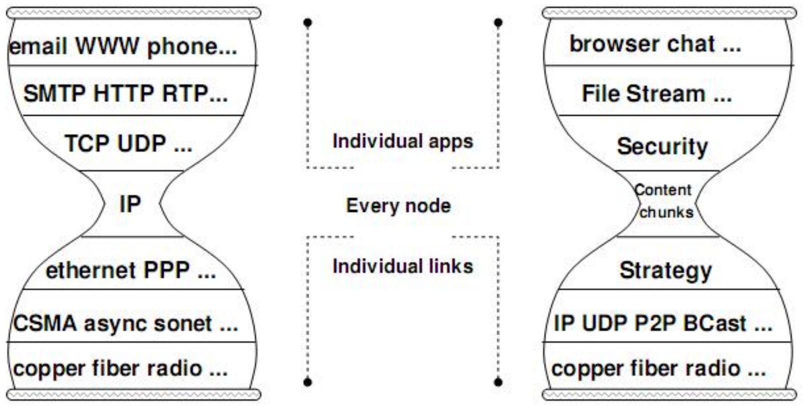


Fig. 3.3: CCN replaces the global component of the network stack (IP) with chunks of named content. Source: [19]

In addition to name-based routing and caching support, there are two critical aspects where CCN differs from IP, namely *strategy* and *security*, which appear as new layers in the protocol stack. The strategy layer is in charge of making fine-grained, dynamic optimization choices to best exploit multiple connectivities under changing conditions. Security in CCN [173] is based on the content itself, rather than the connections over which it travels.

Communications in CCN is receiver-driven in that a consumer asks for content by broadcasting its interest (content request) over all available interfaces. CCN defines two types of packets: *Interest* and *Data*. Any node hearing the interest and having the matching piece of data can respond with a Data packet. Hence, Data is transmitted only in response to an Interest and consumes that Interest packet - maintaining thus a strict flow balance similar to TCP data and acknowledgement packets.

CCN uses user-friendly, hierarchical names like URLs. Similar to P2P file swarming approaches, the original content is divided into multiple chunks. Each packet contains the content URL-like name along a chunk identifier (e.g., a hash over the chunk) and a signature that secures the binding between name and content chunk (i.e., a standard digital signature of name+content). The content publisher signs each chunk along with its name, enabling intermediate routers to (optionally) validate an incoming chunk using this per-chunk signature or letting this task to the end content consumer.

Content publishers announce content availability in the spirit of BGP prefix announcement or TRIAD name-based routing protocol. The basic operation of a CCN router is very similar to an IP router. Upon an Interest (or request) packet arrives on an interface, a longest-match look-up — in parallel to a local cache look-up — is done on its name to make a forwarding decision. If there are multiple servers announcing content availability, an interest packet is forwarded toward all these content sources.

CCN routers have three main data structures: (1) the FIB (Forwarding Information Base), (2) the Content Store (buffer memory), and the PIT (Pending Interest Table). The FIB is used to forward Interest packets toward potential source(s) of matching Data. It is almost identical to an IP FIB except it allows for a list of outgoing interfaces rather than a single one. Only Interest packets are routed toward potential Data sources using the FIB.

To forward Data packets back to the requester(s), the PIT is maintained by forming a sort of trail of ‘bread crumbs’. Entries in the PIT are removed as soon as they have been used to forward a matching Data packet, i.e., on Data packet consumes’ one Interest entry. A ‘soft-state’ model is used to time out PIT entries which never were consumed by the corresponding Data packets, letting the requesters responsible for re-issuing Interest packets – if they still want the Data.

The Content Store is (at least) the same as the buffer memory of an IP router but has a different replacement policy in that it can act as a cache to satisfy other requests (Interest packets) due to CCN packets being idempotent, self-identifying and self-authenticating. In contrast, point-to-point IP packets have no value after being forwarded to the destination and are thus discarded after forwarding.

When an Interest packet arrives over some interface, it will be first looked-up on the ContentStore (acting as cache) and served returned upon match, or forwarded towards the data source based on a longest-match lookup on the FIB. If another matching PIT entry exists, it is updated by adding the interface to the PIT entry's requesting interfaces list. Once the Interest packet hits a router with a fresh copy or it reaches a server which has advertised availability of the desired content, the actual Data packets are sent back following the chain of PIT entries all the way to the original requester(s).

CCN is not without challenges. Focusing on those around content-oriented routing and forwarding, one fundamental issue is how to scale a system with a potentially unlimited name space given the feasibility constraints of available in-network fast forwarding state. While the FIB to route interest packets towards the data sources can be arguably aggregated with requirements matching those of today's IP networks (cf. name-based vs. prefix-based packet forwarding [170]), the PIT represents a potential bottleneck for state, as it needs to hold enough entries (at least for a brief period of time in the order of one RTT) to forward data packets back to the requesters. This calls for research in suitable designs of high capacity packet buffers to hold the Content Store and the PIT – recalling that the full benefits of CCN would depend on enabling high capacity caches (together with appropriate policies and storage management). Novel fast forwarding engines are required that allow memory-efficient switching decisions based on variable-length, hierarchical names and support for fast updates of the forwarding tables upon name-based routing protocol changes, and, more critically, Interest/Data traffic. [184]

RTFM

The RTFM architecture [148] is one design choice to map the design principles of the PSIRP project [185]. PSIRP starts from a viewpoint where all network operations are based on named information items across all layers. It is assumed that each piece of information has a statistically unique name and that applications can request the network to deliver named information. Hence, the primary function of the network is to locate and deliver information rather than to locate hosts and arrange communications between them. The design principles of PSIRP are [20]:

- Principle A1: Everything is information: The architecture is based on information throughout all layers, including in particular the inter-networking one. An information item is defined as the simplest unit transmitted by the network and identify each with a rendezvous identifier (RId), a statistically globally unique identifier.

- Principle A2: Information is scoped: Information exists in a context called scope. This concept supports grouping information that is relevant to specific application domains, as well as reducing the space to be searched for a RId and the application of access control policies enforced by the scope. It therefore supports composition of information, enabling the mapping of higher-level concepts onto the concepts of information items and scopes. In order for information to be found it must reside in at least one scope, but is not limited to only one. Scopes themselves are also information. Their RIds are called scope identifiers (SId).

- Principle A3: Equal control: Publishing information is sender-controlled while retrieval of information is receiver-controlled, provided access has been granted. Thus, communication will not take place without both parties having agreed through a trusted party. With that, the architecture provides a balance of power between publishers and subscribers, offering a new set of network services that shifts the network from send-receive between endpoints to a publish-subscribe model of information.

The RTFM architectural proposal gets its name from the functional building blocks that are recursively applied. The *rendezvous* (R) is in charge of matching subscriptions to publications and information scoping. The *topology* (T) management creates and maintains (sub-optimal) delivery trees used for traffic forwarding, acting both pro-actively (optimization) and re-actively (on-demand). The *forwarding* (F) functions perform the actual datagram delivery based on label switching techniques. Finally, *mediation* (M) refers to the node-to-node physical data transmission.

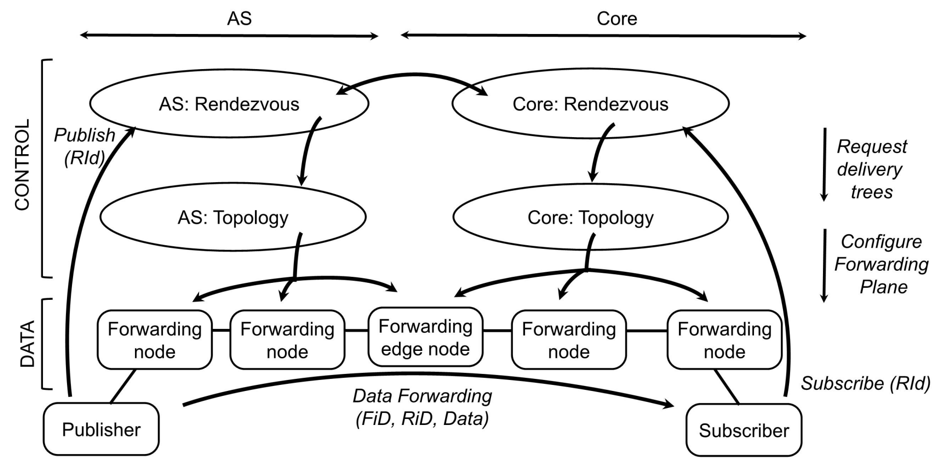


Fig. 3.4: High level overview of the RTFM architecture. Source: [D]

A high level operational overview of the RTFM is as follows (see Fig. 3.4). After a node subscribes to a publication, a distributed rendezvous system (e.g. a type of DHT or semi-hierarchical solution as in DONA [24]) must first find a copy of the publication's metadata. Using the distributed rendezvous structure to route to a copy of the wanted data, the topology management systems are expected to gather enough information to identify the delivery trees needed to forward the actual data to the subscriber(s). Note that the RTF functions are not necessary co-located in nodes and are distributed and recursive in nature.

Basically, this approach to content-oriented networking separates the act of routing content requests (subscriptions) –potentially over a 'slow path' in the spirit of DNS resolutions or DHT-based routing– from the act of delivering the actual data from convenient sources and at line rate through a so-called 'fast path'. The fast path data delivery can be then implemented by explicitly including forwarding directives (i.e., source-routing) to move the content packets from the sources to their destinations.

In the *black box* rendezvous based networking approach [186], the key idea is to regard the network as a collection of black boxes based on a set of recursive rendezvous functions. The boxes operate in trusted domains hiding their internal topology and exposing outwards only labels and interest definitions. Recursivity [81] and *scoped information layers* are pivotal architectural patterns with a major goal: *scalability*.

With the same goal of scalability but at a lower layer, efficient data structures enabling the content-oriented forwarding functions (e.g., switching, label processing, caching) are called for to achieve the challenging scalability requirements of information-oriented networks heavily based on a virtually 'unlimited' set of flat identifiers.

3.3 Networking with Probabilistic Data Structures

When the only tool you have is a hammer, then every problem begins to look like a nail.

— Abraham Maslow (1908 - 1970)

Advances in efficient packet forwarding techniques have been central to continuously moving traffic smoothly through the Internet at increasing rates. IP address lookup is one of the major bottlenecks in high-performance routers due to the challenges of the longest prefix matching operations. The speed and scalability of the IP lookup or packet classification schemes largely determines the performance of routers, and hence the Internet as a whole. Therefore, much work has been invested in data structures and algorithms for packet forwarding and classification [40, 43]. Research in the design of forwarding table compacting techniques has been a continuum since the early 90's [38, 39], and still goes on, yielding novel compact representations for structured graphs such as tries [42], new algorithms and data structures for IP lookups, packet classification and conflict detection [41], and advances in high-speed memory technologies among others.

Content-oriented network architectures – such as those discussed in the previous Section 3.2.3 – are characterized by introducing new namespaces for content objects. A common property of the proposed naming schemes is relying on flat identifiers (e.g., 256-bit hash values [148]) and/or long, non-fixed size URL-like names (e.g., TRIAD [121], CCN [19]) to uniquely identify single pieces of content. Other network architectures that separate identifiers from locators [29, 31, 123, 132] or aim at scalable Ethernet designs [171, 187], face similar challenges when handling packets carrying flat identifiers. A flat naming scheme simplifies address administration or content identification but is hard to scale due to the lack of aggregation capabilities. Hierarchical names of arbitrary sizes (e.g., CCN named data [19]) are also hard to handle at wire speed due to the challenges of performing LPM-like lookup operations on arbitrary long identifiers resulting from non-fixed size components.

Similar to the advances in algorithms and data structures that enabled the feasibility of high-performance IP routers, we surmise that new enablers in the forwarding plane may be fundamental to the realization of content-oriented networks. More specifically, we expect probabilistic techniques to play a key role to guide the construction of data structures well-suited for the requirements of packet forwarding in content-oriented networks.

The remainder of this Chapter provides an overview of probabilistic algorithms and data structures, their fundamental trade-offs, and examples of their application in the field of distributed systems and packet forwarding technologies.

3.3.1 Probabilistic Algorithms and Data Structures

Deterministic algorithms are commonly characterized by fundamental properties such as running time (speed) and memory usage (space). In practice, both attributes are in a trade-off relationship resulting in algorithmic solutions that require either low memory or a short running time, but not both simultaneously.

Probabilistic data structures are data structures based on algorithms that employ a certain degree of randomness in their operations. Probabilistic or randomized algorithms – and the companion data structures – introduce *correctness* as a new dimension in the traditional inter-play of fundamental properties (e.g., speed, memory). By inserting a certain amount of randomness into a given problem, lower memory usages and shorter running times can be obtained. For many difficult problems, if an approximate solution is acceptable, this relaxation on correctness is worth to pay and leads to practical implementations with (average) higher system performance and efficiency levels. In such cases, a randomized approach can be the simplest, the fastest, or both.

There is a broad field of applications where randomized algorithms outperform deterministic algorithms with provably high probability, examples of which include [188]: data structures (e.g., set membership, sorting, order statistics, searching), algebraic identities (e.g., polynomial and matrix identity verification, interactive proof systems), graph algorithms (e.g., minimum spanning trees, shortest paths, minimum cuts), counting and enumeration (e.g., matrix permanent counting combinatorial structures), parallel and distributed computing (e.g., deadlock avoidance, distributed consensus), probabilistic existence proofs, mathematical programming, etc.

Probabilistic data structures used to encode a collection of information are commonly termed as "lossy", as they usually require discarding some information. In information technology, lossy compression refers to as a data encoding method that reduces the size of the representation at the cost of some loose of fidelity at the time of decompressing. In exchange for losing data, lossy probabilistic data structures can store all information in constant space and respond to membership queries in constant time.

Under the specific application category of data structures, in addition to a myriad of variants relying on hashing techniques, popular probabilistic data structures include skip lists [189], Bloom filters [50], and lossy dictionaries [190]:

- A skip list [189] is probabilistic data structures, based on parallel linked lists, that uses probabilistic balancing rather than strictly enforced balancing. As a result of the randomized additions of links (following a geometric/negative binomial distribution), the insert and deletion operations in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees. The search performance of skip lists is restricted as only one key is stored per data structure node. Skip lists are an attractive data structure alternative for a variety of applications such as peer-to-peer distributed search, multiple predicate matching in database systems, high-dimensional approximate nearest neighbor search, and concurrent lock-free information retrieval.

- A Bloom filter [50] is a space-efficient probabilistic data structure to store an approximate representation of a set S. Each element is mapped to k bit positions in an m-bit array as determined by k independent hash functions over the element. Due to hash collisions, the set of elements is subject to an amount of *false positive* members, but all elements of S are included (i.e., no false negatives). False positives are those events for which the data structure claims an element to be in S even though it was not actually inserted.

- A lossy dictionary [190] is a probabilistic data structure that stores and retrieves key/value pairs in a space-efficient manner at the cost of two-sided errors. That is, in addition to false positives, lossy dictionaries are also subject to false negatives (claiming an inserted key not to be present). The false negative relaxation allows a very fast and simple data structure making almost optimal use of memory. The lossy encoding reduces the memory requirements for a set of data by changing the encoding characteristics of the data, introducing an intuitive trade-off between the quality of the lossy data and how much space is required.

Relying on hash functions to compress the information into a new, compact data space is the common denominator of Bloom filters, lossy dictionaries, and other probabilistic data structure generalizations. One way to efficiently transform a string of data into a probabilistic data structure is by using a hash function to generate a reproducible signature (i.e., fingerprint) of each piece of the data. The fingerprint of each item can then be stored in a smaller space. A hash function maps elements from a universal set $S \subseteq U$ of size w to a smaller domain of size b. If the set S contains many elements and $b \ll w$, elements of S will be allocated to the same sub-space in the new domain, leading to what is commonly known as a collision. To minimize collisions it is important that hash functions uniformly distribute the values of S into the smaller domain.

While both the Bloom filter and the lossy dictionaries allow set membership queries of the sort, "*Is* $x \in S$?", only the lossy dictionaries support value retrievals. The caveat of lossy dictionaries is the occurrence of false negatives. While false negatives may be tolerable for some applications (e.g., distributed caching systems) other applications require a strict policy of always returning the correct answer to the inserted elements (i.e. no false negatives) plus a certain rate on false positives. This is the case of packet forwarding applications that express the forwarding algorithm as set membership problem. In probabilistic port-forwarding algorithms of the sort "*is packet x in port P?*" or "*is link l in route R?*", a false negative would compromise the packet delivery to the intended destination(s), while false positives only cause some packets being unnecessary duplicated along their way to the destination.

In the following, we describe in more detail the Bloom filter probabilistic data structure and review its main variants and applications in the field of routing and forwarding.

3.3.2 Bloom Filters

Due to its simplicity and wide applicability, Bloom filters have become very interesting objects of study and a daily aid of system implementations. Burton H. Bloom's 40-year-old data structure [191] is beloved by theoreticians due to the mathematics that underpin the randomized flipping of 0s into 1s, and is beloved by practitioners as a powerful ally when aggregating data sets. Bloom filters turn resource-intense (memory, computation) operations into simple, resource-friendly set membership problems. The Bloom domain [60, 192] spans from hardware implementations, all the road up the system stack to the software application domain, where it first saw the light to perform space- and time-efficient dictionary lookups [191]. Broder and Mitzenmacher have coined the Bloom filter principle [60]:

> Whenever a list or set is used, and space is at a premium, consider using a Bloom filter *if* the effect of false positives can be mitigated.

The basic operations involve adding elements to the set and querying for element membership based on the state (1 or 0) of bit positions as determined by the outputs of a number of independent hash functions over the element. The basic Bloom filter does not support the removal of elements; however, a number of extensions have been developed that also support removals. The accuracy of a Bloom filter depends on the size of the filter, the number of hash functions used in the filter, and the number of elements added to the set. The more elements are added to a Bloom filter, the higher the probability that the query operation reports false positives.

A Bloom filter is an array of m bits for representing a set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements. Initially all the bits in the filter are set to zero. The key idea is to use k independent hash functions, $h_i(x), 1 \leq i \leq k$ to map items $x \in S$ to a random number uniform in the range $1, \ldots m$. An element $x \in S$ is inserted into the filter by setting the bits $h_i(x)$ to one for $1 \leq i \leq k$. Conversely, y is assumed a member of S if the bits $h_i(y)$ are set, and guaranteed not to be a member if any bit $h_i(y)$ is not set.

Figure 3.5 presents an overview of a Bloom filter of length 32 where three elements ($n = 3$) have been inserted, namely x, y, and z. Each of the elements is hashed using $k = 3$ hash functions to bit positions in the bitstring. The corresponding bits are set to 1. Now, when an element not in the set, w, is looked up, it will be hashed using the same three hash functions into bit positions. In this case, one

Tab. 3.3: Parameters of the Bloom filter data structure.

Parameters	Increase
Number of hash functions (k)	More computation, lower false positive rate as $k \rightarrow k_{opt}$
Size of filter (m)	More space is needed, lower false positive rate
Number of elements in the set (n)	Higher false positive rate

of the positions is zero and hence the Bloom filter reports correctly that the element is not in the set. It may happen that all the bit positions of a not inserted element report that the corresponding bits have been set. When this occurs, the Bloom filter will erroneously claim that the element is a member of the set. These erroneous reports are called *false positives*. We observe that for the inserted elements, the hashed positions correctly report that the element bits are set in the bitstring.

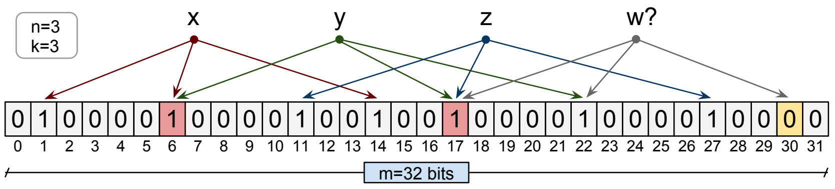


Fig. 3.5: Overview of the Bloom filter probabilistic data structure.

For optimal performance, each of the k hash functions should be a member of the class of universal hash functions, which means that the hash functions map each item in the universe to a random number uniform over the range. The development of uniform hashing techniques has been an active area of research. An almost ideal solution for uniform hashing is presented in [193]. In practice, hash functions yielding sufficiently uniform distributed outputs, such as MD5 or CRC32, are useful for most probabilistic filter purposes [194]. For candidate implementations, see the empirical evaluation of 25 hash functions by Henke *et al.* [195]. For further details, deeper theoretical foundations, and system-specific applications, we refer to related work, such as [44, 45, 195, 196].

A Bloom filter constructed based on S requires space $O(n)$ and can answer membership queries in $O(1)$ time. Given $x \in S$, the Bloom filter will always report that x belongs to S, but given $y \notin S$ the Bloom filter may report a false positive claiming $y \in S$.

Table 3.3 examines the behaviour of three key parameters when their values are either decreased or increased. Increasing or decreasing the number of hash functions towards k_{opt} can lower false positive ratio while increasing computation in insertions and lookups. The cost is directly proportional to the number of hash functions. The size of the filter can be used to tune the space requirements and the false positive rate (fpr). A larger filter will result in fewer false positives. Finally, the size of the set that is inserted into the filter determines the false positive rate.

One noteworthy property of Bloom filters is that the false positive performance depends only on the bit-per-element ratio (m/n) and not on the form or size of the hashed elements. As long as the size of the elements can be bounded, hashing time can be assumed to be a constant factor. Considering the trend in computational power versus memory access time, the practical bottleneck is the amount of (slow) memory accesses rather than the hash computation time. Whenever a filter application needs to run at line speed, resource-friendly and hardware-amenable per-packet operations become critical [44].

3.3.3 Improved Bloom Filters

The basic Bloom filter design has been matter of multiples extensions and enhancements, resulting in many shapes and forms, optimized for the specific application requirements and the underlying trade-off between size and accuracy. Making this choice and optimizing the parameters for the expected uses cases are fundamental factors to achieve the desired performance in practice.

There is a large body of work [192] on extended Bloom filter designs aiming at addressing specific functional concerns regarding space and transmission efficiency, false positive rate, dynamic operation in terms of increasing workload, dynamic operation in terms of insertions and deletions, counting and frequencies, popularity-aware operation, and mapping to elements and sets instead of simple set membership tests among others. Table 3.4 summarizes the distinguishing features of the Bloom filter variants proposed in the literature. Each variant can be classified by the output type (e.g., boolean, frequency, value), and whether counting (C), deletion (D), or popularity-awareness (P) are supported (Yes/No/Maybe), in addition to whether false negatives (FN) are introduced. Bloom filter variants with counting capabilities can also be used to probabilistically encode arbitrary functions by considering the cardinality of each set element being functional value and each set element being a variable.

Related work on enhanced Bloom filter designs aiming for better false positive rates include the Power of Two Choices filter [62] and the Partitioned Hashing [197], which combine the power of choices at hashing time to improve the performance of Bloom Filters. The main idea of [62] is for each element to choose one of c sets of hash functions so that the number of ones in the filter is reduced. The penalty in the on-line scenario is that each query has to hash using all c sets of hash functions and will return a positive if any one of these c sets of hash functions return true. The authors have shown that in low bits per element scenarios the benefits of the power of choices are not noticeable. On the other hand, [197] reduces the false positive probability by a careful choice of the group of hash functions that are well-matched to the complete set of input elements. The caveats of this scheme are that it is not well suited for dynamic sets and not practical in distributed environments where the hash functions need to be the same in each distributed instance. Similarly, the

Best-of-N method [63] proposes a standalone Bloom filter design where N candidate Bloom filters are constructed using different hash functions for a given set of elements. Then, the candidate with least amount of bits set is chosen to be deployed.

Bloom filter derivatives marked as popularity-aware share the approach of tuning the Bloom filter parameters (e.g., number of hash functions k) in accordance to the expected or observed element queries, optimizing thus the false positive rate for the most queried items. The Retouched Bloom filter [198] introduces a new tradeoff that achiever better false positive rates at the cost of introducing false negatives.

Since there is none Bloom filter that fits all, one key question that application designers should ask is whether false negatives are tolerable or not. Relaxing this constraint can help drastically in reducing the overall false positive rate (cf. retouched Bloom filters [198]), but raises also the question whether the Bloom filter is the right data structure choice despite alternative designs specific to the application domain (cf. [199]), approximate dictionary-inspired approaches [200, 201], cache-efficient variants (blocked Bloom filter) and Golomb coding implementations as proposed by Putze *et al.* [202], space-efficient versions of cuckoo hashing [203], and more complex but space-optimal alternatives [200, 204].

Each Bloom filter variant or replacement introduces a specific trade-off involving execution time, space efficiency, functionality, etc. Ultimately, which probabilistic data structure is best suited depends a lot on the application specifics. Indeed, the variations of the standard Bloom filter are commonly the result of specific requirements of network and distributed system applications.

3.3.4 Routing and Forwarding Applications

Bloom filters have been around in systems applications since 1970 when it was first proposed as a compact probabilistic data structure to represent words in a dictionary. However, interest for networking applications emerged only around 1995, after which this area has gained widespread attraction from both academia and industry. Today, Bloom filters are one of the most popular data structures, with manyfold applications in distributed environments such as P2P networks, Web proxies, caches and database servers, and hardware implementations enabling resource-efficient network functions such as IP look-ups, packet classification, measurement, security, and so on.

Bloom filters can be a powerful tool whenever you have a set of elements and space is an issue (e.g., high speed memories, packet headers). Since the false positive performance of hash-based data structures like the Bloom filters does not depend at all on the nature of the elements (i.e., size, structure, type) but on the ratio $memory/elements$, they become an appealing environment to deal with large sets of identifiers that may be too long, unstructured (i.e., non-aggregatable), or both.

In the remainder of the section, we focus on remarkable uses of Bloom filters (and its main vari-

Tab. 3.4: Key features of the Bloom filter variants, including the additional capabilities: Counting (C), Deletion (D), Popularity-awareness (P), False-Negatives (FN), and the output type. Source: [192]

Filter	**Key feature**	**C**	**D**	**P**	**FN**	**Output**
Standard Bloom filter	Is element x in set S?	N	N	N	N	Boolean
Adaptive Bloom filter	Frequency by increasing number of hash functions	Y	N	N	N	Boolean
Bloomier filter	Frequency and function value	Y	N	N	N	Freq., $f(x)$
Compressed Bloom filter	Compress filter for transmission	N	N	N	N	Boolean
Counting Bloom filter	Element frequency queries and deletion	Y	Y	N	M	Boolean or freq.
Decaying Bloom filter	Time-window	Y	Y	N	N	Boolean
Deletable Bloom filter	Probabilistic element removal	N	Y	N	N	Boolean
Distance-sensitive Bloom filters	Is x close to an item in S?	N	N	N	Y	Boolean
Dynamic Bloom filter	Dynamic growth of the filter	Y	Y	N	N	Boolean
Filter Bank	Mapping to elements and sets	Y	Y	M	N	x, set, freq.
Generalized Bloom filter	Two set of hash functions to code x with 1s and 0s	N	N	N	Y	Boolean
Hierarchical Bloom filter	String matching	N	N	N	N	Boolean
Memory-optimized Bloom filter	Multiple-choice single hash function	N	N	N	N	Boolean
Popularity conscious Bloom filter	Popularity-awareness with off-line tuning	N	N	Y	N	Boolean
Retouched Bloom filter	Allow some false negatives for better false positive rate	N	N	N	Y	Boolean
Scalable Bloom filter	Dynamic growth of the filter	N	N	N	N	Boolean
Secure Bloom filters	Privacy-preserving cryptographic filters	N	N	N	N	Boolean
Space Code Bloom filter	Frequency queries	Y	N	M	N	Frequency
Spectral Bloom filter	Element frequency queries	Y	Y	N	M	Frequency
Split Bloom filter	Set cardinality optimized multi-BF construct	N	N	N	N	Boolean
Variable-length Signature filter	Popularity-aware with on-line tuning	Y	Y	Y	Y	Boolean
Weighted Bloom filter	Assign more bits to popular elements	N	N	Y	N	Boolean

ants) in routing and forwarding tasks. These cases include IP lookups, loop and duplicate detection, forwarding engines, deep packet scanning, publish/subscribe systems and multicast.

Forwarding Engines

Bloom filters can be applied in various parts in a routing and forwarding engine. Probabilistic techniques have been used for efficient IP lookups. IP routers forward packets based on their address prefixes. Each prefix is associated with the next hop destination. CIDR-based routing and forwarding uses the longest prefix match for finding the next hop destination. This is commonly solved using a binary search, a trie search, or a TCAM. IP lookups can be made more efficient by dividing the addresses into tables based on their length and then utilizing binary search to find the longest common prefix. The d-left hashing technique has been used to make this lookup more compact and efficient [205].

Another example on probabilistic structures developed for fast packet forwarding, is an algorithm [206] that uses Bloom filters for Longest Prefix Matching (LPM). The algorithm performs parallel queries on Bloom filters, to determine address prefix membership in sets of prefixes sorted by prefix length. This work indicates that Bloom filter–based forwarding engines can offer favorable performance characteristics compared to TCAMs used by many routers. The main idea is to have different regular Bloom filters for different address prefixes. These Bloom filters are implemented in hardware and updated by a route computation process. The route manager uses counting Bloom filters to keep track of how the regular Bloom filters should be instrumented.

Asymmetric Bloom filters that allocate memory resources according to prefix distribution have been proposed for LPM. By using direct lookup array and Controlled Prefix Expansion (CPE), worst-case performance is limited to two hash probes and one array access per lookup. Performance analysis indicates that average performance approaches one hash probe per lookup with less than 8 bits per prefix [206]. The system employs a set of W Counting Bloom Filters where W is the length of input addresses, and associates one filter with each unique prefix length. A hash table is also constructed for each distinct prefix length. Each hash table is initialized with the set of corresponding prefixes, where each hash entry is a (prefix, next hop)–pair.

In enterprise and data center networks, the scalability of the data plane has become increasingly challenging with the growth of forwarding tables and link speeds. Simply building switches with larger amounts of faster memory is not appealing, since high-speed memory is both expensive and power hungry. Implementing hash tables in SRAM is not appealing either because it requires significant over-provisioning to ensure that all forwarding table entries fit. The BUFFALO architecture [187] proposes Bloom filters stored in a small SRAM to compress the information of the addresses associated with each outgoing link. Leveraging the flattening of IP addresses and the shortest-path routing, BUFFALO proposes a practical switch design that gracefully handles false positives without reducing the packet-forwarding rate, while guaranteeing that packets reach their destinations with bounded stretch with high probability. Routing changes are handled by dynamically adjusting the filter sizes based on Counting Bloom Filters stored in slow memory.

Bloom filters have been used to improve network router performance [207]. Song *et al.* used a Counting Bloom Filter to optimize a hash table used in network processing, such as maintaining per-flow context, IP route lookup, and packet classification. The small, on-chip Bloom filter eliminates slow, off-chip lookups when the searched flow is not found, and minimizes the number of lookups required when the flow is found. This is done by associating a hash table bucket with each Bloom filter counter. The bucket associated with the counter with the lowest value and lowest index is then always accessed, and the corresponding item is stored in that bucket. Counters are also artificially incremented to eliminate collisions. This leads to one worst-case off-chip lookup for flows stored.

Bloom filters can also be used in multicast forwarding engines. A multicast packet is sent through a multicast tree. A multicast router maps an incoming multicast packet to outgoing interfaces based on the multicast address. Initially, Grönvall suggests an alternative multicast forwarding technique using Bloom filters [208]. In this technique, a router has a Bloom filter for each outgoing interface. The filters contain the addresses associated with the interfaces. When a multicast packet arrives on one interface, the Bloom filters of each outgoing interface are checked for matches. The packet is forwarded to all matching interfaces. This technique is interesting, because it does not store any addresses at the router; however, the addition and removal of multicast addresses requires that the Bloom filters are updated, for instance, by adapting the parameters (m, n, k) and/or by relying on a Bloom filter variant that supports deletions.

Another approach to support multicast is to move state from the network elements to the packets themselves in form of Bloom filter–based representations of the multicast trees. This notion has been exploited by Ratnasamy *et al.* in “Revisiting IP multicast” [55]. The authors propose source border routers to include an 800-bit Bloom-filter-based shim header (`TREE_BF`) in packets. TREE_BFs represent AS-level paths of the form $AS_a : AS_b$ in the dissemination tree of multicast packets. Moreover, a second type of Bloom filters is used to aggregate active intra-domain multicast groups piggybacked in BGP updates. The presented method uses standard IP-based forwarding mechanisms enriched with the built-in TREE_BF to take the inter-domain forwarding decisions.

The BANANAS framework [209] for explicit and multipath routing in the Internet is based on the observation that a path can be encoded as a short hash (PathID) of a sequence of globally known identifiers. This per packet dynamic state can be compressed into a hash or a Bloom filter, with optional soft-state information computed by intermediate routers.

Bloom filters can be used for loop detection in network protocols. IP uses the Time-To-Live

(TTL) field to detect and drop packets that are in a forwarding loop. The TTL counter is incremented for each network hop. For small loops, TTL may still allow a substantial amount of looping traffic to be generated. Icarus [210] is a system that uses Bloom filters for preventing unicast loops and multicast implosions. The idea is straightforward, namely to use a Bloom filter in the packet header as a probabilistic loop detection mechanism. Each node has a corresponding mask that can be ORed with the Bloom filter in the header of a packet, and then determine whether or not a loop has occurred. Detection accuracy can be traded off against space required in the packet header.

Content-Based Routing

As discussed in Section 3.2.2 on the problem of content routing [161], the content-based publish/subscribe paradigm for system design offers unique benefits for many data-intensive applications. Coupled with peer-to-peer technology, it can serve as a central building block for developing data-dissemination applications deployed over a large-scale network infrastructure. A key open problem in creating large-scale content-based pub/sub infrastructures relates to efficiently and accurately matching subscriptions with various predicates to incoming events [211].

A Bloom filter-based approach has been proposed for general content-based routing with predicates [212]. Achieving expressive and efficient content-based routing in publish/subscribe systems is a difficult problem. Traditional approaches prove to be either inefficient or severely limited in their expressiveness and flexibility. The routing method based on Bloom filters shows high efficiency while simultaneously preserving the flexibility of content-based schemes. The resulting implementation contributes to a fast, flexible and fully decoupled content-based publish/subscribe system.

Bloom filters and additional predicate indices were used in a mechanism to summarize subscriptions [213, 214]. An Arithmetic Attribute Constraint Summary (AACS) and a String Attribute Constraint Summary (SACS) were used to summarize constraints, because Bloom filters cannot directly capture the meaning of other operators than equality. The subscription summarization is similar to filter merging, but it is not transparent, because routers and servers need to be aware of the summarization mechanism. In addition, the set of attributes needs to be known a priori by all brokers and new operators require new summarization indices. The benefit of the summarization mechanism is improved efficiency, since a custom-matching algorithm is used that is based on Bloom filters and the additional indices.

The authors of [215] introduce a novel approximate method for XML data filtering, in which a group of Bloom filters represented a routing table entry and filtered packets according to XPath queries encoded to it. In this method, millions of path queries can be stored efficiently. At the same time, it is easy to deal with the change of these path queries. Performance is improved by using Prefix Filters to decrease the number of candidate paths. This Bloom filter-based method takes less time to build a routing table than an automaton-based method. The method has a good performance with acceptable fpr when filtering XML packets of relatively small depth with millions of path queries.

Deep Packet Scanning and Packet Classification

Bloom filters have found applications also in deep packet scanning, in which applications need to search for predefined patterns in packets at high speeds. Bloom filters can be used to detect predefined signatures in packet payloads. When a suspect packet is encountered, it can then be moved for further investigation. One advantage of Bloom filters is that they can be efficiently implemented in hardware and parallelized [216, 217, 218], which can result in high-performance and energy-efficient operation.

Packet classification continues to be an important challenge in network processing. It requires matching each packet against a database of rules and forwarding the packet according to the highest priority matching rule. Within the hash-based packet classification algorithms, an algorithm that is gaining interest is the tuple space search algorithm that groups the rules into a set of tuple spaces according to their prefix lengths. An incoming packet can now be matched to the rules in a group by taking into consideration only those prefixes specified by the tuples. More importantly, matching of an incoming packet can now be performed in parallel over all tuples. Within these tuple spaces, a drawback of utilizing hashing is that certain rules will be mapped to the same location, also called a collision. The negative effect of such a collision is that it will result in multiple memory accesses and subsequently longer processing time. The authors of [219] propose a pruned Counting Bloom Filter to reduce collisions in the tuple space packet classification algorithm. The approach decreases the number of collisions and memory accesses in the rule set hash table in comparison to a traditional hashing system.

The storage requirements of the well-known crossproduct algorithm used in packet classification can be significantly reduced by using on-chip Bloom filters. For packets that match p rules in a rule set, a proposed algorithm requires $4 + p + e$ independent memory accesses to return all matching rules, where e is a small constant that depends on the false positive rate of the Bloom filters [220].

While the problem of high-performance packet classification has received a great deal of attention in recent years, the research community has yet to develop algorithmic methods that can overcome the drawbacks of TCAM-based solutions. A hybrid approach, which partitions the filter set into subsets that are easy to search efficiently, is introduced in [221]. The partitioning strategy groups filters that are close to one another in tuple space, which makes it possible to use information from single-field lookups to limit the number of subsets that must be searched. Running time can be traded off against space consumption by adjusting the coarseness of the tuple space partition. The authors find that for two-dimensional filter sets, the method finds the best-matching filter with just four hash probes while limiting the memory space expansion factor to about 2. They also introduce a novel method for Longest Prefix Matching (LPM), which is used as a component of the overall packet classification algorithm. The LPM method uses a small amount of on-chip memory to speed up the search of an off-chip data structure, but uses significantly less on-chip memory than earlier methods based on Bloom filters.

Efficient and compact state representation is needed in routers and other network devices, in which the number and behaviour of flows needs to be tracked. The Approximate Concurrent State Machine (ACSM) approach was motivated by the observation that network devices, such as NATs, firewalls, and application level gateways, keep more and more state regarding TCP connections [52]. The ACSM construction was proposed to track the simultaneous state of a large number of entities within a state machine. ACSMs can return false positives, false negatives, and 'do not know' answers. Their construction follows the Bloom filter principle and proposes a space-efficient fingerprint compressed d-left hash table design.

Security

The hashing nature of the Bloom filter makes it a natural fit for security applications. Spafford (1992) was perhaps the first person to use Bloom filters to support computer security. The OPUS system [222] uses a Bloom filter which efficiently encodes a wordlist containing poor password choices to help users choose strong passwords. Two years later, Manber and Wu [223] presented two extensions to enhance the Bloom-filter-based check for weak passwords.

The privacy-preserving secure Bloom filters by Bellovin and Cheswick [224] allows parties to perform searches against each other's document sets without revealing the specific details of the queries. The system supports query restrictions to limit the set of allowed queries.

Bloom filters have been used by Aguilera *et al.* [225] to detect hash tampering in a network-attached disks (NADs) infrastructure. Also in the field of forensic filesystem practices, the *md5bloom* manipulation tool [226] employs Bloom filters to efficiently aggregate and search hashing information, demonstrating its practicality of identifying object versioning in Linux libraries.

Attig, Dharmapurikar and Lockwood [227] describe an FPGA implementation of an array of Bloom filters and a hash table used for string matching to scan malicious Internet packets. The system searches 25 Bloom filters with string signature lengths from 2 to 26 bytes in parallel. False positives are resolved by exact match search using the hash table. Matches generate UDP packets that notify the user, a monitoring process, or a network administrator.

Antichi *et al.* [228] used Counting Bloom Filters to detect TCP and IP fragmentation evasion attacks. Attack signatures were split to 3-byte substrings which were inserted into a CBF. One CBF per attack signature string per flow was used. Incoming fragmented packet data was then matched against the CBF's and attack substrings detected. Each substring detected was removed from the corresponding CBF. Corresponding full string matchers were also enabled when a substring was detected. When the CBF was empty to the degree α, the attack string was considered detected, and the full string matcher was used to check for false positives. In case the full string matcher detected the attack, the flow was blocked. The authors report a greater than 99% detection rate and false positive ratios of 1% or less.

Bloom filters are used in the Trickles stateless network stack and transport protocol for preventing replay attacks against servers [105]. Two Bloom filters of identical size and using the same family of hash functions are used to simplify the periodic purge operation. The counting variant (CBF) is used in [229] to provide a lightweight route verification mechanism that enables a router to discover route failures and inconsistencies between advertised Internet routes and the actual paths taken by the data.

Focusing on the distributed denial-of-service (DDoS) issues, Ballani *et al*. [230] were among the first to use in-network Bloom filters to pro-actively filter out attacks, allowing each host to explicitly declare to the network routing infrastructure what traffic it wants routed to it. In addition to performing the standard longest-prefix match before forwarding packets, a router performs a reachability check using Bloom filters. Similar in their reliance on Bloom filters, Phalanx [231] combines the notion of capabilities with a multi-path-aware overlay, implementing Bloom filters to reduce state requirements while still providing probabilistic guarantees for in-network security. Wang *et al.* [232] propose *congestion puzzles* to mitigate bandwidth-exhaustion attacks. Congested routers challenge clients to generate hashes that match certain criteria in order to obtain bandwidth. Basic Bloom filters are maintained at routers to detect duplicate solutions.

In [233], Wolf presents a mechanism where packet forwarding is dependent on credentials represented as a packet header size Bloom filter. Credentials are issued by en-route routers on flow initiation and later verified on a packet-basis.

In wireless sensor networks (WSNs), a typical attack by compromised sensor nodes consists of injecting large quantities of bogus sensing reports, which, if undetected, are forwarded to the data collector(s). The statistical en-route filtering approach [57] proposes a detection method based on a Bloom filter representation of the report generation (collection of keyed message authentications), that is verified probabilistically and dropped en-route in case of incorrectness. In order to address the problem of multiuser broadcast authentication in WSNs, Ren *et al*. [234] propose a neat integration of several cryptographic techniques, including Bloom filters, the partial message recovery signature scheme and the Merkle hash tree.

The current Internet architecture allows a malicious node to disguise its origin during denial-of-service attacks with IP spoofing. A well-known solution to identify these nodes is IP traceback. The main types of traceback techniques are (1) to mark each packet with partial path information probabilistically, and (2) to store packet digests in the form of Bloom filters at routers and reconstruct attack paths by checking neighboring routers iteratively.

The Source Path Isolation Engine (SPIE) [235] implements a packet attribution system, in which the system keeps track of incoming and outgoing packets at a router. Simply storing all the resulting information is not feasible. Therefore, Snoeren *et al.* proposed to use Bloom filters to reduce the state requirements. A Bloom filter stores a summary of packet information in a probabilistic way. One key observation is that each router maintains its own Bloom filters and thus their hash functions are independent. A SPIE-capable router creates a packet digest for every packet it processes. The digest is based on the packet's non-mutable header fields and a prefix of first 8 bytes of the payload. These digests are then maintained by a network component for a predefined time.

When a security component, such as an intrusion detection system, detects that the network is under attack, it can use SPIE to trace the packet's route through the network to the sender. A single packet can be traced to its source given that the routers on the route still have the packet digest available. A false positive in this setting means that a packet is incorrectly reported as having been seen by a router. When the source of a packet is traced, false positives mean that the reverse path becomes a tree (essentially branches to multiple points due to false positives).

The notion of packet attribution was extended to payload attribution by Shanmugasundaram *et al.* [236] with the Hierarchical Bloom filter. This probabilistic data structure allows the query of a part of a string. SPIE uses the non-mutable headers and a prefix of the payload, whereas with Hierarchical Bloom filters it is sufficient to have only the payload to perform a traceback.

The key idea of the IP traceback [237] is to sample only a small percentage (e.g., 3%) of the digests of the sampled packets. Relying on a low sampling rate is critical to relax the storage and computational requirements and allow link speeds to scale to OC-192 or higher rates.

The Generalized Bloom filter (GBF) [58] was conceived to address single-packet IP traceback in a stateless fashion by probabilistically encoding a packet's route into the packets themselves. The key feature of the GBF is the double set of hash functions to set and reset bits hop-by-hop, which provides built-in protection against Bloom filter tampering at the cost of some false negatives.

Counter braids [238] revisits the problem of accurate per-flow measurement. The authors present a counter architecture, called Counter Braids, inspired by sparse random graph codes. In a nutshell, Counter Braids "compresses while counting." It solves the central problems (counter space and flow-to-counter association) of per-flow measurement by 'braiding' a hierarchy of counters with random graphs. Braiding results in drastic space reduction by sharing counters among flows; and using random graphs generated on-the-fly with hash functions avoids the storage of flow-to-counter association.

3.4 Summary

In this chapter, the fundamental properties of the original Internet architecture were described laying the motivation and rationale of ongoing efforts towards novel approaches to networking with focus on the access and distribution of named content. The content-oriented networking paradigm fundamentally departs from the traditional host-centric approaches to naming, routing and forwarding. Data structures and algorithms to implement core networking functions play a fundamental role in enabling the realization and adoption of new network technologies. The sub-field of probabilistic techniques and data structures has been shown to be powerful ally of networking application designers. In content-oriented networks, where flat identifiers and new naming spaces pose important challenges to scalable packet forwarding at line rates, the trade-offs introduced by probabilistic approaches appear worth to consider in order to support the packet forwarding needs of content-oriented architectures.

In the next chapter, we discuss the thesis contributions in exploring the application of probabilistic data structures to realize compact forwarding methods. The novelty of the developed techniques will be discussed along their application in architectural proposals and the resulting principles of a compact forwarding plane for content-oriented networks.

Chapter 4

Contributions and Discussion

This chapter presents a more detailed discussion of the contributions around the concept of compact forwarding in content-oriented architectures. The chapter starts by reviewing the research questions posed in the introductory Section 2.3. Then, the guiding principles that appear in the proposed compact forwarding methods are discussed, highlighting the essence of the lessons learnt and the technical merits of the author´s contributions. Throughout the discussion, the reader is referred to the annexed publications [A]-[H] for further details.

4.1 Reviewing the Contributions

The motivating trigger for this thesis is the emergence of network architectures that put content objects at the center of the architecture in detriment of traditional network node addressing. This ambitious endeavor calls for a profound rethinking of the communication paradigm. At the lowest layers, i.e., the hardware-based packet forwarding substrate, we face scalability and performance challenges when trying to move labeled pieces of content at line speed and to potentially multiple destinations simultaneously. Similar to IP multicast group addresses, which, in effect, are flat identifiers that do not easily lend themselves to topological aggregation, naive approaches to forwarding require in-network forwarding state to grow linearly with the number of endpoints, or potentially worse in the case of content-oriented architectures, with the number of advertised content objects.

During the characterization of the compact forwarding research problem (Section 2.3) we raised a series of questions that have been addressed throughout the publications [A]-[H]. In the following, we synthesize the answers to these questions:

What is a suitable forwarding substrate for content-oriented networks departing from the host-centric paradigm of IP? Which are candidate features and data structures of such forwarding

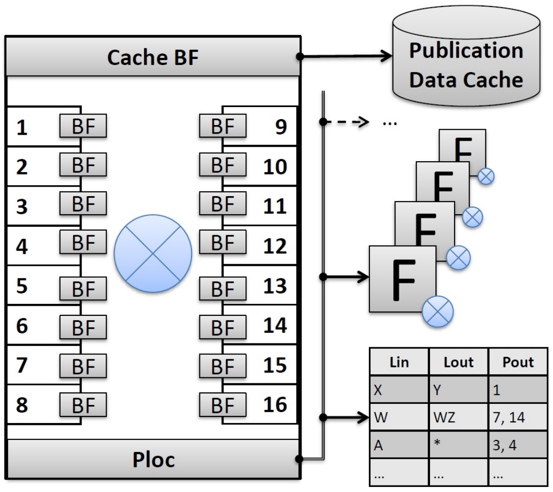
planes? Putting identifiable pieces of bits (information/content/data) at the heart of network architectures – able to scale and evolve as network demands changes – calls for a forwarding plane that is simultaneously scalable, resource-efficient, namespace-agnostic, and secure from design. To this end, we have identified and applied a series of principles and techniques: (1) the separation of control plane functions from the fast-forwarding data plane, (2) the flexible use of hash-based data structures to compactly embody inter-networking namespaces such as links, nodes, and content identifiers, (3) the multicast model of communication, and (4) secure considerations to hide the network properties and resources from potential attackers while allowing the receivers more control over the packet delivery.

When trying to take forwarding decisions on a large space of non-aggregatable identifiers, and memory space is an issue, a hash-based probabilistic data structure like the Bloom filter can be a powerful tool. Its capacity to answer set membership questions (i.e. false positive performance) does not depend at all on the nature of the elements (i.e., size, structure) but on the ratio of memory to number of inserted elements. Therefore, hash-based data structures like Bloom filters appear convenient to handle large sets of probabilistically unique identifiers that are either too long, unstructured (i.e., non-aggregatable), or both.

With the challenges of content-oriented forwarding in mind, we have studied, designed and proposed implementations for compact port-forwarding functions $i = F(I, L, H)$ based on probabilistic data structures to resolve the output port(s) $\{i\}$ of packets carrying information I, using local state memory L, and performing headers-in-headers functions H.

The **SPSwitch** [A] is a Bloom-filter-inspired port-forwarding engine well-suited for forwarding on a flat label space. As can be seen in the SPSwitch model of Figure 4.1, each possible message output is represented by a Bloom filter. For an incoming message with a label I, the compact forwarding function $F(L, I, H)$ of the switching engine basically asks each port p (in parallel) a set membership question of the sort "*is label I in outport L_p?*" Note that output destinations (i.e. ports) are not just limited to physical port-in/out interfaces but should be regarded as generic outputs, potentially including also local processes, virtual ports, recursive operations, and cache systems. A companion 'control plane' is responsible to maintain the Bloom filters per outport by inserting into the Bloom filter(s) the corresponding packet label(s). Upon data packet arrival, all possible outputs in the datapath are queried in parallel to make the forwarding decision. As Bloom filters do not return false negatives, packets are forwarded along the intended outputs, and, with some error probability, packets are duplicated on extra outports.

A naive p-bank Bloom filter approach consisting of allocating one fixed-size Bloom filter per potential outport, presents some limitations inherent of basic Bloom filter constructs: a) lack of associated values: just binary probabilistic set-membership responses; b) expensive deletion: counting



Lin	Lout	Pout
X	Y	1
W	WZ	7, 14
A	*	3, 4
...	...	...

Fig. 4.1: The SPSwitch forwarding engine. Source: [A]

Bloom filters are costly in memory sizes; c) no notion of time: missing association of filter elements or cells with timing information; d) unbalanced usage of memory per outport: unpredictable destination demands difficult the overall system design and memory allocation. In order to overcome these issues, we proposed an implementation of the SPSwitch based on leveraging a hash-based packet classification technique by Bonomi *et al.* called d-left fingerprint compressed filter (FCF) [52]. Acting as a probabilistic hash table where element fingerprints are stored along the outport value, the FCF behaves as a fast forwarding table with some probability of falsely returning additional values. Due to its hash-based nature, the forwarding decisions can be taken at constant time and may accommodate a variety of packet headers (e.g., 256-bit content IDs, flat forwarding labels). By storing probabilistic keys together with fixed size values (e.g. outport information), the FCF-based SPSwitch design (i) makes a more efficient usage of the total memory available, (ii) gracefully and fairly degrades the false positive performance among all possible outputs as more elements are inserted, (iii) simplifies element deletions and aging mechanisms, and (iv) yields better false positive performance in practice, especially when targeting very low false positive rates and large port densities.

When exploring the solution space of source routing alternatives, the simplest form consists of concatenating the forwarding nodes' network identifiers on the path between the communicating parties. Concerns with traditional source routing solutions include the overhead of extra packet headers and the security implications of disclosing network information and allowing the sending nodes to explicitly determine how packets are routed.

We have contributed to the design of a compact forwarding method in **LIPSIN** [B] based on encoding and carrying a Bloom filter in the packet header containing a list of elements (e.g., source path/tree). This way, LIPSIN addresses one of the main caveats of source routing, namely the overhead of having to carry all the routing information in the packet. Moreover, in-packet Bloom filters (iBF) do not reveal these identifiers to the sending nodes, nor the sequence or amount of hops involved.

For each point-to-point link, a Link ID (e.g. $\overrightarrow{AB}$, $\overleftarrow{AB}$) is assigned per direction without requiring a common agreement between the nodes. Link IDs take the form of a single element Bloom filter of length *m* (256) and with *k* (5) bits set to 1 and form thereby a probabilistically unique link naming space ($m!/(m-k)! \approx 10^{12}$). Assuming enough network topology information (e.g., by network-wide views or gathered during flow initiation / content request messages), a delivery tree can be constructed by inserting (bitwise ORing) the required Link IDs between source(s) and sink(s) into the forwarding iBF. On packet reception, each forwarding node checks its candidate outgoing Link IDs against the iBF-labeled incoming packet (see Figure 4.2). On match, the packet is forwarded along that link. False positives will cause packets duplicated over some extra links. In this case, the proposed compact forwarding function $i = F(L, I, H)$ basically consists of concurrently asking set membership questions of the sort "*is link identifier L_i in packet label I?*"

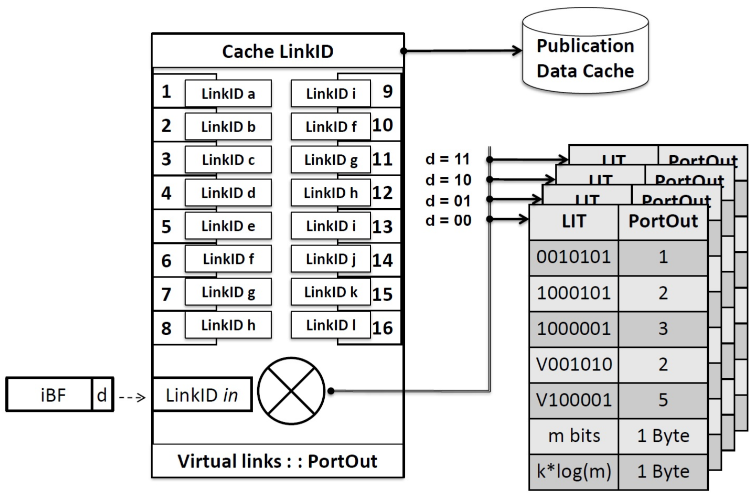


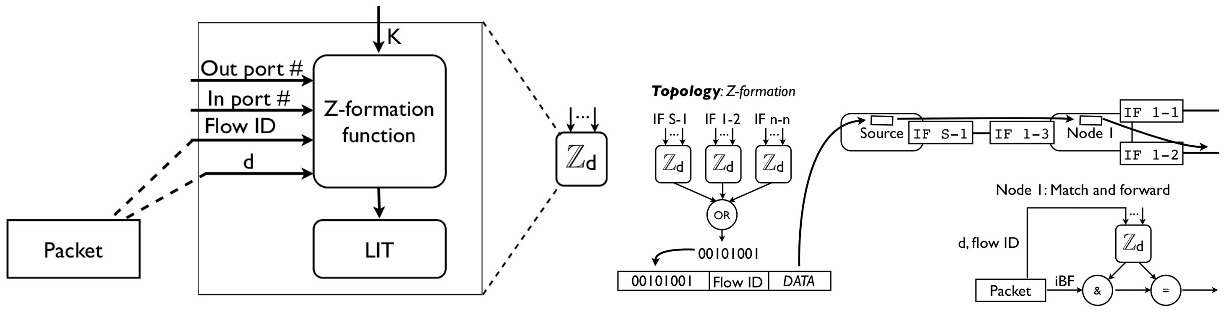
Fig. 4.2: The LIPSIN forwarding engine based on in-packet Bloom filters. Source: [B]

As a clear advantage of a source routing approach, forwarding state L becomes independent of the global address space and is limited to the set of neighbouring nodes (plus additional default or backup entries and a number of multi-hop virtual links). The reduced state required in forwarding nodes makes source routing attractive for obvious scalability and performance reasons such as hardware-friendly fast forwarding due to small table lookups and the lack of heavy header-rewriting operations.

The main caveat of vanilla iBF forwarding is that as long as the network topology is stable, any host with a valid iBF is able to send unwanted traffic to overwhelm the links over the iBF-defined path and attack the destination(s). To protect the iBF forwarding plane from malicious abuses, we have proposed the **Z-formation** compact forwarding method – the main contribution of the publication [D]. The core idea is a dynamic computation of link identifiers based on packet contents and a time-based shared secret kept in the forwarding substrate. The edge-pair labels are computed on a per-packet basis and take the local time varying context (e.g., time-based secret key $K(t)$, in/out interface numbers) as input to the computation function (see Figure 4.3(a)).

Basically, the Z-formation can be defined as header manipulation function H that dynamically determines every possible packet output L_i based on in-packet information I and additional node state $L(K)$. The function H is used on a per-node basis but does not alter the content of iBF after the forwarding decision has been made. In contrast, the header-in-header functions H proposed by the Deletable Bloom filter design [F] and the per-hop permutations introduced in [H] and experimentally validated in [51] result in a permanent change of the packet state I represented by the iBF.

The cryptographic computation of edge-pair labels enables the iBF to simultaneously act as a capability and a source routing identifier for the multicast tree. The inclusion of the previous edge makes it more difficult for an attacker to inject traffic into an existing flow and reduces the probability of loops. Without explicit authorization of the receiver and/or involvement of the topology system, data packets are not forwarded in the network due to the absence of static host addressing mechanisms. Secure iBFs act as "encrypted" source routes (ala capabilities) compactly represented in fixed-size iBFs, maintaining the link identities undisclosed and meaningful (i.e. routable) only for nodes *en-route*, bound to additional packet information (e.g., content identifier, IP 5-tuple, type-of-service field), and expirable after some time period (e.g. a factor of the secret $K(t)$ renewal time).



(a) The Z-formation probabilistic port-forwarding function.

(b) Secure compact forwarding with self-routing capabilities.

Fig. 4.3: The Z-formation compact forwarding method. Source: [D]

What are the dimensions and limits of the solution space to move content objects at scale? Can we do better than the fundamental trade-offs of distributed systems theory by introducing non-deterministic (probabilistic) techniques? Our dissertation around compact forwarding explores forwarding mechanisms based on probabilistic data structures that require less space in packet headers and in-network forwarding tables than traditional, deterministic approaches. The application of lossy probabilistic data structures to the packet forwarding problem opens a new vector in the design space, namely transport network efficiency due to erroneous packet duplications. Hence, a probabilistic approach to packet forwarding introduces a memory-correctness trade-off, where memory is represented by the amount of packet header and network state information, and correctness is traduced in extra bandwidth consumption due to unnecessary packet duplications. By introducing randomized techniques into the packet forwarding problem, lower memory usages and shorter running times can be obtained, which turn the probabilistic compact forwarding methods more scalable and resource-friendly than alternative, deterministic approaches.

Compact forwarding methods based on network-hosted Bloom filters [A] require fewer bits per entry in network forwarding tables than the size of the packet header identifiers I on which forwarding decisions are taken. The compact in-network state approach of the SPSwitch requires a fixed amount of space per entry independently from the specific namespace. While memory-efficient on a per entry basis (see [Table 2, A]) and without other means for aggregation other than synthetically (i.e. lossy compression), the main scaling challenge is the $O(n)$ memory requirement of the compact content-oriented forwarding table proposed in the SPSwitch design.

When Bloom filters are carried in packet headers, it allows for fixed packet header sizes acting as a forwarding identifiers that compactly represent multi-hop routing state information – potentially coupled with security credentials (cf. Z-formation [D]). As a consequence, forwarding tables at network nodes are kept small and require only one entry per next-hop (e.g., physical and/or logical neighbours plus multi-hop virtual links). The compact iBF forwarding method introduces however larger packet headers as they represent another header in addition to the content identifier. The probabilistic approach enables to keep the size of the packet header constant while offering different forwarding efficiency levels in function of how much network state is to be saved (e.g., network-based virtual links vs. more filled iBFs as discussed in [C]). This compact forwarding approach enables moving state (forwarding information) between packets (iBFs) and network nodes. For instance, with 256-bit iBFs (the size of two IPv6 addresses), stateless multicast can be supported by including around 35 links (cf. Figure 4.4) or enable Internet-wide unicast communications, which can be assumed to require less than 14 hops.

A topology system having internal representation of the delivery trees and knowing which links the packets need to pass, can determine when to use Link IDs and when to create state in the form of virtual Link IDs or specific forwarding identifier mappings. If larger multicast groups are required then network state can be installed by defining virtual links spanning multiple hops or by adding entries in the SPSwitch deployed at domain boundaries (cf. [Figure 6, B][C]). Additionally, a series of

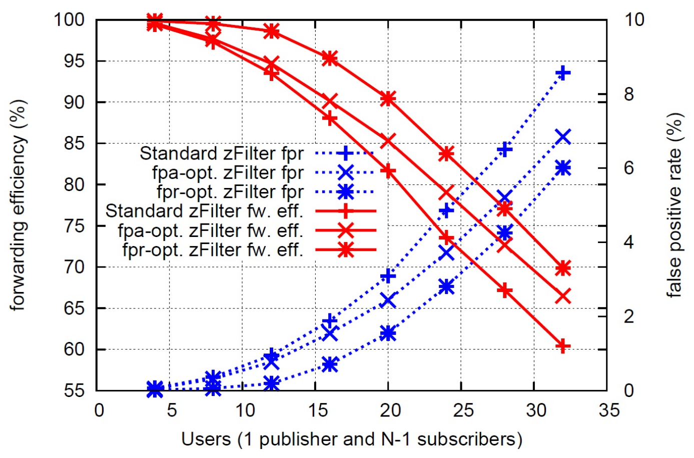


Fig. 4.4: False positive (right axis) and forwarding efficiency (left axis) simulation results for AS 6461 with d=8 Link ID Tag optimization and k=5 hash functions. Source: [B]

iBFs can be stacked (similar to MPLS) to increase the total routing information carried, for instance, when large multicast groups are required.

Traditional deterministic approaches to packet forwarding are commonly constrained by the maximum capacity of the forwarding tables (i.e. FIB size) and suffer from limitations to additional packet headers due to maximum transfer unit (MTU) specification of the transport networks. Probabilistic approaches like the SPSwitch or the iBFs allow to operate beyond these limits (e.g., adding more elements to the fixed-sized Bloom filters) at the cost of forwarding efficiency penalties. The SPSwitch design based on variable size fingerprints allows for an optimized usage of the available fast memory, and gracefully has its false positive rate adapted to the number of inserted elements. Similarly, compact forwarding based on fixed-sized iBFs does not place a hard limitation on the number (or nature) of the (source routing) elements carried in the iBF-based packet header. However, due to security concerns and a minimum forwarding performance, the number of bits set to 1 in the iBF will be commonly restricted to a certain fill factor ρ (typically around 0.5).

The Z-formation forwarding method [D] requires only minimal network state L in form of a forwarding context consisting of a numbered list of interfaces (physical plus virtual) and a time-varying secret key $K(t)$. The associated cost is additional per-packet processing, since edge-pair identifiers need to be computed on a per-packet basis to include packet information (e.g. content/flow ID) and the link that the packet came in from.

Fundamental parameters when dealing with one-sided error prone forwarding methods include defining a target threshold on bandwidth efficiency to account for both penalties due to extra packet duplications and potentially larger packet header sizes. Given a fixed amount of memory space, either in network-based forwarding tables or packet header size, different operational points can be achieved to scale up by compacting more forwarding information (i.e., forwarding directives) at the cost of reduced bandwidth efficiency.

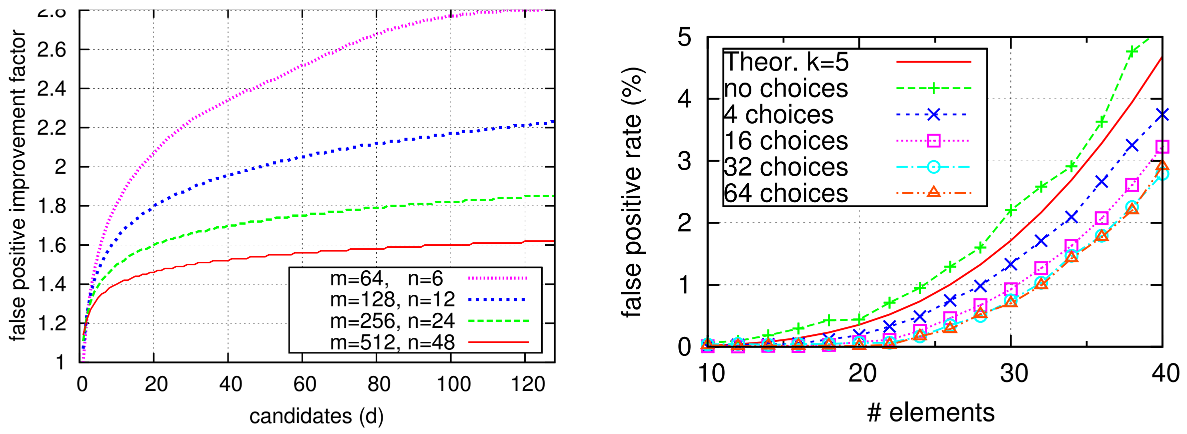
What considerations and enhancements are needed to build a correct distributed forwarding gear on top of one-sided error prone forwarding decisions? One-sided error hash-based data structures in the spirit of Bloom filters represent an appealing way of providing synthetic aggregation of location-independent (non-aggregatable) namespaces, *if* the effects of false positives can be managed. Consequently, the gains of using lossy data structures like the Bloom filter are only attainable if the compact forwarding methods can be made practical, i.e., if false positives can be contained to guarantee correctness of the forwarding functions under finite resource usage (i.e. loop-free).

The key observation that motivated our exploration of one-sided error data structures for packet forwarding is the *bounded effect of false positives* in content-oriented, interest-driven architectures based on publish/subscribe primitives or similar data-oriented APIs. Firstly, content oriented primitives like publish/subscribe inherently tolerate false positives, since non-requested content items have a limited live in the network and do not create forwarding states. Moreover, end-nodes are expected to effectively process only those pieces of information for which they have explicitly expressed their interest. Secondly, with support for opportunistic caching in the network, falsely forwarded packets result in copies of data being replicated and potentially used to fulfil future requests of nearby receivers. Thirdly, packets forwarded due to false positives are not propagated over many hops due to the large label space (uniqueness) and the exponentially decreasing probability of *chained false positive* forwarding decisions.

The price of relying on probabilistic data structures that give up correctness and transport efficiency in favor of less state and manageability is requiring some extra considerations to have enhanced control over the effects of false positives. Here is where this thesis makes a series of technical contributions in the form of extensions to probabilistic data structures. While initially designed to deal with false positives of the compact forwarding methods, the algorithmic techniques are general enough to be applied to other applications relying on probabilistic data structures.

In the design space of lossy compressed forwarding tables, we found that the d-left fingerprint compressed filter (FCF), originally designed to approximately track the state of network traffic flows in dynamic environments [52], may be a better approach than maintaining a Bloom filter per outport for the performance and functional reasons previously discussed and detailed in [A]. Our in-depth study of the iBF design space [G] includes the development and practical validation of three useful extensions (1) to increase the practicality and performance of iBFs by exploiting the power of choices at hashing time to choose the best performing iBF among d candidates, (2) to enable false-negative-free element deletions by encoding collision-free iBF regions, and (3) to provide a secure method for iBF constructions by coupling packet-specific information and a time-based hashing mechanism to the iBF set/check operations. As a general data structure, iBFs can be useful for networking designs that tolerate false positives and decide to move state to the packets themselves.

The application of the *power of choices* technique enables choosing the most convenient set of hash functions (i.e. iBF bit pattern combination) or best network path has been shown to be a very powerful and handy technique to deal with the probabilistic nature of hash-based data structures; providing finer control over false positives and enabling compliance to system policies and design optimization goals. The performance benefits of having d candidate Bloom filter representations is shown in Figure 4.5 and been verified in practice with the forwarding efficiency gains shown in Figure 4.4.

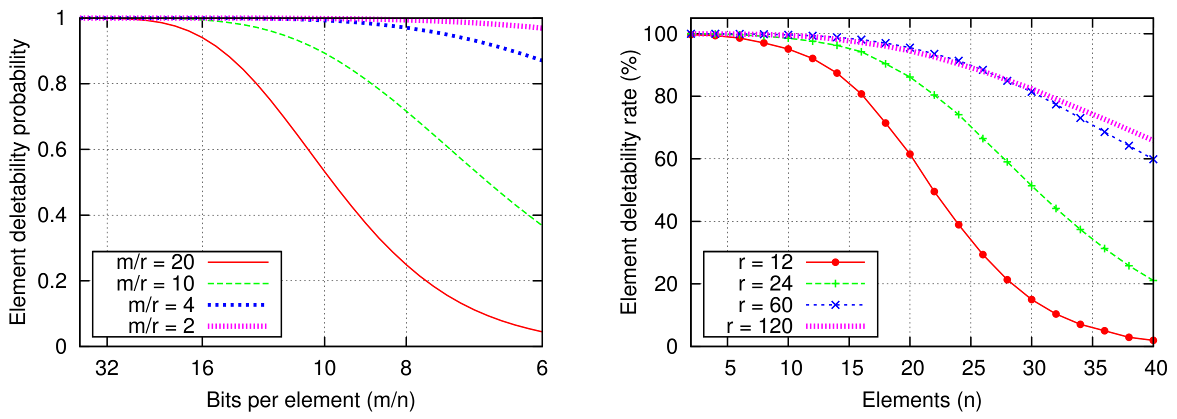


(a) Theoretical estimate of the false positive improvement when having d candidate Bloom filters. (b) Best observed false positive rates in m=256 bit Bloom filters usgin k=5 hash functions.

Fig. 4.5: False positive performance gains of the power of choices technique. Source: [G]

The contribution of the **Deletable Bloom filter (DlBF)** [F] is a space-efficient element deletion technique that allows (probabilistic) element deletions without the overhead and limitations of previous work on Counting Bloom filters or false-negatives-prone solutions. This way, forwarding entries compacted in Bloom filters can be dynamically updated at a fraction of the cost in terms of memory. In case of iBF forwarding approaches, the DlBF enables removing already processed forwarding directives (i.e., link identifiers) as the iBF traverses the network. Other alternatives to the DlBF that support element deletions would be too space consuming when placed in small packet headers. Figure 4.6 shows the capability of the DlBF to remove elements by using only a small amount of bits (r) to encode the bit regions where collisions happened.

A straightforward approach to handle element deletions in standalone network-based Bloom filters (as in the SPSwitch model [A]) consists of keeping a copy of the data structure (i.e. shadow forwarding table) in the control plane memory containing the full (or partially extended) information



(a) Deletability estimate as function of the filter density m/n for different collision bitmap sizes r.

(b) Experimental deletability rate (mean values) of a 240-bit DlBF for different number of regions r.

Fig. 4.6: Compact deletability capabilities of the Deletable Bloom filter. Source: [F]

(e.g., uncompressed forwarding table, counting Bloom filters), and, upon updates, reconstruct the Bloom filter that should be used on the fast forwarding plane (i.e. working forwarding table). Such a per-port Bloom filter re-computation has been proposed elsewhere [187], and although feasible, a periodical replacement of the working forwarding table may be more costly than direct element removal/updates on the fingerprint compressed forwarding table proposed for the SPSwitch, where single entries can be tracked individually and only elements subject to false positives would require additional conflict resolution using additional control plane (slow path) information.

Security issues need to be considered in a networking environment like the Internet where attackers may have incentives to infer the network topology, send unauthorized traffic, or cause denial of services with broadcast storms or amplified targeted attacks to specific nodes. For the sake of yielding a correct and practical iBF forwarding machinery, we have contributed to the understanding of forwarding anomalies [51] and security implications [H] of a multicast-capable forwarding plane based on iBFs. The properties themselves that make multicast attractive (e.g., efficient 1:n and m:n data transfer) are at the same time the root of the challenges of providing a secure networking environment. Similarly, the probabilistic nature of the hash-based Bloom filter data structure is simultaneously the base for its appealing features (e.g, compact state, simplicity) and the door for a number of attack vectors.

In order to guarantee *forwarding service availability* of Bloom filter based data planes under malicious attacks [H], additional data plane mechanisms are required in order to ensure that only packets from authorized users are forwarded, providing thus resistance to (potentially distributed) DoS attacks. Solutions to the three forwarding anomalies (packet storms, forwarding loops, and flow duplication) include adapting the Bloom filter parameters and performing hop-specific bit permutations [51], in addition to the cryptographically generated dynamic link identifiers [D]. We have experimentally validated the effectiveness of the solution suggesting that a probabilistic approach to packet forwarding based on iBFs can be made practical also from a security perspective – provided a series of mechanisms are included from design.

More specifically, the Z-formation method [D] enables including additional parts of the content of a packet as an input to the keyed edge-pair label computation. As an example, it becomes possible to tie the edge-pair labels to the IP 5-tuple and the type-of-service field in the packet. This means that the iBF is only valid on its intended path, only for a specific time and quality of service, and only with the given 5-tuple. The proposed coupling of iBFs to time and packet contents provides an effective method to secure iBFs against tampering and replay attacks.

To our knowledge, Z-formation based iBFs is the first approach that combines forwarding identifiers and capabilities in an efficient way, thereby creating the notion of *self-routing capabilities*. By virtue of the Bloom-filter-based construction of the capabilities, even the link names remain statistically undisclosed. By binding the routing capabilities to specific flows, time periods, and network paths, we create a high barrier for attackers, making it hard to forge valid capabilities. Moreover, the iBF approach places the state requirement at the source, instead of the routers, which greatly alleviates the potential for DoS attacks against the network infrastructure.

Along this journey, we have learned about the gap between theory and practice via extensive simulation work and practical networking environments like a publish/subscribe networks [B,C], inter-domain multicast [H][51], and data center networks [E][239]. We were able to confirm the observation that hashing techniques in practice do differ from theoretical expectations [G]. For instance, for the same memory to element ratio, small sized Bloom filters exhibit higher false positive rates than larger sized Bloom filters. Compact forwarding based on iBFs calls for algorithmic techniques that help in rectifying the divergences from theoretical models to practical applications, in addition to guarantee forwarding correctness and address the security concerns of a public networking infrastructure.

Correct forwarding despite false positives

Basically, a false positive at packet forwarding time results in multiple values being returned from the port-forwarding function $F(I, L, H)$. The strategies to deal with these events are multi-fold.

In the case of *unicast communications*, one approach is to choose one candidate outgoing link and rely on the completeness of a (shortest-path) routing scheme. The forwarding decision is encoded in the packet information I so that if the packet returns to the same forwarding node (e.g., because lacking of a matching forwarding entry at the next hop), the same forwarding decision is not repeated. This approach has been proposed by the authors of the BUFFALO architecture [187], showing its practicality in enterprise networks with shortest path routing. Packets are guaranteed – with high probability – to reach their destinations with bounded stretch increase following a "trial-and-error" approach to handle false positive unicast forwarding.

The approach we have proposed in unicast networks with centralized network control and known network topology (i.e. data center forwarding services [E]) consists of leveraging path-multiplicity and network-wide information to have multiple candidate iBF representations (i.e. bit patterns). This strategy allows to choose either the set of hash functions that exhibits no false positive for a given path (as proposed with the Link ID Tags in [B]), or to exploit the multiple paths to select an alternative path between the communicating parties (as in the multi-path oblivious routing service [E]). In a way, this approach to circumvent false positives re-inserts determinism in the one-sided error-prone forwarding substrate.

In the case of *multicast communications*, a false positive event at packet forwarding time is indistinguishable from an explicitly programmed multicast directive. In this scenario, reducing the rate of false positives is paramount to keep the forwarding efficiency levels acceptable and avoid critical false positive events (e.g., forwarding loops). The d-candidate Link ID Tag extension [B] allows to construct iBFs that can be optimized, e.g., in terms of the false positive rate, compliance with network policies, or multi-path selection. The basic idea consists of constructing a series of candidate iBFs each using a different combination of hash functions and to select the best-performing candidate according to any appropriate metric or network policy (e.g., expected false positive rate, avoid false positives over inter-domain, congested, and/or low capacity network links).

In either approach, in the event of a packet reaching a network node without a matching forwarding entry due to a false-positive-driven forwarding decision, the packet might be returned to the previous hop (e.g. as in [187]) over the incoming interface with a mark (i.e. changing the in-packet state) to note this dead end. A new forwarding decision can now leverage the updated in-packet state along additional context information like the new incoming interface. In such situations, one optimization approach may consist of a temporal cached decision to block further packets within the same flow being forwarded along that interface. Another strategy we have validated in [51] consists of performing per-hop permutations of the iBF bit vector. This way, packet forwarding becomes history-dependent, i.e., a falsely forwarded (or duplicated) packet returning to a node in the path defined by the iBF would have the bit patterns changed in a way that infinite loops can be avoided with very high probability. The use of a deletable extension as proposed in the DlBF [F] is another example of changing the packet state I by removing already processed elements in the iBF. This way, loops can be avoided with very high probability requiring only that one bit of the looping links being in a deletable region.

The above packet state manipulation techniques correspond to headers-in-headers functions H

that result in a permanent change of the packet information I and consequently alter the result of the compact forwarding functions $F(L, I, H)$.

Alternative solutions may consider passing falsely forwarded packet(s) over a 'slow-path' to a control entity (running either at the control plane of the forwarding node or at a remote centralized location) to resolve the issue (e.g. Rack Managers [E]). This solution is acceptable as long as the rate of false positive events is low enough so that (i) it does not burden the control channel, and (ii) the additional delays are tolerable.

One common theme in our probabilistic approach of compact forwarding has been tackling the randomly looking flat identifiers with randomized algorithms: (1) fighting flat identifiers with hash-based probabilistic data structures that act as (lossy) data aggregators (e.g. SPSwitch [A]), and (2) fighting the randomness of hash function outputs with a multiplicity of choices in (i) the set of hash functions used to succinctly represent the same information collection (e.g. Link ID Tags [B]), or (ii) candidate collection of forwarding identifiers that result in packets reaching the same target(s) over alternative paths (e.g. false-positive-free iBF routing table [E]). Moreover, a randomized multipath forwarding technique like Valiang Load Balancing (VLB) has been shown to be very useful to cope with the randomly-looking and time-varying traffic matrices observed in cloud data center networks [E].

4.2 Principles and Applications

Those are my principles, and if you dont like them... well, I have others. — Groucho Marx

We now state several design and technical principles that we found fundamental for the conception of the compact forwarding enablers towards scalable content-oriented network architectures. Design principles are deduced from design goals — also known as requirements — that cover both functional and performance objectives. Table 4.1 summarizes how those principles have been repeatedly utilized in the proposed forwarding methods and applied in the network architectures.

Principle 1: Separate routing from forwarding

Content-oriented routing protocols should be separated from the forwarding elements. Disaggregation of the routing functions (e.g., path computation) from the individual routing elements seems to be the best way to cope with the scale of the envisioned namespace for information objects. Separating the control plane from the data plane is not a new idea but rather a traditional, even controversial, networking perspective well-known from the context of the public switched telephone network (PSTN), Tab. 4.1: Summary of principles and applications of the contributions to compact forwarding.

Principle	Applications and algorithmic techniques
P1: Separate routing from forwarding	Slow-path to match communication interests and fast-path for content delivery [A-C]
	Handle routing intelligence at resource-rich control entities uncoupled from the forwarding substrate [E]
P1a: Generality	Hash functions apply to any namespace (form, size) [A,B,G]
P1b: Simplicity	Simple $O(1)$ insert and query operations [A,B,G]
P2: Allow a flexible operation point	Tunable trade-off memory (network, packet) for efficiency [A-C]
	Move forwarding state to packet headers [B]
	Install virtual links to offload packet headers [B,C,H]
P3: Multicast-friendliness	Parallelizable next-hop query on all possible outputs [A,B]
	Compact encoding of information flows / multicast trees [B,C]
P4: Receiver-controlled data plane security	Hide network identities (links and nodes) by generating randomly-looking forwarding identifiers [B,D]
	Default-off unless valid forwarding identifiers are granted [B-D] and renewed by receivers [H]
	Forwarding identifiers are path-dependent and expirable [D]

Synchronous Optical Networking (SONET) or Synchronous Digital Hierarchy (SDH). Modern router architectures are already built upon a clean split of the hardware-based forwarding elements (i.e., line cards) from the software-based control plane modules. Ongoing trends [88, 89] suggest taking this separation further and running the control plane in remote boxes which communicate with the forwarding elements via a well-defined protocol (e.g. OpenFlow [89]).

As observed by Feamster *et al.* [84], the growth of the Internet has introduced considerable complexity into the global routing infrastructure, with features being added to BGP to support more flexibility at a larger scale. Arguably, this complexity has made routing protocol behaviour hardly understandable, increasingly unpredictable, and error prone [85]. In an IP world, separating routing from forwarding means IP "routers" becoming "lookup-and-forward" switches to move packets as rapidly as possible without being concerned about path selection. The forwarding path of carrier-grade networks must be highly optimized and is commonly assisted by hardware – relatively costly and difficult to change over time.

One approach for scalable information-centric networks is relying on a 'slow' plane to route requests for content to (rendezvous) repositories where information from the sources can be convened, keeping the fast data plane "limited" to fast hardware-based forwarding operations plus additional supporting functions (e.g., caching) [148]. The separation of routing from forwarding allows for a separation of concerns that enables complementary approaches to perform the act of locating the content from its delivery.

The rendezvous functions to match the interests of the communicating parties can evolve in a way that multiple parties can join and provide this service similar to over-the-top content providers, search engines, or more open, in the spirit of P2P/BitTorrent-like communities and DHT overlays. With regard to the latter, forwarding methods in traditional DHTs are not considered compact as they typically maintain entries in the forwarding table of the size of the identifier to be routed. However, if we allow multicast-type of forwarding in the DHT resolution process compact forwarding tables based on probabilistic data structures (cf. SPSwitch FCF [A]) could be considered.

Like any clean functional separation in networking, this approach allows to deal with the unexpected and unknown complexity of the resulting content-oriented routing (or resolution) system, which is better suited to evolve independently from the fast forwarding stratum and allows innovation at both packet level forwarding and control plane functionality. The following two sub-principles offer further guidance for the design of the compact forwarding methods.

Sub-Principle A: Generality The forwarding methods should be generic enough with regard to the identifiers on which the forwarding decisions are taken. This enables setting flexible network indirection points in the network potentially based on different namespaces (e.g., with potentially different granularities) while still re-using the same basic hardware-assisted switching operations. Being generic in its operations, hash-based forwarding mechanisms can be applied in different networking scenarios, not only with regard to the semantics of the naming space but also to the layer at which the indirection decisions are taken (e.g., traditional L2, L3 or L4-L7).

Hence, the assumptions of the upper layer control plane and namespaces can be kept to a minimum. The main abstraction is a (flat) label which is essentially a bit string representing any higher level information (e.g. information object, network link, virtual machine, multicast tree). In this sense, the compact forwarding methods try to address the requirements of the most challenging namespace, namely the name-independent schemes that enable routing on topology-independent or arbitrary identifiers.

Hash functions are well-suited for these naming purposes as their operations are agnostic to the nature and actual identity of the input elements. Bloom-filter-like data structures are hence a convenient place holder for large amounts of non-aggregatable identifiers. The hash-based link identifiers and the resulting routing iBF-based "channel" abstraction provides a great deal of flexibility concerning the underlying network that carries data between two entities: it could be an actual point-to-point link, or an IP network, on ATM network, or a SONET, for example.

The result is a more polymorphic nature of forwarding, at least imposing no restrictions to a variety of routing (i.e. network control) paradigms, and ideally offering parallel support to a set of them with the same underlying hardware.

Sub-Principle B: Simplicity The forwarding methods should be based on simple per-packet primitives. That is, the forwarding decision functions should be resource-friendly and work at line speeds. Clearly, the underlying mechanisms must not only be feasible but should also give a fixed, hardware-amenable target to implement, avoiding the uncontrolled growth of options and features that force an endless re-engineering of routers' chipsets (cf. [113]).

Simplicity should be favored for the design of time-constraint forwarding functions returning the outport(s) of incoming packets based on local context and the in-packet information. While not enough to fully define a forwarding engine that in practice is formed by a number of pipelined functional blocks (e.g., QoS scheduling, caching, counter updates, logging, etc.), the compact port-forwarding functions developed can be easily plugged at different stages of the packet processing pipeline of networking hardware.

As a synergistic relation, simplicity positively affects complexity i.e., the time or pace essential to complete the forwarding operations. Indeed, in many ways, we try to follow C.A.R. Hoare's statement about the relationship of simplicity and reliability: the price of reliability is the pursuit of the utmost simplicity.

Principle 2: Allow a flexible operation point

The forwarding machinery should enable the network architect/operator to find a sweet spot for a given networking scenario, for instance, being able to trade network resources usage (e.g. bandwidth efficiency due to larger packet headers and unnecessary packet duplications) for network state reduction (e.g. smaller forwarding tables). This principle allows the same forwarding methods to be optimized for the available (technology-dependent) resource pool (e.g., bandwidth, high-speed memory, link stability). After deployment, some resources may be hard to upgrade (e.g. on-chip memories), and flexibility in choosing the operation point (even at runtime) enables giving up on some dimension to postpone costly hardware upgrades or to adapt to temporal usage spikes.

While decoupling the control plane and upper layer identifier space from details of the forwarding engine facilitates their independent evolution, flexible forwarding functions allow the system to adapt (at run-time) to changes in scale due to unexpected network usages.

Bloom filters naturally allow for a flexible operation point as they can accommodate larger quantities of forwarding information (than initially planned) for a fixed memory size at the cost of transport network efficiency. The probabilistic approach to packet forwarding based on Bloom-filter-like data structures allows to be able of have a non-fixed (virtual) capacity in terms of quantity of (approximate) routing information.

The same principle of flexibility appears in the proposed approach to move network state to packet headers (i.e. iBFs) and backwards in form of network-installed virtual links. The core idea is to add some information to the packet header to assist routers along the packet path to perform the forwarding decisions using less resources (e.g., per packet computation plus memory accesses). Notable examples of such methods include ATM, IP/Tag switching , (G)MPLS, and so on. While offloading routing state to packet headers to make packet processing easier is an old idea [100], the proposed compact forwarding mechanisms based on this principle have taken a probabilistic approach that introduce the dimension of forwarding efficiency in the routing and forwarding space [C].

Principle 3: Multicast-friendliness

The forwarding methods should provide native multicast capabilities. In contrast to the IP host-centric background, content-oriented networking provides a new model for communication focused on data and not nodes. This way, the underlying networking substrate can be abstracted possibly to an extent where it can be no longer based on persistent, IP-address-like names (cf. with [143]).

As is well known, the traditional approach to IP multicast [164] has both scalability and deployability problems [78], since it involves adding state to all of the intermediate routers. The IP network is originally an unicast network where multicasting is offered as an additional service in this network. In contrast, content-oriented networking regard multicast as its native routing and forwarding approach, with unicast being relegated to "just" an especial case. Under such networking model, hosts names lose their pivotal role in favour of giving a name to the data that a communicating entity is interested in. Which specific node converts the original information into the form that is finally delivered becomes less relevant, as long as the data is timely and correct. In addition to handling packet multicasting as a natural primitive, the forwarding plane should offer scalable alternatives.

Expressing the packet forwarding problem as a set membership answered by either in-packet of in-network Bloom-filters makes support for multicast a natural operation at packet forwarding time. The compact representation of a multicast tree into in-packet Bloom filters is a promising approach to make multicast scalable to large number of multicast groups. By separating the multicast group management and multicast forwarding we can avoid the need for distributed multicast route computation and per group state in multicast routers. Instead of managing the group state in the routing system, the state can be maintained either at the source or in a separate group management component operated by the network owner or even outsourced to resource-rich, distributed data center facilities.

Principle 4: Receiver-controlled data plane security

The forwarding methods should be designed with security in mind and not as an afterthought. In open/commercial networking environments, network architecture designers need to assume that if some resource can be exploited, it will be. This requires assuming from the beginning that the architecture will be subject of malicious usages. While security is an overarching solution spanning the complete networking and application stack, the forwarding plane should provide built-in security features that contribute to a secure architecture.

The traditional host-oriented way of inter-networking is based on routing packets using the destination host IP address. The network was designed to serve the packet sender by delivering packets to the receiver the best it can. However, this scheme does not consider the possibility that the receiver may not be willing to receive that particular piece of data and assumes that the sender will always honour the recipient's consent. This is a root cause for a number of well-known problems, such as unwanted traffic. Existing network architectures protect against unwanted traffic mainly based on add-ons on top of the architecture like flow or packet classification combined with proactive or reactive filtering.

The forwarding plane should provide a building block addressing questions of (1) what packets can be forwarded, and (2) ensuring that approved communications occur as described. If as the result of a number of parties agreeing to communicate a particular network path(s) is approved as such, then the forwarding plane should provide mechanisms to enforce that packets in fact are delivered through the authorized path(s). Moreover, the forwarding plane should contribute to blocking the initiation (or continuation) of unapproved communications effectively blocking the communications early on close to the source, before they can escalate and consume network resources or deny the service to specific targets (e.g., DDoS attacks).

Our security considerations to protect the forwarding plane include (i) moving the control of which packets to receive and for how long, (ii) having the fast data plane off per-default, and (iii) hiding network topology and location information. At the cost of extra per-packet computations, the secure iBF forwarding can be used to provide a network-assisted DDoS protection scheme that empowers the receivers. In the resulting approach, the hosts have no names; only links are named. To be able to send, any prospective sender needs to acquire a forwarding identifier that simultaneously acts as a path capability. By delegating capability creation to a rendezvous component that explicitly considers both the senders' and receivers' interest, the power shifts from senders to receivers, which are in control of what they want to receive.

It is all about trade-offs

As is so often in engineering disciplines, network design is all about trade-offs. In the seminal example discussed in the original paper by Saltzer, Reed and Clark [91], the authors claim that "sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement." That is, the End-to-End argument was pioneering in recognizing the existence of cost-performance trade-offs that justify the incorporation of economic considerations in networked system designs.

A more recent illustration of such trade-offs in the Internet architecture has been eloquently proposed by Doyle *et al.* [16] by using the paradox "robust yet fragile" when referring to the nature of the Internet. According to the authors, the Internet is an example of "organized complexity" and can be described as a H.O.T. network, or Highly Optimized/Organized Tolerance/Trade-offs. This notion represents the trade-offs made by networks engineers when connecting computer routers, where both economic and technological trade-offs play an important role. Other noteworthy trade-offs in the Internet include the memory-stretch trade-offs of compact routing [113], the dilemma between flexibility-security [93], the control-openness tension between the Internet and the traditional telecom networks evolution to IMS/NGN architectures [240], protocol optimality versus simplicity [241], and many more.

By considering a probabilistic approach to packet forwarding, this thesis explores the trade-off between reducing forwarding state and consuming extra bandwidth. We are certainly not the first to apply this principle in networking. For instance, the PoMo architecture [168] also proposes a routing and forwarding solution that trades overdeliveries for reduced state and reduced dependence of node network locators. A probabilistic data structure like the Bloom filter challenges traditional space-time trade-offs of deterministic algorithms in exchange for correctness. Indeed, the design of Bloom filters and its derivatives is fundamentally about trade-offs striking the right balance between memory, computation and (false positive) performance. When applied to packet forwarding, this probabilistic approach allows for new solutions in the four orthogonal metrics of packet routing, namely, in-network state, in-packet state, efficiency, and adaptation costs.

Previous work [100] that noted the fact of bandwidth being cheap compared to expensive processing and memory accesses proposes forwarding techniques to trade larger packet headers for reduced per-packet processing requirements. Along the network state versus packet processing trade-offs, the secure Z-formation forwarding method brings the in-network state requirements down to a minimum at the cost of additional packet processing overhead to dynamically generate link identifiers.

The novel design of the DlBF [F] contributes with a yet unexplored trade-off in probabilistic data structures to enable false-negative-free element deletions. Depending on the fraction of bits devoted to encode the regions where collisions are detectable, different degrees of element and bit deletability can be obtained. Similarly, increasing the number of candidates consumes valuable bitspace to include the index d in the iBF-based packet header, but yields greater opportunities of generating a better performing bit pattern candidate. Yet another trade-off appears in our exploration of scalable multi-path forwarding services in data center networks, where the compact forwarding methods allow for a stateless switching approach at the cost of some reduced path multiplicity [E].

One historical example of trade-offs in networking is packet-switching versus circuit-switching technologies. Packet switching technologies were developed in a time when capacity was costly and introduced the paramount idea of multiplexing demand into fine granular packets in contrast to breaking up the end-to-end capacity into coarse circuits. Instead of provisioning costly capacity for peak demands (as in circuit-switching), packet-switching only requires to guarantee that average capacity satisfies average demands, leading to statistical multiplexing gains.

While there is no debate around the benefits of this approach, as network economies evolved and transport capacity became cheap, at the core of ISP networks, circuit-oriented designs are commonly demanded, as evidenced by the adoption of connection-oriented services (e.g., T-MPLS, PBT), DWDM and optical technologies in general. Fast re-provisioning control planes like G-MPLS bring down the over-provisioning costs of traditional circuit-switching technologies like TDM or ATM. The tension between the two technologies (circuit vs. packet-oriented) is alive and connection-oriented technologies will continue to play a significant role below the global inter-networking layer, let it be IP or a content-oriented alternative. In a way, source routing based on iBFs is in line with the needs and evolution of carrier networks to provide flexible connection-oriented services in the spirit of MPLS, the so-sought approaches for flow-based networking [242, 243], and the separation of the control planes from the forwarding elements [88, 89, 244].

4.3 Summary

This chapter provides a more detailed discussion of the thesis contributions, which can be divided into three sets: (i) principles, (ii) algorithmic techniques, and (iii) applications. The departure from previous approaches and the major achievements were discussed, including the involved trade-offs. The probabilistic approach to packet forwarding taken in this thesis represents one exciting front that allows for new trade-offs with appealing properties in the field of what this thesis refers to as *compact forwarding*, an area open for additional methods, alternative approaches, and new networking scenarios.

Chapter 5

Concluding Remarks and Future Work

A paradigm shift is brewing in networking moving the focus to accessing and distributing named pieces of content rather than host-centric conversational communications. While the overall blueprint of a pure content-oriented network architecture is a complex endeavour at its early stage, initial efforts have already started, and more importantly from a technical point of view, the first architectural approaches are being proposed and evaluated.

A common challenge in content-oriented networks is the need to take scalable forwarding decisions at high speed to move packets labeled with a potentially infinite universe of flat identifiers. Towards the technical viability of a forwarding plane capable of satisfying the needs of a content-oriented paradigm, we have departed from the host-centric forwarding approach of the hierarchical IP world dominated by deterministic algorithms (e.g., longest IP prefix matching), in favor of probabilistic packet forwarding methods that guarantee the packet delivery to the intended destination(s).

From the broader architectural considerations of content-oriented networking, this thesis only grasps the tip of the iceberg by seeking to challenge the commonly held view that networking needs to be sender-driven, centered on endpoint (network interface) identifiers, and implemented based on deterministic techniques (e.g., exact match lookup, binary tree search, TCAM). We have explored this avenue by introducing the concept of *compact forwarding* based on relaxing the outport match condition on packet fields to tolerate extra values – in addition to the correct one(s) – as acceptable outcomes of the compact forwarding methods. We have applied this idea in both network-based and packet-based (approximate) forwarding information approaches following on the unifying (Bloom) principle of reducing state requirements and simplifying multicast support by tolerating some bandwidth penalties due to false positives and potentially larger packet headers. By exchanging correctness (traduced in forwarding efficiency penalties) for space/memory time requirements (traduced in lossy aggregated forwarding information), the compact forwarding methods allow for new trade-offs in the traditional routing and forwarding design space.

The broader issue of the specific scalable content-oriented routing approach (i.e. how to distribute the state to the nodes and the in-packet information) is still to be solved by the specific content network. Its overall feasibility will depend on the introduced namespace(s) and the means to scope and work with higher-level aggregates (i.e., information domains, scopes, etc.) to address the challenges of information-centric inter-networking (cf. [20]).

At the core of our probabilistic approach is expressing the packet forwarding problem as two extreme set membership problems solved by virtue of variations of the popular hash-based data structure Bloom filter that succinctly represent the forwarding state information in both network nodes and packet headers. The cost of simplifying the port-forwarding operations and lossy compacting the forwarding information base is additional considerations to manage the effects of false positives. Hence, one central question this thesis contributes to, is whether a one-sided error prone forwarding substrate can serve as a suitable data plane holding approximate state upon which content-oriented networks can be deployed. We found that, provided the right choice of parameters and a handy set of algorithmic techniques, a probabilistic approach to packet forwarding yields an useful design space beyond traditional deterministic techniques.

Considering the definition proposed for the problem of compact forwarding, other strategies and companion data structures – probabilistic or not – may be conceived to match the features and objectives of the forwarding methods that we have worked out. Potential solutions may be biased towards specific trade-offs in the design space, propose alternative packet classification and fast forwarding techniques, or develop enhancements to the encoding of the approximate forwarding state. Specific open questions related to the latter include research on more compact or variable size encoding of the deletable regions [G] (e.g. based on erasure- or error-correcting codes?).

At the same time, we believe that the field of applications of the compact forwarding methods developed is open to other inter-networking scenarios such as overlay solutions (e.g. application-layer routing) and resource-constraint networks (e.g. wireless sensor networks). Deployability of new network architectures is certainly an issue and replacing running infrastructure is a costly proposition. An initial overlay solution (cf. [245]) and advances in network virtualization and software-defined networks are the most promising avenues to see new forwarding primitives in real networks. On this direction, the generality of the hash-based forwarding methods allows for an easy adaptation to other distributed system incarnations. We hope that our work on generalized probabilistic data structure extensions [F] becomes subject not only of new applications but also motivate enhancements or alternative solutions. We strongly believe that further optimized approaches are possible.

One specific application that would be interesting to explore is the applicability of the SPSwitch forwarding engine to compactly represent the Pending Interest Table and/or the Forwarding Information Base of the CCN network node model [184]. A probabilistic approach (in the spirit of Bloom filters) could be well suited to perform fast and memory-efficient name lookups, especially as the potential problem of forwarding loops may be circumvented through the cache-friendly flow-balanced design of CCN (i.e. Data packets consume Interest entries). Open challenges include how to deal with the unbounded, hierarchical name space and the required techniques for fast updates of the forwarding and caching engine memory [246].

Along the iBF solutions, we expect the maturity of techniques for fast host mobility [247], multicast VPN services [65], edge-controlled inter-domain multicast [H], data center forwarding services [E], and the marriage with optical technologies. Moreover, handling failure scenarios (e.g. fast iBF re-route [248]) is an area that requires further studies that may largely benefit from the long-year experience in IP/MPLS deployments, and more recently, carrier Ethernet solutions. Topology- and fault-carrying iBFs together with preventive backup forwarding entries in network nodes are certainly within the scope of future work.

Data centers are indeed a networking scenario calling for innovation in the routing and forwarding space due to the challenges of dynamic virtual machine connectivity, multi-tenancy, and ware-house scale facilities. At the same time, the tightly managed environment of data center networks allow for innovative solutions being deployable without Internet-wide agreements to address the needs of the infrastructure provider and the cloud-scale applications. In addition, turning prototypes into reality is expected to become less of a challenge due to the rise of software-defined networks enabled by open interfaces to directly program the forwarding behavior of commercial networking gear, as targeted by the OpenFlow community.

The potential impacts of general purpose computer programmability with the line-rate performance of commercial hardware are far-reaching and well aligned with the principle of separating routing from forwarding. For instance, networking inside the data center can be turned into a software problem, tractable by the same developers of the cloud content providers and bypassing thereby the traditional development cycle to have new features and functionalities brought to production only by equipment vendors. We have argued about this trend [239] and the first industry efforts towards operational software-defined networks are already undertaken [249]. Yet another example is our ongoing work [250] to combine line-rate forwarding with (remote) open-source routing stacks towards high-performance, cheap, and novel network designs enabling virtual network services and scale-out router architectures.

From a broader perspective, there is an amazing list of research questions around content-oriented networking proposals, spanning from potential additional features of the forwarding plane (e.g. network coding) to broader architectural considerations such as the fundamental dichotomy between flat and hierarchical naming schemes. Remarkable efforts towards enabling content-oriented networks are being undertaken in the continuation of pioneering work at EU FP7 PSIRP [185] and PARC

CCN [184] into the newly started follow-up projects Name Data Networking (NDN) [246] and Publish Subscribe Internet Technology (PURSUIT) [251]. Paramount questions arising from the ability to expose named content being exchanged include the development of scalable name-based routing plane(s), the role of caching and its implications on congestion and error control, content-oriented transport functions and forwarding strategies, validation of the security foundations, delay tolerant operations, and the further specification of suitable fast forwarding engines as we have pursued with the SPSwitch [A], LIPSIN [B], and SiBF [E].

Bibliography

[1] Vinton G. Cerf and Robert E. Khan. A protocol for packet network intercommunication. *IEEE TRANSACTIONS ON COMMUNICATIONS*, 22:637–648, 1974.

[2] Jonathan B. Postel, Carl A. Sunshine, and Danny Cohen. The arpa internet protocol. *Computer Networks*, 5:261–271, 1981.

[3] D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA, 1988. ACM.

[4] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[5] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), September 2007.

[6] J.C. Mogul and J. Postel. Internet Standard Subnetting Procedure. RFC 950 (Standard), August 1985.

[7] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592.

[8] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519 (Proposed Standard), September 1993. Obsoleted by RFC 4632.

[9] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771 (Draft Standard), March 1995. Obsoleted by RFC 4271.

[10] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001.

[11] J.D. Touch. Those pesky nats [network address translators]. *Internet Computing, IEEE*, 2002.

[12] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.

[13] C. Perkins. IP Mobility Support for IPv4. RFC 3344 (Proposed Standard), August 2002. Updated by RFC 4721.

[14] Mark Handley. Why the internet just works. BT Technology Journal (2006) vol. 24 (3) pp. 119–129.

[15] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[16] John C. Doyle, David L. Alderson, Lun Li, Steven Low, Matthew Roughan, Stanislav Shalunov, Reiko Tanaka, and Walter Willinger. The "robust yet fragile" nature of the internet. *Proceedings of the National Academy of Sciences of the United States of America*, 102(41):14497–14502, October 2005.

[17] D. Clark, L. Chapin, V. Cerf, R. Braden, and R. Hobby. Towards the Future Internet Architecture. RFC 1287 (Informational), December 1991.

[18] Scott Shenker. Fundamental design issues for the future internet (invited paper). *IEEE Journal on Selected Areas in Communications*, 13(7):1176–1188, 1995.

[19] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12, New York, NY, USA, 2009. ACM.

[20] Dirk Trossen, Mikko Särelä, and Karen R. Sollins. Arguments for an information-centric internetworking architecture. *Computer Communication Review*, 40(2):26–33, 2010.

[21] Jon Crowcroft. Toward a network architecture that does everything. *Commun. ACM*, 51(1):74–77, 2008.

[22] V. Jacobson. If a clean slate is the solution what was the problem? Stanford "Clean Slate" Seminar., Feb 2006.

[23] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. *SIGCOMM Comput. Commun. Rev.*, 38(4):219–230, 2008.

[24] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*, 2007.

[25] Jennifer Rexford and Constantine Dovrolis. Future internet architecture: clean-slate versus evolutionary research. *Commun. ACM*, 53(9):36–40, 2010.

[26] Teemu Koponen. *A Data-Oriented Network Architecture*. PhD thesis, Helsinki University of Technology, October 2008.

[27] Gero MÃ¼hl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische UniversitÃ¤t Darmstadt, 2002.

[28] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. Rofl: routing on flat labels. *SIGCOMM' 06*, 2006.

[29] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable internet protocol (AIP). In *SIGCOMM' 08*, 2008.

[30] Dirk Trossen (edit.). Conceptual architecture of PSIRP including subcomponent descriptions. Deliverable D2.2, PSIRP project, 2008.

[31] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[32] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis. Locator/ID separation protocol (LISP). Work in progress (draft-ietf-lisp-07), April 2010.

[33] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (cbt). *SIGCOMM Comput. Commun. Rev.*, 23(4):85–95, 1993.

[34] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), August 2006.

[35] Ralph P. Grimaldi and Rose-Hulman. *Discrete and Combinatorial Mathematics; An Applied Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1985.

[36] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[37] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 125–133, New York, NY, USA, 1996. ACM.

[38] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. *SIGCOMM Comput. Commun. Rev.*, 27(4):3–14, 1997.

[39] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed ip routing lookups. *SIGCOMM Comput. Commun. Rev.*, 27(4):25–36, 1997.

[40] David E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, 2005.

[41] Christine Maindorfer. *Algorithms and data structures for IP lookup, packet classification and conflict detection*. PhD thesis, University of Freiburg, 2009.

[42] Sailesh Kumar, Jonathan Turner, Patrick Crowley, and Michael Mitzenmacher. Hexa: Compact data structures for faster packet processing. *Network Protocols, IEEE International Conference on*, 0:246–255, 2007.

[43] Weidong Wu. *Packet Forwarding Technologies*. Auerbach Publications, Boston, MA, USA, 2007.

[44] A. Kirsch, M. Mitzenmacher, and G. Varghese. *Algorithms for Next Generation Networks, Computer Communications and Networks*, chapter Hash-Based Techniques for High-Speed Packet Processing, pages 181–218. Springer-Verlag, Feb 2010.

[45] George Varghese. *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[46] Dmitri Krioukov, k c claffy, Kevin Fall, and Arthur Brady. On compact routing for the internet. *SIGCOMM Comput. Commun. Rev.*, 37(3):41–52, 2007.

[47] Ittai Abraham, Dahlia Malkhi, and David Ratajczak. Compact multicast routing. In *DISC'09: Proceedings of the 23rd international conference on Distributed computing*, pages 364–378, Berlin, Heidelberg, 2009. Springer-Verlag.

[48] Martin Karsten, S. Keshav, Sanjiva Prasad, and Mirza Beg. An axiomatic basis for communication. *SIGCOMM Comput. Commun. Rev.*, 37(4):217–228, 2007.

[49] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic Routing Encapsulation (GRE). RFC 1701 (Informational), October 1994.

[50] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[51] Mikko Särelä, Christian Esteve Rothenberg, Tuomas Aura, Andras Zahemszky, Pekka Nikander, and Jörg Ott. Forwarding Anomalies in Bloom Filter Based Multicast. Technical report, Aalto University, October 2010. Submitted for publication.

[52] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrah, Sushil Singh, and George Varghese. Beyond bloom filters: from approximate membership checks to approximate state machines. In *ACM SIGCOMM*, 2006.

[53] Carl A. Sunshine. Source routing in computer networks. *SIGCOMM Comput. Commun. Rev.*, 1977.

[54] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM CCR*, 1989.

[55] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP multicast. In *SIGCOMM'06*, 2006.

[56] Tilman Wolf. Data path credentials for high-performance capabilities-based networks. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 129–130, New York, NY, USA, 2008. ACM.

[57] Fan Ye, Haiyun Luo, Songwu Lu, Lixia Zhang, and Senior Member. Statistical en-route filtering of injected false data in sensor networks. In *INFOCOM*, pages 839–850, 2004.

[58] Rafael P. Laufer, Pedro B. Velloso, Daniel de O. Cunha, Igor M. Moraes, Marco D. D. Bicudo, Marcelo D. D. Moreira, and Otto Carlos M. B. Duarte. Towards stateless single-packet IP traceback. In *LCN '07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 548–555, Washington, DC, USA, 2007. IEEE Computer Society.

[59] A Whitaker and D Wetherall. Forwarding without loops in Icarus. In *Proc. of OPENARCH*, 2002.

[60] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 2005.

[61] Paul Hurley and Marcel Waldvogel. Bloom filters: One size fits all? In *LCN '07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 183–190, Washington, DC, USA, 2007. IEEE Computer Society.

[62] S. Lumetta and M. Mitzenmacher. Using the power of two choices to improve Bloom filters. Accepted to Internet Mathematics. Preprint at http://www.eecs.harvard.edu/m̃ichaelm.

[63] M. Jimeno, K. Christensen, A. Roginsk. A power management proxy with a new best-of-n bloom filter design to reduce false positives. In *21st IEEE International Performance, Computing, and Communications Conference*, pages 125–133, 2002.

[64] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting Bloom filters. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 684–695, London, UK, 2006. Springer-Verlag.

[65] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, and P. Nikander. Mpss: Multiprotocol stateless switching. pages 1 –6, mar. 2010.

[66] Martín Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, and Scott Shenker. Rethinking enterprise network control. *IEEE/ACM Trans. Netw.*, 17(4):1270–1283, 2009.

[67] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*, 2009.

[68] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM CCR*, 39(4):51–62, 2009.

[69] Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying nox to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[70] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *SIGCOMM CCR*, 35(5):41–54, 2005.

[71] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[72] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 25(1):157–187, 1995.

[73] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[74] National Research Council. *Looking Over the Fence at Networks: A Neighbor's View of Networking Research.* National Academies Press, Washington, D.C.,, 2001.

[75] Christophe Diot, Walid Dabbous, and Jon Crowcroft. Multipoint communication: A survey of protocols, functions, and mechanisms. *IEEE Journal on Selected Areas in Communications*, 15(3):277–290, 1997.

[76] Pragyansmita Paul and S. V. Raghavan. Survey of multicast routing algorithms and protocols. In *ICCC '02: Proceedings of the 15th international conference on Computer communication*, pages 902–926, Washington, DC, USA, 2002. International Council for Computer Communication.

[77] Beichuan Zhang, Sugih Jamin, and Lixia Zhang. Universal ip multicast delivery. In *in Proceedings of the International Workshop on Networked Group Communication (NGC*, pages 781–806, 2002.

[78] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[79] Hugh W. Holbrook and David R. Cheriton. Ip multicast channels: Express support for large-scale single-source applications. pages 65–78, 1999.

[80] Shivkumar Kalyanaraman. Network architecture: Principles, guidelines (reference notes). Online: http://www.ecse.rpi.edu/Homepages/shivkuma/teaching/sp2006/archdetailednotes.doc.

[81] John Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, December 2007.

[82] Jennifer Rexford, Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Geoffrey Xie, Jibin Zhan, and Hui Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *In Proceedings of HotNets III*, 2004.

[83] Hong Yan, David A. Maltz, T. S. Eugene Ng, Hemant Gogineni, Hui Zhang, and Zheng Cai. Tesseract: A 4d network control plane. In *NSDI'07*, 2007.

[84] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. The case for separating routing from routers. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 5–12, New York, NY, USA, 2004. ACM.

[85] Nick Feamster, Hari Balakrishnan, and Jennifer Rexford. Some foundational problems in interdomain routing. In *In HotNets, 2004. (Cited on*, pages 41–46, 2004.

[86] Aurel A. Lazar. Programming telecommunication networks. *IEEE Network*, 11:8–18, 1997.

[87] David L Tennenhouse, Jonathan M Smith, W. David Sincoskie, David J Wetherall, and Gary J Minden. A survey of active network research. *IEEE COMMUNICATIONS MAGAZINE*, 35:80—86, 1997.

[88] L. Yang, R. Dantu, T. Anderson, and R. Gopal. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746 (Informational), April 2004.

[89] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, 2008.

[90] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A knowledge plane for the internet. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10, New York, NY, USA, 2003. ACM.

[91] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[92] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technol.*, 1(1):70–109, 2001.

[93] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, New York, NY, USA, 2001. ACM.

[94] Mingdong Tang, Jing Yang, and Guoqing Zhang. Compact routing on random power law graphs. *Dependable, Autonomic and Secure Computing, IEEE International Symposium on*, 0:575–578, 2009.

[95] Cedric Westphal and James Kempf. A compact routing architecture for mobility. In *MobiArch '08: Proceedings of the 3rd international workshop on Mobility in the evolving internet architecture*, pages 1–6, New York, NY, USA, 2008. ACM.

[96] Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. Compact routing in power-law graphs. In *DISC'09: Proceedings of the 23rd international conference on Distributed computing*, pages 379–391, Berlin, Heidelberg, 2009. Springer-Verlag.

[97] Lenore J. Cowen. Compact routing with minimum stretch. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[98] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: lower bounds. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 207–216, New York, NY, USA, 2006. ACM.

[99] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

[100] Girish P. Chandranmenon and George Varghese. Trading packet headers for packet processing. *IEEE/ACM Trans. Netw.*, 4(2):141–152, 1996.

[101] Charles Kalmanek. A retrospective view of atm. *SIGCOMM Comput. Commun. Rev.*, 32(5):13–19, 2002.

[102] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala. Source Demand Routing: Packet Format and Forwarding Specification (Version 1). RFC 1940 (Informational), May 1996.

[103] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.

[104] Peter Newman, Greg Minshall, and Thomas L. Lyon. Ip switching—atm under ip. *IEEE/ACM Trans. Netw.*, 6(2):117–129, 1998.

[105] Alan Shieh, Andrew C. Myers, and Emin Gün Sirer. A stateless approach to connection-oriented protocols. *ACM Trans. Comput. Syst.*, 26(3):1–50, 2008.

[106] Tuomas Aura and Pekka Nikander. Stateless connections. In *ICICS '97: Proceedings of the First International Conference on Information and Communication Security*, pages 87–97, London, UK, 1997. Springer-Verlag.

[107] Guozhen Tan, Hengwei Yao, Yi Liu, and Ningning Han. Qos provision for ipv6 traffic using dynamic packet state. In *ICAS-ICNS '05: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, page 33, Washington, DC, USA, 2005. IEEE Computer Society.

[108] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering. IPv6 Flow Label Specification. RFC 3697 (Proposed Standard), March 2004.

[109] Anat Bremler-Barr, Yehuda Afek, and Sariel Har-Peled. Routing with a clue. *SIGCOMM Comput. Commun. Rev.*, 29(4):203–214, 1999.

[110] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722.

[111] Q. Hu and B. Carpenter. Survey of proposed use cases for the ipv6 flow label. Independent Submission (draft-hu-flow-label-cases-00), April 2010.

[112] D. Fedyk, L. Berger, and L. Andersson. Generalized Multiprotocol Label Switching (GMPLS) Ethernet Label Switching Architecture and Framework. RFC 5828 (Informational), March 2010.

[113] Lucian Popa, Ion Stoica, and Sylvia Ratnasamyl. Rule-based forwarding (rbf): improving the internet's flexibility and security. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[114] Ken Calvert. Reflections on network architecture: an active networking perspective. *SIGCOMM Comput. Commun. Rev.*, 36(2):27–30, 2006.

[115] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94, 2007.

[116] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *SIGCOMM'02*, 2002.

[117] Jerome H. Saltzer. Naming and binding of objects. In *Operating Systems, An Advanced Course*, pages 99–208, London, UK, 1978. Springer-Verlag.

[118] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–86, New York, NY, USA, 2002. ACM.

[119] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the internet. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 343–352, New York, NY, USA, 2004. ACM.

[120] D. Clark, R. Braden, A. Falk, and V. Pingali. Fara: Reorganizing the addressing architecture. *ACM SIGCOMM Computer Communication Review*, pages 313–321, 2003.

[121] D. R. Cheriton and M. Gritter. Triad: A new next-generation internet architecture. `http://www-dsg.stanford.edu/triad/`, July 2000.

[122] Paul Francis Ramakrishna. Ipnl: A nat-extended internet architecture. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 69–80, New York, NY, USA, 2001. ACM.

[123] Bengt Ahlgren, Jari Arkko, Lars Eggert, and Jarno Rajahalme. A node identity internetworking architecture. In *INFOCOM*. IEEE, 2006.

[124] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. *SIGCOMM Comput. Commun. Rev.*, 40(4):75–86, 2010.

[125] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. The flattening internet topology: natural evolution, unsightly barnacles or contrived collapse? In *PAM'08: Proceedings of the 9th international conference on Passive and active network measurement*, pages 1–10, Berlin, Heidelberg, 2008. Springer-Verlag.

[126] Krishna Gummadi. Designing an internet for content delivery and not communication. NeXtworking07, 2nd COST-NSF Workshop on Future Internet, Berlin, Germany., April 2007.

[127] Serge Fdida. The network is a database. NeXtworking07 , 2nd COST-NSF Workshop on Future Internet, Berlin, Germany, April 2007.

[128] Alex C. Snoeren, Kenneth Conley, and David K. Gifford. Mesh-based content routing using xml. *SIGOPS Oper. Syst. Rev.*, 35(5):160–173, 2001.

[129] Van Jacobson. A new way to look at networking. Google Tech Talks, August 2006. http://video.google.com/videoplay?docid=-6972678839686672840&q=van+jacobson&pr=goog-sl.

[130] Sanjoy Paul. Postcards from the edge: A cache-and-forward architecture for the future internet. NeXtworking07, 2nd COST-NSF Workshop on Future Internet, Berlin, Germany, April 2007.

[131] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. Virtual ring routing: network routing inspired by dhts. *SIGCOMM Comput. Commun. Rev.*, 36(4):351–362, 2006.

[132] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. Rofl: routing on flat labels. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 363–374, New York, NY, USA, 2006. ACM.

[133] Novella Bartolini, Emiliano Casalicchio, and Salvatore Tucci. A walk through content delivery networks. In Mariacarla Calzarossa and Erol Gelenbe, editors, *MASCOTS Tutorials*, volume 2965 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2003.

[134] Al-Mukaddim Khan Pathan and Rajkumar Buyya. A taxonomy and survey of content delivery networks. Technical report, 2007.

[135] M. Freeman. *Democratizing Content Distribution*. PhD thesis, New York University, September 2007. Supervised by David MaziÃ¨res.

[136] Peter Hebden and Adrian Pearce. Data-centric routing using bloom filters in wireless sensor networks. In M. Palaniswami, editor, *Fourth International Conference on Intelligent Sensing and Information Processing (ICISIP-06),*, pages 72–78, Bangalore, India, 2006. IEEE Press.

[137] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, New York, NY, USA, 2003. ACM.

[138] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

[139] R. Baldoni, M. Contenti, S. Tucci Piergiovanni, and A. Virgillito. Modelling publish/subscribe communication systems: Towards a formal approach. *words*, 00:304, 2003.

[140] Y. Liu and B. Plale. Survey of publish subscribe event systems, 2003.

[141] Christos Tryfonopoulos, Stratos Idreos, and Manolis Koubarakis. Publish/subscribe functionality in ir environments using structured overlay networks. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 322–329, New York, NY, USA, 2005. ACM.

[142] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[143] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the internet. In *SIGCOMM '04*, 2004.

[144] D. Trossen (ed.). Conceptual Architecture of PSIRP Including Subcomponent Descriptions (D2.2). http://psirp.org/publications, June 2008.

[145] FP7 4WARD Project. D2.1 Technical Requirements. http://www.4ward-project.eu/, Aug 2008.

[146] J.J. Garcia-Luna-Aceves. Content-centric networking. IETF, DTNRG, March 2006.

[147] James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. Haggle: a networking architecture designed around mobile users. In *Annual IFIP Conference on Wireless On-demand Network Systems and Services*, 2006.

[148] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, 2008.

[149] Sivakumar Ganapathy and Tilman Wolf. Design of a network service architecture. In *Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2007.

[150] Tilman Wolf. Service-centric end-to-end abstractions in next-generation networks. In *Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2006.

[151] Michael Demmer, Kevin Fall, Teemu Koponen, and Scott Shenker. Towards a modern communications api. In *Proceedings of HotNets-VI*, 2007.

[152] H.T. Kung and C.H. Wu. *The Internet as a Large-Scale Complex System, (Editors), published by as part of Santa Fe Institute series,*, chapter Content Networks: Taxonomy and New Approaches, pages 203–225. Oxford University Press, 2002.

[153] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Content-based addressing and routing: A general model and its application, 2000.

[154] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles*, Charleston, SC, December 1999.

[155] Mehmet Altinel and Michael J. Franklin. Efficient filtering of xml documents for selective dissemination of information. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 53–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[156] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[157] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. *Distributed Computing Systems, International Conference on*, 0:0262, 1999.

[158] Antonino Virgillito, Roberto Beraldi, and Roberto Baldoni. On event routing in content-based publish/subscribe through dynamic networks. *Future Trends of Distributed Computing Systems, IEEE International Workshop*, 0:322, 2003.

[159] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[160] Zihui Ge, Ping Ji, Jim Kurose, and Don Towsley. Matchmaker: Signaling for dynamic publish/subscribe applications. In *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*, page 222, Washington, DC, USA, 2003. IEEE Computer Society.

[161] Antonio Carzaniga, Aubrey J. Rembert, and Alexander L. Wolf. Understanding content-based routing schemes. Technical Report 2006/05, Faculty of Informatics, University of Lugano, September 2006.

[162] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, Karlsruhe, Germany, August 2003.

[163] H. Holbrook and B. Cain. Source-specific multicast for IP. RFC 4607. 2006.

[164] S.E. Deering. Host extensions for IP multicasting. RFC 1112 (Standard), August 1989. Updated by RFC 2236.

[165] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 2003.

[166] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. In *INFOCOM' 2000*, 2000.

[167] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms. Explicit multicast (Xcast) concepts and options. IETF RFC 5058, 2007.

[168] Leonid B. Poutievski, Kenneth L. Calvert, and James N. Griffioen. Routing and forwarding with flexible addressing. *Journal Of Communication and Networks*, 2007.

[169] K. L. Calvert, J. Griffioen, and L. Poutievski. Separating Routing and Forwarding: A Clean-Slate Network Layer Design. In *In proc. of the Broadnets Conf.*, 2007.

[170] Craig Shue and Minaxi Gupta. Packet forwarding: Name-based vs. prefix-based. In *IEEE Global Internet Symposium*, May 2007.

[171] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 3–14, New York, NY, USA, 2008. ACM.

[172] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: The implications of universal redundant traffic elimination. In *ACM SIGCOMM*, 2008.

[173] Van Jacobson and Diana Smetters. Securing network content. Technical report, PARC, Oct 2009. 2009.

[174] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. *SIGOPS Oper. Syst. Rev.*, 34(2):19–20, 2000.

[175] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2001.

[176] Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, 2002.

[177] Bogdan C. Popescu, Maarten van Steen, Bruno Crispo, Andrew S. Tanenbaum, Jan Sacha, and Ihor Kuz. Securely replicated web documents. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 104.2, Washington, DC, USA, 2005. IEEE Computer Society.

[178] Hari Balakrishnan, Scott Shenker, and Michael Walfish. Semantic-Free Referencing in Linked Distributed Systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, February 2003.

[179] Michael Walfish, Hari Balakrishnan, and Scott Shenker. Untangling the web from dns. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 17–17, Berkeley, CA, USA, 2004. USENIX Association.

[180] Z. Wilcox-O'Hearn. Names: Decentralized, secure, human-meaningful: Choose two. Online: http://zooko.com/distnames.html, Sept 2003.

[181] M. Stiegler. An introduction to petname systems. Online: http://www.skyhunter.com/marcs/petnames/IntroPetNames.html, Feb 2005. updated June 2010.

[182] Ganesh Ananthanarayanan, Kurtis Heimerl, Matei Zaharia, Michael Demmer, Teemu Koponen, Arsalan Tavakoli, Scott Shenker, and Ion Stoica. Enabling innovation below the communication api. Technical Report UCB/EECS-2009-141, EECS Department, University of California, Berkeley, Oct 2009.

[183] Christian Vogt. Improving the scalability of internet routing. In Tania Tronco, editor, *New Network Architectures*, volume 297 of *Studies in Computational Intelligence*, pages 189–203. Springer Berlin / Heidelberg, 2010.

[184] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. pages 1–12, 2009.

[185] PSIRP Project Team. Psirp project home page. http://www.psirp.org.

[186] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *ACM MobiArch '08*, 2008.

[187] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. Buffalo: Bloom filter forwarding architecture for large organizations. In *Conext*, 2009.

[188] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[189] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[190] Rasmus Pagh and Flemming Friche Rodler. Lossy dictionaries. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 300–311, London, UK, 2001. Springer-Verlag.

[191] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[192] Sasu Tarkoma and Christian Esteve Rothenbergand Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *To appear in IEEE Communications Surveys and Tutorials*.

[193] Anna Ostlin and Rasmus Pagh. Uniform hashing in constant time and linear space. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 622–628, New York, NY, USA, 2003. ACM.

[194] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA '08*.

[195] Christian Henke, Carsten Schmoll, and Tanja Zseby. Empirical evaluation of hash functions for multipoint measurements. *SIGCOMM Comput. Commun. Rev.*, 38(3):39–50, 2008.

[196] George Marsaglia and Wai Wan Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):37–51, 2002.

[197] Fang Hao, Murali Kodialam, and T. V. Lakshman. Building high accuracy bloom filters using partitioned hashing. In *SIGMETRICS '07*, 2007.

[198] Benoit Donnet, Bruno Baynat, and Timur Friedman. Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives. In *CoNEXT '06*, New York, NY, USA, 2006.

[199] Paul Hurley and Marcel Waldvogel. Bloom filters: One size fits all? *Local Computer Networks, Annual IEEE Conference on*, 0:183–190, 2007.

[200] Ely Porat. An optimal bloom filter replacement based on matrix solving. In *CSR '09: Proceedings of the Fourth International Computer Science Symposium in Russia on Computer Science - Theory and Applications*, pages 263–273, Berlin, Heidelberg, 2009. Springer-Verlag.

[201] Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership (extended abstract). In *ICALP '08: Proceedings of the 35th international collo-*

quium on Automata, Languages and Programming, Part I, pages 385–396, Berlin, Heidelberg, 2008. Springer-Verlag.

[202] Felix Putze, Peter Sanders, and Johannes Singler. Cache-, hash- and space-efficient bloom filters. In *WEA'07: Proceedings of the 6th international conference on Experimental algorithms*, pages 108–121, Berlin, Heidelberg, 2007. Springer-Verlag.

[203] Rasmus Pagh. Cuckoo hashing. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

[204] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 823–829, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[205] Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve IP lookups. In *INFOCOM*, pages 1454–1463, 2001.

[206] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212, New York, NY, USA, 2003. ACM.

[207] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. Fast hash table lookup using extended bloom filter: an aid to network processing. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 181–192, New York, NY, USA, 2005. ACM.

[208] Björn Grönvall. Scalable multicast forwarding. *SIGCOMM Comput. Commun. Rev.*, 2002.

[209] H. Tahilramani Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. Bananas: an evolutionary framework for explicit and multipath routing in the internet. *SIGCOMM Comput. Commun. Rev.*, 2003.

[210] A. Whitaker and D. Wetherall. Forwarding without Loops in Icarus. In *Open Architectures and Network Programming*, pages 63–75, 2002.

[211] Ioannis Aekaterinidis and Peter Triantafillou. Publish-subscribe information delivery with substring predicates. *IEEE Internet Computing*, 11(4):16–23, 2007.

[212] Zbigniew Jerzak and Christof Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS '08*, 2008.

[213] Peter Triantafillou and Andreas Economides. Subscription summaries for scalability and efficiency in publish/subscribe systems. In J. Bacon, L. Fiege, R. Guerraoui, A. Jacobsen, and G. Mühl, editors, *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, 2002.

[214] Peter Triantafillou and Andreas Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *ICDCS*, pages 562–571, 2004.

[215] Xueqing Gong, Weining Qian, Ying Yan, and Aoying Zhou. Bloom filter-based xml packets filtering for millions of path queries. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 890–901, Washington, DC, USA, 2005. IEEE Computer Society.

[216] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd S. Sproull, and John W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro*, 24(1):52–61, 2004.

[217] Francis Chang, Kang Li, and Wu chang Feng. Approximate caches for packet classification. In *INFOCOM*, 2004.

[218] Yuhua Chen and Oladapo Oguntoyinbo. Power efficient packet classification using cascaded bloom filter and off-the-shelf ternary cam for wdm networks. *Comput. Commun.*, 32(2):349–356, 2009.

[219] Mahmood Ahmadi and Stephan Wong. Modified collision packet classification using counting bloom filter in tuple space. In *PDCN'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, pages 315–320, Anaheim, CA, USA, 2007. ACTA Press.

[220] Sarang Dharmapurikar, Haoyu Song, Jonathan Turner, and John Lockwood. Fast packet classification using bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 61–70, New York, NY, USA, 2006. ACM.

[221] Haoyu Song, Jonathan S. Turner, and Sarang Dharmapurikar. Packet classification using coarse-grained tuple spaces. In *ANCS*, pages 41–50, 2006.

[222] Eugene H. Spafford. OPUS: preventing weak password choices. *Comput. Secur.*, 11(3):273–278, 1992.

[223] Udi Manber and Sun Wu. An algorithm for approximate membership checking with application to password security. *Inf. Process. Lett.*, 50(4):191–197, 1994.

[224] Steven M. Bellovin and William R. Cheswick. Privacy-enhanced searches using encrypted bloom filters. Technical Report CUCS-034-07, Columbia University and AT&T, 2004.

[225] Marcos K. Aguilera, Minwen Ji, Mark Lillibridge, John MacCormick, Erwin Oertli, Dave Andersen, Mike Burrows, Timothy Mann, and Chandramohan A. Thekkath. Block-level security for network-attached disks. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 159–174, Berkeley, CA, USA, 2003. USENIX Association.

[226] Vassil Roussev, Yixin Chen, Timothy Bourg, and Golden G. Richard III. md5bloom: Forensic filesystem hashing revisited. *Digital Investigation*, 3(Supplement-1):82–90, 2006.

[227] Michael Attig, Sarang Dharmapurikar, and John W. Lockwood. Implementation results of bloom filters for string matching. In *FCCM*, pages 322–323, 2004.

[228] Gianni Antichi, Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci. Counting bloom filters for pattern matching and anti-evasion at the wire speed. *IEEE Network*, 23(1):30–35, 2009.

[229] Edmund L. Wong, Praveen Balasubramanian, Lorenzo Alvisi, Mohamed G. Gouda, and Vitaly Shmatikov. Truth in advertising: lightweight verification of route integrity. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 147–156, New York, NY, USA, 2007. ACM.

[230] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. Off by default! In *Proc. 4th ACM Workshop on Hot Topics in Networks (Hotnets-IV)*, College Park, MD, November 2005.

[231] C Dixon and T Anderson. Phalanx: Withstanding multimillion-node botnets. *Usenix NSDI*, 2008.

[232] XiaoFeng Wang and Michael K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 257–267, New York, NY, USA, 2004. ACM.

[233] Tilman Wolf. A credential-based data path architecture for assurable global networking. In *Proc. of IEEE MILCOM*, Orlando, FL, October 2007.

[234] Kui Ren, Wenjing Lou, and Yanchao Zhang. Multi-user broadcast authentication in wireless sensor networks. In *SECON*, pages 223–232, 2007.

[235] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. *IEEE/ACM Trans. Netw.*, 10(6):721–734, 2002.

[236] Kulesh Shanmugasundaram, Hervé Brönnimann, and Nasir Memon. Payload attribution via hierarchical bloom filters. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 31–41, New York, NY, USA, 2004. ACM.

[237] Minho Sung, Jun Xu, Jun Li, and Li Li. Large-scale ip traceback in high-speed internet: practical techniques and information-theoretic foundation. *IEEE/ACM Trans. Netw.*, 16(6):1253–1266, 2008.

[238] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *SIGMETRICS*, pages 121–132, 2008.

[239] Carlos Alberto Bráz Macapuna, Christian Esteve Rothenberg, and MauríÃÂcio Magalháes. In-packet bloom filter based data center networking with distributed OpenFlow controllers. In *IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010)*, Miami, Florida, USA, 12 2010.

[240] Christian Esteve Rothenberg and Andreas Roos. A review of policy-based resource and admission control functions in evolving access and next generation networks. *J. Netw. Syst. Manage.*, 16:14–45, March 2008.

[241] Jiayue He, Jennifer Rexford, and Mung Chiang. Don't optimize existing protocols, design optimizable protocols. *SIGCOMM Comput. Commun. Rev.*, 37(3):53–58, 2007.

[242] L.G. Roberts. A radical new router. *IEEE Spectrum*, 46:34–39, July 2009.

[243] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[244] A. Farrel, J.-P. Vasseur, and J. Ash. A Path Computation Element (PCE)-Based Architecture. RFC 4655 (Informational), August 2006.

[245] Konstantinos Katsaros, George Xylomenos, and George C. Polyzos. Multicache: An overlay architecture for information-centric networking. *Computer Networks*, In Press, Corrected Proof:–, 2010.

[246] L. Zhang et al. Named data networking (ndn) project. Technical Report NDN-0001, PARC Technical Report, October 2010.

[247] Mikko Särelä, Jörg Ott, and Jukka Ylitalo. Fast inter-domain mobility with in-packet bloom filters. In *MobiArch '10: Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture*, pages 9–14, New York, NY, USA, 2010. ACM.

[248] András Zahemszky and Somaya Arianfar. Fast reroute for stateless multicast. In *ICUMT*, pages 1–6. IEEE, 2009.

[249] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proc. of the 9th Symposium on Operating Systems Design and Implementation*, Oct 2010.

[250] Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. Quagflow: partnering quagga with openflow. In *SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pages 441–442, New York, NY, USA, 2010. ACM.

[251] PURSUIT Project Team. Pursuit project home page. http://www.fp7-pursuit.eu.

Appendix A

Publication A

C. Esteve Rothenberg, F. Verdi and M. Magalhães. “Towards a new generation of information-oriented internetworking architectures.” In *ACM CoNext, First Workshop on Re-Architecting the Internet (Re-Arch08)*, Dec. 2008, Madrid, Spain.

Towards a new generation of information-oriented internetworking architectures

Christian Esteve, Fábio L. Verdi and Maurício F. Magalhães
University of Campinas (UNICAMP)
School of Electrical and Computer Engineering
{chesteve, verdi, mauricio}@dca.fee.unicamp.br

ABSTRACT

In response to the limitations of the Internet architecture when used for applications for which it was not originally designed, a series of *clean slate* efforts have emerged to shape the so-called *future Internet*. Recently, visionary voices have advised a shift in the networking problem under research, moving from seamless host-reachability to *internetworking of information*. We contribute to the healthy debate on future Internet design and discuss ongoing information oriented efforts. Inspired by recent works in Bloom-filter-like data structures, we propose the *SPSwitch* as a novel switching engine to make wire speed forwarding decisions on flat information labels. We address part of the scalability issues in a data-oriented forwarding layer by trading overdeliveries for state reduction and line speed operations.

Categories and Subject Descriptors

C.2.1 [**Packet-switching networks**]: Network communication

1. INTRODUCTION

For a few years, funding agencies around the world have been promoting the research towards the so-called future Internet. *Clean-slate design* has been a buzz term for networking project proposals. However, the lack of palpable results and clear business models raises doubts whether network revolution makes sense at all. Today's use of the Internet reveals well known limitations in terms of mobility, security, address space exhaustion, routing and content delivery efficiency. Nevertheless, it works reasonably well [14]. For the long term, continuously patching the Internet with ad-hoc protocol extensions and overlay solutions seems to be a complex and costly solution.

The Internet has shifted from being a simple host connectivity infrastructure to a platform enabling massive content production and content delivery, transforming the way information is generated and consumed. From its original design, the Internet carries datagrams inserted by sending hosts in a best effort manner, agnostic to the semantics and purpose of the data transport. There is a sense that the network could do more and better given that today's use of the network is about retrieval of named pieces of data (e.g., URL, service, user identity) rather than specific destination host connections [15]. Hence, the enhancements at the internetworking layer should not be limited to QoS or routing efficiency: data *persistence*, *availability* and *authentication* [17] of the data itself leveraged with timeliness are beneficial in-network data-centric capabilities to be embraced from design.

Last decade's efforts towards rearchitecturing the Internet have mainly focused on end-host reachability with novel concepts (e.g., id/loc split) addressing the 'classic' end-to-end security, mobility and routing issues. All of these proposals are more-or-less host centric. However, this trend is changing, and senior researchers that have participated in the Internet development since its beginning, have advised tackling the future Internet problem from an *information interconnection* perspective. Van Jacobsen [15] provides a vision to understand the motivation for a networking revolution; while the first networking generation was about wiring (telephony) and the second generation was about interconnecting wires (TCP/IP), the next generation should be about interconnecting information at large. This architectural shift implies rethinking many fundamentals by handling information (data, content) as a first-class object.

Recent concerning events (and more to come) may potentially promote and accelerate the adoption of new internetworking paradigms. Today's economy is Internet-sensitive, service outages due to DDoS attacks[1] or due to limitations of BGP insecure routing[2] carry important worries and ex-

[1]Internet reports claim potential costs of $31.000 per minute for Amazon's two hour outage in June 2008.
[2]Pakistan Telecom routing mis-configuration for YouTube's address block propagated internationally, breaking the reachability of the popular video service in February 2008.

penses. Additionally, new forms of SPAM and evolving phishing methods are threatening today's so successful IP-based communication's experience.

This paper is certainly not the first to turn into data-oriented networking or to leverage the publish / subscribe communication paradigm. First, we discuss on the significance of future internetworking research (§ 2) and gather a set of newly emerged concepts from ongoing information-oriented Internetworking activities (§ 3) that set the context and motivation of our work. We move a step towards the feasibility of new data-oriented architectures by proposing the SPSwitch, a novel switching application based on recent Bloom-filter-inspired data structures (§ 4), validated through preliminary experimental results (§ 5). The goals of our future work (§ 6) are hardware-friendly forwarding schemes for a global scale information-oriented architecture.

2. RE-ARCHITECTING THE INTERNET

Research to circumvent current Internet limitations can be divided into those advocating a completely new architecture (*clean-slate*), and those defending an *evolutionary* approach due to incremental deployability concerns. From a research perspective, *clean-slate design* does not presume *clean-slate deployment* and aims at innovation through questioning fundamentals.

A key question is to what extent a new paradigm thinking 'out-of-the-TCP/IP-box' for the future network is really necessary, e.g., as packet switching was to circuit switching in the 70's. The reasoning is based on the large scale use of the Internet for dissemination of data [15]. Tons of connected devices are generating and consuming content, without caring about the actual data source as long as integrity and authenticity are assured [17].

We can also observe this shift toward information-centric networking in the momentum of service oriented architectures (SOA) and infrastructures (SOI), XML routers, deep packet inspection (DPI), content delivery networks (CDN) and P2P overlay technologies. A common issue is the necessity to manage a huge quantity of data items, which is a quite different task than reaching a particular host. In today's Internet, forwarding decisions are made not only by IP routers, but also by middleboxes, VLAN switches, MPLS routers, DPIs, load balancers, mesh routing nodes and other cross-layer approaches. Moving down data-centric functions to the lower networking layers could be in tune with the trend in access and backbone technologies represented by the coupling of the dominant Ethernet access protocol and label switched all optical transport networks.

More than an endless discussion around clean-slate design and actual network (r)evolution deployment, what we really need for the future Internetworking is 1) 'clean-slate thinking' beyond the TCP/IP heritage to foster innovation through questioning paradigms; and 2) feasibility work on an information-oriented infrastructure capable of supporting the actual and future demands over the network of networks.

3. PARADIGMS OF INFORMATION ORIENTED INTER-NETWORKING

Until recently, research in a new generation Internet has prompted architectural proposals (e.g., TRIAD, FARA, Plutarch, UIP, IPNL, HIP, ROFL) that mainly aimed at solving the *host reachability problem* by providing more flexible, expressive, and comprehensive naming and addressing frameworks than the Internet hierarchical IP address space. A move towards *information interconnection* can be observed in recent projects addressing the future Internet such as PSIRP [8], 4Ward [12], Trilogy, ICT's FIRE and other activities within EU FP7 and NSF FIND. Data-centric architectural proposals have started to emerge (e.g., DOA, i3, DONA, Haggle) and are similar in spirit to 'peer-to-peer', 'content-delivery', 'sensor' and 'delay-tolerant' networks.

3.1 Information-centric concepts

In *information/content/data - oriented/centric* networks, the flow of messages is driven by the nodes that have expressed their interest and the information identifiers of the messages, rather than by explicit destination host interface names (IP addresses) assigned by senders. Reachability of destinations is not anymore delimited by topological information but by the notion of *information scope* [21]. Having the data location hidden makes the semantics of what defines a sender or receiver of data less relevant than the data itself, intuitively providing enhanced security (e.g., DDoS mitigation) and bridging connectivity challenged underlying networks (e.g., DTN).

The *publish/subscribe* paradigm [11] is a promising trend to instantiate the so sought modern communication API [10] for information-centric systems. Pub/sub systems have been widely studied and employed for specific event-dissemination applications and have appealing characteristics like spatial and temporal decoupling [11]. In Internet-scale topic-based pub/sub internetworking [8, 20, 21], topics are unique information identifiers at different layers in the architecture accommodating different granularities and semantics (e.g., messages, channels, documents) to support every type of communications (e.g., transactional, interactive, etc.).

The suitability and benefits of moving the pub/sub layer downwards into the networking stack is one of the challenging objectives of interest-driven architectures where naming, routing, forwarding and addressing get fresh semantics (see Table 1).

In the envisioned *internetworking service bus*, information objects are first-class citizens introducing a new global unmanaged*namespace*. A form of publication *metadata* information is required to enable the self-authentication of the data, fragmentation, scope delimitation, inter-domain policies, in-network management, caching, and so on [8].

A global namespace for data items enables *caching* capabilities for every type of communications. In comparison, caching over TCP/IP is costly and application-specific. In case of non-mutable information objects caching becomes

Table 1: Concepts of information-oriented networking versus the original Internet design. Rethinking fundamentals.

Original Internet	Information-Oriented / Content-Centric Internetworking
Sender	Content producer (publisher)
Receiver	Content consumer (subscriber)
Sender-based control	Receiver-based control
Client/Server communications	Publish/Subscribe Sender and Receiver uncoupled
Host-to-host	Service access / Information retrieval
Topology / Domain	Information scope
Unicast	Unified uni-, multi- and anycast
Explicit destination	Implicit destination
End-to-End (E2E)	End-to-Data (E2D)
Host name (look-up oriented)	Data/Content name ("search" activity)
Secure channels, host authentication	Integrity and trust derived from the data

trivial, whereas for streaming applications, caching can be seen as long in-network buffers. Hence, the architecture natively plays the role of current CDNs and avoids redundant traffic over network links [1]. Furthermore, a new namespace for information objects could easily accommodate multi-, any-, con- and unicast types of communication in addition to novel forms of network coding to increase the network's efficiency and resilience.

3.2 A few reference architectures

In this section, we briefly introduce the basics of two recent design choices from the EU FP7 Publish/Subscribe Internetworking Routing Paradigm (PSIRP) project [8] we selected as reference architectures.

The RTFM architecture [20] gets its name from the functional building blocks that are recursively applied. The *rendezvous* (R) is in charge of matching subscriptions to publications and information scoping. The *topology* (T) management creates and maintains (sub-optimal) delivery trees used for traffic forwarding, acting both proactively (optimization) and re-actively (on-demand). The *forwarding* (F) functions perform the actual datagram delivery based on label switching techniques. Finally, *mediation* (M) refers to the node-to-node physical data transmission.

A high-level operational overview of the RTFM could be as follows. After a node subscribes to a publication, a distributed rendezvous system (e.g. a type of DHT or semi-hierarchical solution as in DONA [17]) must first find a copy of the publication's metadata. Using the distributed rendezvous structure to route to a copy of the wanted data, the topology management systems are expected to gather enough information to identify the delivery trees needed to forward the actual data to the subscriber(s). Note that the RTF functions are not necessary co-located in nodes and are distributed and recursive in nature.

In the *black box* rendezvous based networking approach [21], the key idea is to regard the network as a collection of black boxes based on a set of recursive rendezvous functions. The boxes operate in trusted domains hiding their internal topology and exposing outwards only labels and interest definitions. Recursivity [9] and *scoped information layers* are pivotal architectural patterns with a major goal: *scalability*. With the same goal but at a lower layer, efficient data structures enabling the data-centric networking functions (e.g., switching, label processing, caching) are called for to achieve the challenging scalability requirements of information-oriented networks heavily based on virtually 'unlimited' set of flat identifiers.

4. FAST FORWARDING ON FLAT LABELS

The overall picture of an information-oriented network architecture is complex and deserves very detailed discussions spanning multiple disciplines (internetworking, network management, semantic layer, etc.). However, there is a common challenge in any data-oriented paradigm: the need to take switching decisions at wire speed (Gbps) based on a large universe of flat (non-topological, non-aggregatable) identifiers (e.g., 256-bit cryptographic hash values).

Related work relying on flat labels includes ROFL [7], a proposal for Internet-scale routing on flat host identifiers based on neat DHT constructs. In our work we focus on flat identifiers with fundamentally different architectural principles (see Table 1). DONA [17] employs flat self-certifying labels for data objects operated by *find/register* primitives over IP networks, whereas our work is more ambitious and could run on top of L2 and L3.

For the sake of generality and the objectives of this paper, we use the term *flat label* for data identifiers or any topology-independent packet header forwarding identifiers.

4.1 Publish/Subscribe Switch

The Publish/Subscribe Switch (SPSwitch) is an abstract switching element that relays messages through strict port-forwarding operations. In a more elaborated design, the SP-Switch performs more complex actions like label switching or querying the cache system.

For the purposes of this work, it is enough to consider the generic problem of having to take switching decisions based on large flat identifiers (labels). Note that output destinations (ports) are not just limited to physical port-in/out interfaces but should be regarded as generic outputs, including also local processes, virtual ports, recursive operations, and cache systems.

In the SPSwitch representation of Figure 1, each possible message output is represented by a Bloom filter [3], forming our first p-bank switching approach (§ 4.3) and a reference switching model for an enhanced data structure (§ 4.4).

4.2 The role of Bloom filters and space efficient probabilistic data structures

Given the huge space and flatness of the information identifiers, our intuition is that Bloom filters and other compact

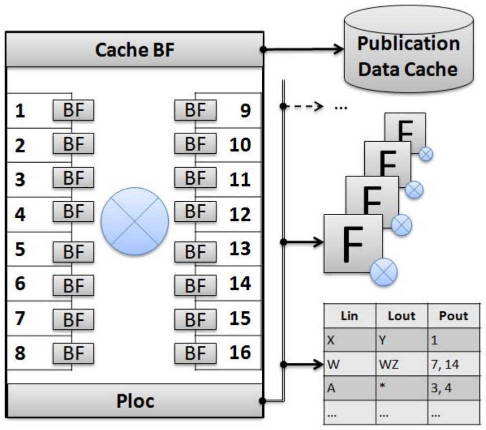


Figure 1: Switching model for incoming datagrams carrying flat labels. Generic destinations (ports) are aggregated in Bloom filters.

hash-based data structures will play a fundamental role as efficient data aggregators in any information-centric architecture. Basically, a Bloom filter (BF) [3] is a space-efficient hashing-based data structure that answers set membership-queries (e.g., *is label*$_x$ *in output* P_y*?*) with some probability of being wrong (false positive rate). BFs are useful whenever you have a set of elements and space is an issue. Then, an approximate representation like a BF may be a powerful alternative if the effects of false positives can be managed. The performance of a BF does not depend at all on the size of the items but on the ratio $memory/elements$. Therefore, hashing-based data structures are an ideal room to handle the large set of flat identifiers. We refer to the large literature on BFs [3, 4, 6, 16] for details and mathematical background.

Bloom filters are commonly used in IP forwarding and other widely studied networking applications (e.g., caches, P2P, measurement, packet classification) [6]. We expect increasingly more useful applications of BFs and its derivatives in new data-intense networking proposals (e.g., Internet accountability [2], flow management [4], credential-based network security [22], IP multicast revisited [19]) with strict performance and memory requirements. The authors of [13] briefly sketched the idea of aggregating active IP multicast addresses per output interface to achieve scalability.

Due to space limitation we do not compare the SPSwitch design with existing hardware designs for fast networking. We are aware that compact hash tables and hashing functions are a daily aid in IP networking. However, there are notable operational differences and challenges (e.g., longest IP prefix vs. long flat identifier matching).

4.2.1 False positives

It is important to place emphasis on the *bounded effect of false positives* in data-centric interest-driven architectures. First, the pub/sub paradigm inherently tolerates false positives, since datagrams corresponding to non subscribed items do not progress in the network and do not create forwarding states. Moreover, end-nodes will only process explicitly subscribed pieces of information. Second, with support for *opportunistic caching*, copies of data can be used to fulfill possible future requests of close by subscribers. Finally, packets forwarded due to false positives are not propagated over many hops due to the large label space and the decreasing probability of consecutive false positives.

4.2.2 Hardware requirements

Since hashing is performed on a per packet basis, the hash tables are implemented directly into the hardware to achieve low processing times ('constant' time to hash and easily parallelized). The position of the element in the memory array can be directly given by the hash value. Built-in dedicated hashing modules are already available in networking hardware. Moreover, we can even skip the hashing operations by taking advantage of the randomness of the hash-based labels under consideration.

In order to achieve to achieve high data rates, memory accesses and computations must be kept to a minimum during packet processing. Routers are limited in expensive high-speed memory. Projections for routers capacity for the next decade [2, 18] let us assume the availability of high speed memory in routers in the order of tens of Mbits[3].

4.3 Naive p-bank Bloom filter approach

Our first natural approach was to define a SPSwitch formed by a bank of BFs, maintaining a BF for each possible output (2^p). The 'control plane' inserts the label(s) in the required output BFs and upon message arrival all possible outputs are queried in parallel to make the forwarding decisions. Recall that a BF does not return false negatives.

After gaining some practical experiences, we identified some limitations of our naive p-bank BF approach inherent to basic BF constructs [4]: a) lack of associated values: just binary probabilistic set-membership responses; b) expensive deletion[4]: counting BFs are costly in memory sizes; c) no notion of time: costly association of filter elements or cells with timing information; d) unbalanced usage of memory per output: unpredictable destination demands difficult the overall system design and memory allocation.

We realized that a more flexible and expressive data structure was required enabling dynamic port-value assignment and per element handling capabilities. Nevertheless, standard BFs are expected to play a role as filters for large caching systems and in other elements of the architecture due to its simplicity, ease of use, and excellent performance.

[3]Typically SRAM, DRAM, (T)CAM and newly RLDRAM. CAMs are expensive fully associative memories - highest available single-chip CAM is 18Mbit [18].

[4]Deletions potentially introduce false negatives. This is inherent to any probabilistic data structure and needs to be carefully handled.

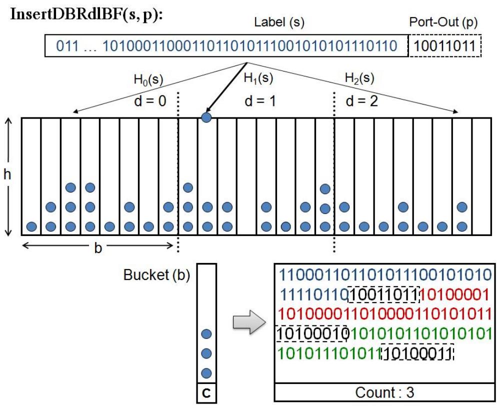


Figure 2: In the stateful d-left Bloom filter, a label fingerprint and its output value are inserted into the least loaded bucket among d candidates.

4.4 Stateful d-left Bloom filter with Dynamic Bit Reassignment

The goal is to build upon a more flexible data structure that could return the output port-value(s) for each programmed label, while still allowing false positives in the form that labels *not in* the set return a value and labels *in* the set return more than one value (aka 'multicast') when only one entry was programmed.

Looking for a candidate structure that could store values (stateful), we took on recent results in BF-inspired data structures by Bonomi et al [4, 5], where the authors define a d-left fingerprint compressed filter (FCF) to track the state of network traffic flows in a dynamic environment. Due to space limitation we refer to [4, 5, 6] for deeper details and rigorous mathematical analysis. In this work, we present our application of the d-left FCF data structure to the specifics of our switching problem and validate it with experimental results.

The d-left scheme is based on the power of two choices (or multiple-choice hashing) and its key feature is yielding near-perfect hash functions, something impractical for dynamic sets. The procedures for *insertion* (label:port) and *look-up* (label) of the stateful d-left FCF are as follows (see Fig.2). A hash table stored in memory is divided into d equal subtables. On *insertion*, d hash functions ($H_i(s)$) uniformly provides a candidate bucket b in each subtable. The least loaded bucket is chosen (breaking ties to the left) to place the item's f-bit fingerprint (e.g., last f bits of the label hash) and the p-bit value. On *look-up*, d buckets need to be checked for fingerprint match and the companion port-value(s) is retrieved.

False positives happen after a fingerprint match when inspecting h elements in d buckets ($Pr = d \cdot h \cdot 2^{-f}$). The neat idea of dynamic bit re-assignment (DBR) [5] is to adjust the size of the fingerprint in function of the bucket's load, yielding better false positive rates due to larger average fingerprints ($f`_{DBR} > f$). The counterpart is having to maintain a counter per bucket and the increased complexity on element insertion due to fingerprint down-resizing (easily implementable with bit shift operations). Further bucket space optimization via semi-sorting of items can be achieved, however, the gain/complexity trade-off is not appealing in our setting where we require to store additional p bits per entry.

5. EXPERIMENTAL RESULTS

The relation between the number of targeted labels *n*, the amount of buckets b and subtables d determines the average load per bucket that can be determined asymptotically as in [5]. By choosing b equal to $n/12$ and $d = 3$, a bucket high $h = 6$ provides a safe margin for overflow ($\approx 10^{-31}$) and a reasonable table utilization $u = n/(dbh) = 2/3$ [4]. Finally, we used the following construction for our experiments: $1M$ 256-bit flat labels (n), 20-bit fingerprints (f) and 10-bit outputs (p). We inserted the n elements into the filters and used a disjoint test set of $10 \cdot n$ labels for counting actual false positives.

Table 2 compares the different forwarding table schemes analytically and includes the predicted calculations and the actual values averaged over 50 experiments. As expected, a simple forwarding table indexed per flat labels has prohibitive costs in both memory and computation. When fixing the available memory, the d-left FCF based approach outperforms the alternative probabilistic structures in actual false positive rates. The gains in performance from the d-choice technique [5] comes from the reduction of the maximum bucket load to about $loglogn/logd$ in the well studied *balls into bins* problem. The DBR optimization results in larger fingerprints (equiv. $f' \approx 22.27$) and thereby substantially lower false positive rates.

We should stress that the beauty of the d-left FCF data structure is not only the gains in terms of memory but in the low and constant memory accesses ($O(1)$). Moreover, a key differentiator of the fingerprint-based approach is to have a powerful probabilistic key (the fingerprint) for managing inserted elements (e.g., querying, updating, deleting).

The actual low performance of the p BFs can be explained due to 1) the fact of aiming very low error rates ($0.6185^{M/n} \approx 10^{-9}$) per BF compared to the theoretical convergence rate $\theta(1/n_{BF})$; and 2) practical issues of the *double hashing* technique [16] to efficiently generate the optimal k hash functions ($ln(2) \cdot M/n \approx 31$).

The actual results are very promising; we may conclude that our envisioned forwarding layer (SPSwitch) based on d-left FCF data structures could evolve to a real system with e.g., 18 Mbits high speed single-chip memories per subtable and handle labels in the order of millions with very low false positive rates (10^{-6}) and bounded worst case performance. The specific system parameters are not so relevant (yet) and mainly serve as a first proof of concept. The most important results are the orders of magnitude and the performance that can be achieved with an optimized d-left FCF inspired data structure.

Table 2: Analytical and experimental comparison of different data structures for the switching procedures.

	Mem. access	Mem. size M	(Mbits)**	(bpe)	False positive	(predicted)**	(actual)**
Standard Table	$O(n) — O(1)$*	$n*(s+p)$	253.68	266.0	0	0	-
Fingerpr. Table	$O(n) — O(1)$*	$n*(f+p)$	28.61	30.00	2^{-f}	$9.54*10^{-7}$	-
p-bank BF	$O(1)$	2^p*m ***	43.63	45.75	$\approx 2^p*0.62^{M/n}$	$2.91*10^{-7}$	$4.33*10^{-3}$
d-left FCF	$O(1)$	$d*b*h*(f+p)$	42.92	45.00	$< d*h*2^{-f}$	$1.72*10^{-5}$	$1.51*10^{-5}$
d-left FCF DBR	$O(1)$	$d*b*(h*(f+p)+c)$	43.63	45.75	$< d*h*2^{-f'}$	$3.57*10^{-6}$	$3.46*10^{-6}$

* Assumes a perfect hash function. ** Parameters: $n = 1.000.008; d = 3; b = 83.334; f = 20; p = 10; h = 6; c = 3; s = 256$.

*** Total memory of the p-bank Bloom filters equal to the value M of the d-left FCF DBR. $m = M/2^p$; $k_{opt} = 31$.

6. FUTURE WORK

Our very next step is to leverage the proposed data structure to work in steady states alternating deletions and insertions, with deletions being handled by timing mechanisms or explicitly. We require further studies on how to apply cache management algorithms (e.g., LRU, LFU) and BF extensions to handle deletions. Upcoming efforts include design optimizations considering memory technology specifics and efficient in/off-chip memory element reallocation. Last but not least, our roadmap includes experimental validation with regard to hardware implementation (NetFPGA) and feasibility on a large scale testbed infrastructure (e.g., Onelab2).

7. CONCLUSIONS

The information-centric usage of today's Internet has changed our daily lives with regard to content generation, consumption and communication patterns. We discussed the relevance of 'clean-slate' research on future Internetworking centered around information and move a step forwards in terms of feasibility. We presented the SPSwitch, a generic forwarding engine based on hashing data structures promising a seedbed of new lines of fast scalable forwarding on flat identifiers. Basically, the SPSwitch trades a small amount of overdeliveries for state reduction and line speed operations. We expect Bloom-filter-inspired systems to play a key role in routing and aggregation of information in data-oriented networks.

Acknowledgements

The interpretation of those ideas made by the authors may or may not agree with the vision of the PSIRP group. We want to thank specially Pekka Nikander and Dirk Trossen for their insightful discussions. We are also thankful to Adam Kirsch for his aid in the field of hash-based data structures. This work is partly funded by CNPq, Capes, FAPESP and Ericsson Research.

8. REFERENCES

[1] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: The implications of universal redundant traffic elimination. In *ACM SIGCOMM*, 2008.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (AIP). In *ACM SIGCOMM*, 2008.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[4] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese. Beyond bloom filters: from approximate membership checks to approximate state machines. In *ACM SIGCOMM*, 2006.

[5] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese. Bloom filters via d-left hashing and dynamic bit reassignment. In *44th Allerton Conference*, 2006.

[6] A. Z. Broder and M. Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1, 2003.

[7] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica. Rofl: routing on flat labels. *SIGCOMM Comput. Commun. Rev.*, 36(4):363–374, 2006.

[8] D. Trossen (ed.). Conceptual Architecture of PSIRP Including Subcomponent Descriptions (D2.2). http://psirp.org/publications, June 2008.

[9] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, December 2007.

[10] M. Demmer, K. Fall, T. Koponen, and S. Shenker. Towards a modern communications api. In *Proceedings of HotNets-VI*, 2007.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[12] FP7 4WARD Project. D2.1 Technical Requirements. http://www.4ward-project.eu/, Aug 2008.

[13] B. Grönvall. Scalable multicast forwarding. *SIGCOMM Comput. Commun. Rev.*, 32(1):68–68, 2002.

[14] M. Handley. Why the internet just works. BT Technology Journal (2006) vol. 24 (3) pp. 119–129.

[15] V. Jacobson. If a clean slate is the solution what was the problem? Stanford "Clean Slate" Seminar., Feb 2006.

[16] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: building a better bloom filter. In *ESA'06.*, pages 456–467, London, UK, 2006. Springer-Verlag.

[17] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.

[18] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits*, 41, 2006.

[19] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting ip multicast. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, New York, NY, USA, 2006. ACM.

[20] M. Särelä, T. Rinta-aho, and T. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, Stockholm., June 2008.

[21] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *ACM MobiArch '08*, 2008.

[22] T. Wolf. A credential-based data path architecture for assurable global networking. In *Proc. of IEEE MILCOM*, Orlando, FL, October 2007.

Appendix B

Publication B

P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. “LIPSIN: Line Speed Publish/Subscribe Inter-Networkings.” In *ACM SIGCOMM’09*, Aug. 2009, Barcelona, Spain.

LIPSIN: Line Speed Publish/Subscribe Inter-Networking

Petri Jokela[1], András Zahemszky[1], Christian Esteve Rothenberg[2],
Somaya Arianfar[1], and Pekka Nikander[1]
[1] Ericsson Research NomadicLab, Finland
{petri.jokela, andras.zahemszky, somaya.arianfar, pekka.nikander}@ericsson.com
[2] University of Campinas (UNICAMP), Brazil
chesteve@dca.fee.unicamp.br

ABSTRACT

A large fraction of today's Internet applications are internally publish/subscribe in nature; the current architecture makes it cumbersome and inept to support them. In essence, supporting efficient publish/subscribe requires data-oriented naming, efficient multicast, and in-network caching. Deployment of native IP-based multicast has failed, and overlay-based multicast systems are inherently inefficient. We surmise that scalable and efficient publish/subscribe will require substantial architectural changes, such as moving from endpoint-oriented systems to information-centric architectures.

In this paper, we propose a novel multicast forwarding fabric, suitable for large-scale topic-based publish/subscribe. Due to very simple forwarding decisions and small forwarding tables, the fabric may be more energy efficient than the currently used ones. To understand the limitations and potential, we provide efficiency and scalability analysis via simulations and early measurements from our two implementations. We show that the system scales up to metropolitan WAN sizes, and we discuss how to interconnect separate networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms

Design

Keywords

Bloom filters, publish/subscribe, multicast, forwarding

1. INTRODUCTION

Many networking applications are internally publish/ subscribe in nature [8]; the actual acts of information creation and consumption are decoupled in time and/or space, and oft en there are multiple simultaneous receivers. For example, RSS feeds, instant messaging, presence services, many typical web site designs, and most middleware systems are either based on a publish/subscribe-like information paradigm or internally implement a publish/subscribe system.

In general, publish/subscribe [15] is a data dissemination method which provides asynchrony between data producers and consumers. Key ingredients include handling data itself as a first class citizen at the naming level, efficient caching to loosen the coupling between producers and consumers in the time dimension, and multicast to efficiently disseminate new data, including both user-published data and system-internal metadata. In addition to pure pub/sub applications, peer-to-peer storage systems and some data-center applications may also benefit from these ingredients [34, 42].

In topic based pub/sub networks, the number of topics is large while each topic may have only a few receivers [24]. IP multicast [13] and application level multicast have scalability and efficiency limitations under such conditions. Similarly, while multicast is a natural choice for data centers, it has the drawback of requiring routers to maintain additional state and performing costly address translations [42]. Hence, the main challenge in efficient pub/sub network design is how to build a multicast infrastructure that can scale to the general Internet and tolerate its failure modes while achieving both low latency and efficient use of resources.

In this paper, we propose a novel multicast forwarding fabric. The mechanism is based on identifying links instead of nodes and using Bloom filters [6] to encode source-route-style forwarding information into the packet header, enabling forwarding without dependency on end-to-end addressing. This provides native support for data-oriented naming and in-network caching. The forwarding decisions are simple and the forwarding tables are small, potentially allowing faster, smaller, and more energy-efficient switches than exists today. The proposed model aims towards balancing the state between the packet headers and the network nodes, allowing both stateless and stateful operations.

The presented method takes advantage of "inverting" the Bloom filter thinking [9]. Instead of maintaining Bloom filters at the network nodes and checking if incoming packets are included in the sets defined by the filters, we put the Bloom filters themselves in the packets and allow the nodes on the path to determine which outgoing links the packet should be forwarded to.

In addition to the design, we briefly describe the two implementations we have built and evaluate the scalability and efficiency of the proposed method with simulations. Further, we give an indication of the potentially achievable speed from our early measurements on our NetFPGA-based implementation.

The rest of this paper is organized as follows. First, in Section 2, we discuss the overall problem and outline the proposed solution. In Section 3, we go into details of the design. Next, in Section 4, we provide scalability evaluation of our forwarding fabric in networks up to metropolitan scales. Section 5 discusses how to inter-connect multiple networks, scaling towards Internet-wide systems, and Section 6 briefly describes our two implementations. Section 7 contrasts our work with related work, and Section 8 concludes the paper.

2. BACKGROUND AND BASIC DESIGN

Our main focus in this paper is on a multicast forwarding fabric for pub/sub-based networking. First, we briefly describe the overall pub/sub architecture our work is based on, and then present our forwarding solution, in the context of that architecture. The presented solution, providing forwarding without end-to-end addressing, is a first step towards an environment preventing DDoS attacks, as the data delivery is based on explicit subscriptions. Finally, at the end of the section, we briefly describe how our proposed forwarding fabric could be used within the present IP architecture.

2.1 A pub/sub-based network architecture

In general, pub/sub provides decoupling in *time*, *space*, and*synchronization* [15]. While publish/subscribe, as such, is well known, it is most often implemented as an overlay. Our work is based on a different approach where the pub/sub view is taken to an extreme, making the whole system based on it. In the work we rely on, inter-networking is based on topic-based publish/subscribe rather than the present send/receive paradigm [32, 39, 41].

The overall pub/sub architecture can be described through a recursive approach, depicted in Figure 1. The same architecture is applied in a recursive manner on the top of itself, each higher layer utilising the rendezvous, topology, and forwarding functions offered by the lower layers; the idea is similar to that of the RNA architecture [20] and the one described by John Day [12]. At the bottom of the architecture lies the forwarding fabric, denoted as "forwarding and more", the main focus of this paper.

The structure can be divided into a data and control plane. At the *control plane*, the topology system creates a distributed awareness of the structure of the network, similar to what today's routing protocols do. On the top of the topology system lies the rendezvous system, which has the responsibility of handling the matching between the publishers and subscribers. The rendezvous does not need to differ substantially from other topic-based pub/sub systems; cf. [15, 23, 36]. Whenever it identifies a publication that has both a publisher (or an up-to-date cache) and one or more active subscribers, it requests the topology system to construct a logical forwarding tree from the present location(s) of the data to the subscribers and to provide the publisher (or the caches) with suitable forwarding information for the data delivery. While being aware of the scalability requirements for rendezvous and topology systems, we do not describe them in details, but refer to our ongoing work in these areas [41, 45].

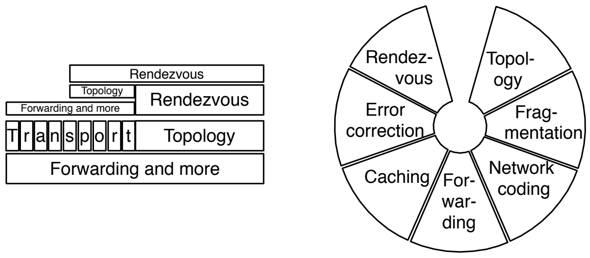


Figure 1: Rendezvous, Topology, Forwarding

The *data plane* takes care of forwarding functionality as well as traditional transport functions, such as error detection and traffic scheduling. In addition to that, a number of new network functions are envisioned (referred to as *more*), such as opportunistic caching [14, 40] and lateral error correction [3].

The data and control plane functions will work in concert, utilizing each other in a *component wheel* [41], similar to the way Haggle managers are organized [33] into an unlayered architecture, providing asynchronous way of communicating between different functional entities in a node.

In this paper, we focus on the forwarding layer, including the required information needed to be passed to it. The rendezvous and topology systems have responsibility for higher-layer operations, such as scalable handling of publish/subscribe requests (multicast tree *join/leave* in IP); they do not affect the forwarding performance directly.

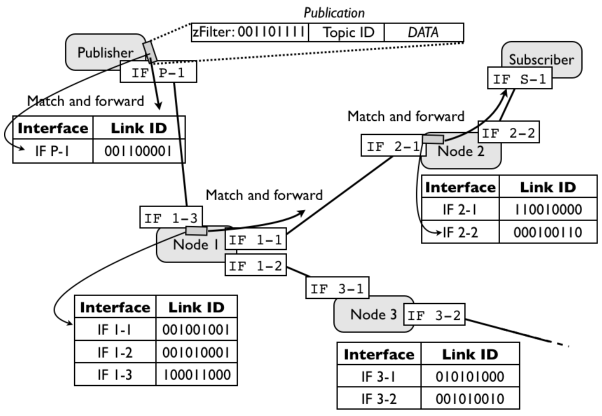
2.2 Recursive bootstrapping

To achieve initial connectivity in the pub/sub network, the rendezvous and topology systems need to be bootstrapped [30]. Bootstrapping is done bottom-up, assuming that the layer below offers (static) connectivity between any node and the rendezvous system. At the lowest layer, this assumption is trivially true, since any two nodes connected by a shared link (wireline or wireless) can, by default, send packets that the other node(s) can receive.

During the bootstrap process, the topology management functions on each node learn their local connectivity, by probing or relying on the underlying layer to provide the information. Then, in a manner similar to the current routing protocols, they exchange information about their perceived local connectivity, creating a map of the network graph structure. The same messages are also used to bootstrap the rendezvous system, allowing the dedicated rendezvous nodes to advertise themselves [32, 41].

2.3 Forwarding on Bloomed link identifiers

In our approach, we do not use end-to-end addresses in the network, and instead of naming nodes, we identify all links with a name. To forward packets through the network, we use a hybrid, Bloom filter based, approach, where the topology system both constructs *forwarding identifiers* by encoding the link identifiers into them in a source routing manner (see Figure 2), and on demand installs new state at the forwarding nodes. In this section, we present the basic ideas in a somewhat simplified form, ignoring a number of details such as loop prevention, error recovery, etc., which are described in Section 3.



Interface	Link ID
IF P-1	001100001

Interface	Link ID
IF 2-1	110010000
IF 2-2	000100110

Interface	Link ID
IF 1-1	001001001
IF 1-2	001010001
IF 1-3	100011000

Interface	Link ID
IF 3-1	010101000
IF 3-2	001010010

Figure 2: Example of Link IDs assigned for links, as well as a publication with a zFilter, built for forwarding the packet from the Publisher to the Subscriber.

For each point-to-point link, we assign two identifiers, called Link IDs, one in each direction. For example, a link between the nodes A and B has two identifiers, $\overrightarrow{AB}$ and $\overleftarrow{AB}$. In the case of a multi-point (e.g. wireless) link, we consider each pair of nodes being connected with a separate link. With this setup, we do not need any common agreement between the nodes on the Link IDs – each Link ID may be locally assigned, as long as the probability of duplicates is low enough.

Basically, a Link ID is an m-bit long name with just k bits set to one. In Section 4 we will discuss the proper values for m and k, and what are the consequences if we change the values. However, for now it is sufficient to note that typically $k \ll m$ and m is relatively large, making the Link IDs statistically unique (e.g., with $m = 248$, $k = 5$, # of Link IDs $\approx m!/(m-k)! \approx 9 * 10^{11}$).

The topology system creates a graph of the network using Link IDs and connectivity information. When it gets a request to determine a forwarding tree for a certain publication, it first creates a conceptual delivery tree using the network graph and the locations of the publisher and subscribers. Once it has such an internal representation of the tree, it knows which links the packets need to pass, and it can determine when to use Link IDs and when to create state [45]. The topology layer is also responsible for reacting to changes in the delivery tree, caused by changes in the subscriber set.

In the default case, we use a source-routing-based approach which makes forwarding independent from routing. Basically, we encode all Link IDs of the tree into a Bloom filter, and place it into the packet header. Once all link IDs have been added to the filter, a mapping from the data topic identifier to the BF is handed to the node acting as the data source and can be used for data delivery along the tree. The representation of the trees in packet headers is source specific and different sources are very likely to use different BFs for reaching the same subscriber sets. To distinguish the BFs in the actual packet headers from other BFs, we refer to the in-packet Bloom filters as zFilters[1].

Each forwarding node acts on packets roughly as follows. For each link, the outgoing Link ID is ANDed with the zFilter in the packet. If the result matches with the Link ID, it is assumed that the Link ID has been added to the zFilter and that the packet needs to be forwarded along that link. With Bloom filters, matching may result with some false positives. In such a case, the packet is forwarded along a link that was not added to the zFilter, causing extra traffic. This sets a practical limit for the number of link names that can be included into a single zFilter.

Our approach to the Bloom filter capacity limit is twofold: Firstly, we use recursive layering [12] to divide the network into suitably-sized components; see Section 5. Secondly, the topology system may dynamically add *virtual links* to the system. A virtual link is, roughly speaking, a unidirectional delivery *tree* that consists of a number of links. It has its own Link ID, similar to the real links. The functionality in the forwarding nodes is identical: the Link ID is compared with the zFilter in the incoming packets, and the packet is forwarded on a match.

2.4 Forwarding in TCP/IP-based networks

While unicast IP packets are forwarded based on address prefixes, the situation is more complicated for multicast. In source specific multicast (SSM) [19], interested receivers join the multicast group (topic) and the network creates specific multicast state based on the join messages. The state is typically reflected in the underlying forwarding fabric, for example, as Ethernet-level multicast groups or multicast forwarding state in MPLS fabrics.

From the IP point of view, LIPSIN can be considered as another underlying forwarding fabric, similar to Ethernet or MPLS. When an IP packet enters a LIPSIN fabric, the edge router prepends a header containing a suitable zFilter, see also Sect. 5.1; similarily, the header is removed at the egress edge. For unicast traffic, the forwarding entry simply contains a pre-computed zFilter, designed to forward the packet through the domain to the appropriate egress edge.

For SSM, the ingress router of the source needs to keep track of the joins received on multicast group through the edge routers, just like any IP multicast router would need to. Hence, it knows the egress edges a multicast packet needs to reach. Based on that information, it can construct a suitable zFilter from the combination of physical or virtual links to deliver the packets, leading to more flexibility and typically less state than in current forwarding fabrics.

3. DESIGN DETAILS AND EXTENSIONS

In this section, we present the details of our link-identity-based forwarding approach. We start by giving a formal description of the heart of the forwarding design, the forwarding decision. Then, we focus on enhancements of the basic design: Link ID Tags generation and selection of candidate Bloom filters. Next, we discuss additional features that make the scheme practical: virtual links, fast recovery after failures, and loop prevention. In the end, we consider control messages and return paths.

3.1 Basic forwarding method

The core of our forwarding method, the forwarding decision, is based on a binary AND and comparison operations,

[1]The name is not due to `zFilter.com` nor the e-mail filter of the same name, but due to one of the authors reading Franquin's Zorglub for the Nth time during the early days of the presented work. The name stuck.

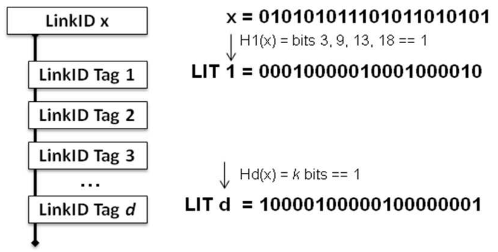

LinkID x
LinkID Tag 1
LinkID Tag 2
LinkID Tag 3
...
LinkID Tag d
x = 01010101110101101010 1
H1(x) = bits 3, 9, 13, 18 == 1
LIT 1 = 00010000010001000010
Hd(x) = k bits == 1
LIT d = 10000100000100000001

Figure 3: An example relation of one Link ID to the d LITs, using k hashes on the Link ID.

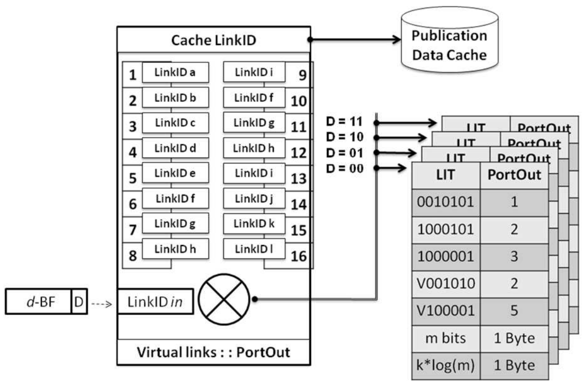


Figure 4: Outgoing interfaces are equipped with d forwarding tables, indexed by the value in the incoming packet.

both of which are very simple to implement in hardware. The base decision (Alg. 1), i.e. whether to forward on a given outbound link or not, can be easily parallelised, as there are no memory or other shared resource bottlenecks. From now on, we build an enhanced system on the top of this simple forwarding operation.

Algorithm 1: Forwarding method of LIPSIN

```
Input: Link IDs of the outgoing links; zFilter in the
       packet header
foreach Link ID of outgoing interface do
    if zFilter & Link ID == Link ID then
        Forward packet on the link
    end
end
```

3.2 Link IDs and LITs

Due to the nature of Bloom filters, a query may return a false positive, leading to a wrong forwarding decision. To reduce the number of false positives, we now introduce *Link ID Tags* (LITs), as an addition to the plain Link IDs. The idea is that instead of each link being identified with a single Link ID, every unidirectional link is associated with a set of d distinct LITs (Fig. 3). This allows us to construct different candidate zFilters and to select the best-performing one from the candidates, e.g., in terms of the false positive rate, compliance with network policies, or multi path selection.

The forwarding information is stored in the form of d forwarding tables, each containing the LIT entries of the active Link IDs, as depicted in Fig. 4. The only modification of the base forwarding method is that the node needs to be able to determine on which forwarding table it should perform the matching operations; for this, we include the index in the packet header.

Construction: When determining the actual forwarding tree based on the network graph, and the locations of the publisher and subscribers, we can apply various policy restrictions (e.g. link-avoidance) and keep traffic engineering in mind (e.g. balancing traffic load or avoiding temporarily congested parts of the network). As a result, we get a set of unidirectional links to be included into the zFilter. The final step is ORing together the corresponding LITs of the included links, yielding a candidate BF. As each link has d different identities, we get d candidate BFs that are "equivalent" representations of the delivery tree. That is, a packet using any of the candidates will follow, at minimum, all the network links inserted into the BF.

Selection: Recall that a false positive will result in an excess delivery; i.e., a packet will be forwarded over a link that is not part of the delivery tree. To achieve better performance in terms of lower false positive probability, we first consider two relatively simple strategies:

(i) Lowest false positive after hashing (fpa): The selected BF should be the one with the lowest false probability estimate after hashing: $min\{{\rho_0}^{k_0}, \ldots, {\rho_d}^{k_d}\}$, where ρ is the fill factor, i.e. the ratio of 1's to 0's.

(ii) Lowest observed false positive rate (fpr): Given a test set T_{set} of link IDs, the candidate BF can be chosen after counting for false positives against T_{set}. The objective is to minimize the observed false positives when querying against a known set of Link IDs active in the forwarding nodes along the delivery tree.

The *fpa* strategy is simple and aims at lower false positives rates for any set of link IDs under membership test. On the other hand, the *fpr* yields the best performance of false positives for a specific test set at the expense of higher computational complexity.

To further enhance *fpr*, false positives at different places can be weighted; i.e., we can consider some false positives less harmful than others. For example, we can avoid forwarding towards non-peered domains, resource constrained regions, or into potential loops. We call such selection criteria as *link avoidance*, since they are based in penalizing those candidate BFs that yield false positives when tested against certain links. For example, the following kinds of criteria could be considered:

(i) Routing policies: A T_{set} of links to be avoided due to routing policies.

(ii) Congestion mitigation: A *static* T_{set} of links avoided due to traffic engineering (e.g., low capacity links) and a *dynamic* T_{set} of congested links.

(iii) Security policies: A T_{set} of links avoided due to security concerns.

As a consequence, having multiple candidate representations for a given delivery tree is a way to minimise the number of false forwardings in the network, as well as restricting these events to places where their effects are smallest.

3.3 Stateful functionality

So far, we have considered stateless operations, where each forwarding node maintains only a static forwarding table storing the LITs. We now carefully introduce state to the network in the form of virtual links and fast failure recovery. While increasing hardware and signaling cost, the state reduces the overall cost due to increased traffic efficiency when facing large multicast groups or link failures.

3.3.1 Virtual links

In the case of dense trees, especially when a number of trees share multiple consecutive links, it becomes efficient to identify sets of individual links with a separate Link ID and associated LITs. We call such sets of links as *virtual links*. The abstraction introduces the notion of tunnels (or link aggregation) into our architecture – a notion more general than traditional one-to-one or one-to-many tunnels, being able to represent any link sets, including partial one-to-many trees, forests of partial trees, many-to-one concast trees, etc.

A virtual link may be generated by the topology layer whenever it sees the need for such a tree. The creation process consists of selecting the individual links over which the virtual link is created, assigning it a new Link ID, and computing the LITs. To finalize the creation process, the topology layer needs to communicate the Link ID, together with the LITs, to the nodes residing on the virtual link.

Note that virtual link maintenance does not need to happen in line speed; there are always alternative ways of sending the same data. For example, if a virtual link is needed to support a very large multicast tree, the sender can still send multiple packets instead of one, each covering only a part of the tree.

Once the virtual link creation process is finished, we can use a LIT of this virtual link in any zFilter instead of including all the individual LITs into it. This reduces the probability for false positives when matching the zFilter on the path. On the other hand, adding forwarding table entries into nodes increases the sizes of the forwarding tables. Given the typical Zipf-distribution of the number of multicast receivers [24], the sizes of the forwarding tables will still remain small compared to the current situation with IP routers. Unfortunately, falsely matching to a virtual link will mean falsely forwarding packets through the entire connected part of the denoted subgraph; however, this can be mitigated by careful naming of the virtual links (e.g. more 1-bits than in the case of physical links) and explicitly avoiding these false positives during BF-selection.

3.3.2 Fast recovery

Whenever a link or a node fails, all delivery trees flowing through the failed component break. In this section, we consider two approaches for fast re-routing around single link and node failures.

Our first approach is to replace a failed link with a functionally equivalent virtual link. We call this as *VLId-based recovery*. The idea is to have a separate virtual backup path pre-configured for each physical link ID, to be dynamically used in case of failure. This virtual backup path has the same Link ID and LITs as the physical link it replaces, but is initially inactive to avoid false forwarding.

The main advantage of this solution is that there is no need to change the packets. Basically, it is enough that the node detecting a failure sends an activation message over the replacement path, activating it for both the failed physical link and any virtual links flowing over the physical link, and then starts to forward the packets normally. When receiving the activation message, the nodes along the backup path reconfigure their forwarding tables, and as a result, the unmodified packets flow over the replacement path.

Another approach is to have a pre-computed zFilter encoding the replacement path. In this method, when a node detects a failure, it simply needs to add the appropriate LIT(s) representing the backup path into the zFilter in the packet. This method does not add any additional signaling or state to the forwarding nodes, but it increases the probability of false positives by increasing the fill factor of the zFilter.

Both of the mechanisms are capable of re-routing the traffic with zero convergence time and without service disruption. Besides protecting against single link failures, they are also able to recover from single node failures, if the operator has configured multiple backup paths or a backup tree towards all the neighbours of the failed node. These two types of failures cover around 85% of all unplanned outages [27]. In the complex cases where the proposed mechanisms are not able to perform local rerouting, new zFilters need to be computed.

3.3.3 Loop prevention

In some cases false positives can result in loops; for instance, consider the case where a zFilter encodes a forwarding path $A \rightarrow B \rightarrow C$, but, due to a false positive, the zFilter also matches with a separate link $C \rightarrow A$, which is used to forward packets from C to A. Without loop prevention, this will cause an endless loop of $A \rightarrow B \rightarrow C \rightarrow A$. Obviously, as the constructed delivery tree may cause a loop, we can still use the *fpr* method to select only loopless candidate BFs. However, this does not guarantee loop freeness as the network changes.

As an alternative solution, we start with each node knowing the neighboring nodes' *outgoing* Link ID and LITs towards the node itself; we call these the *incoming* Link ID and LITs. Now, for each incoming packet, the node checks the *incoming* LITs of its interfaces, except the one from where the packet arrives, and compares them to the zFilter. A match means that there is a possibility for a loop, and the node caches the packet's zFilter and the incoming Link ID for a short period of time. In case of a loop, the packet will return over a different link than the cached one. Our early evaluation is based on this approach and suggests that a small caching memory does not penalize the performance.

As a third alternative, at the inter-AS level we can divide the links into up, transit, and down ones, and utilise the valley-free traffic model. As a final method, it remains always possible to use TTL similar to what IP uses today.

3.3.4 Explicitly blocking false positives

Most false positives cause a packet to be sent to a node that will drop it. In some cases, the traffic generated as a result of a false positive should be fully truncated; e.g., in the case of low capacity or congested links, heavy non-cacheable traffic flows, or inter-domain link policies it may be necessary to locally disable forwarding of some traffic. Hence, we need a means to explicitly block the falsely forwarded traffic flows at an upstream point.

Therefore, any node can signal upstream a request to block a specific zFilter over that physical link. This can be implemented as a "negative" virtual Link ID, where a match blocks forwarding over the link instead of enabling it.

3.4 Control messages, slow path, and services

To inject packets to the slow path on forwarding nodes, each node can be equipped with a local, unique Link ID denoting the node-internal passway from the switching fabric to the control processor. That allows targeted control messages to be passed to one or a few specific nodes, if desired. Additionally, there may be virtual Link IDs attached to these node-local passways, making it possible to multicast control messages to a number of forwarding nodes without needing to explicitly name each of them. If the messages need to be modified, or even stopped on a node, the simultaneous forwarding should be blocked. This can be done by using zFilters constructed for node-to-node communication, or using a virtual Link ID especially configured to pass messages to the slow path and make the forwarding decision after the message has been processed.

Generalising, we make the observation that the egress points of a virtual link can be basically anything: nodes, processor cards within nodes, or even specific services. This would allow our approach to be extended to upper layers.

Another usage of control messages is collecting a symmetric reverse path from a subscriber to the publisher for the purpose of e.g. providing feedback. The publisher can initiate a control message triggering reverse path collection. Getting the message, each intermediate node bitwise ORs the appropriate reverse LIT with the path already collected and forwards it towards the subscriber. When the message finally reaches the subscriber, it will have a valid zFilter towards the publisher. The zFilter was created without interacting with the topology system.

4. EVALUATION

We now study some of the design trade-offs in detail. First, we introduce a few performance indicators, and then explore scalability limits and system performance. We use packet-level ns-3 simulations over realistic AS topologies, gaining insights on the forwarding efficiency of the proposed solution. Finally, we consider security aspects.

4.1 Performance indicators

A fundamental metric is the *false positive rate* of the in-packet Bloom filter. Link ID Tags are already in the form of m-bit vectors, with k bits set to one, as they are added to a candidate BF_i. An accurate estimate of the basic false positive rate can be given once the *fill factor* ρ of the BF is known. The *false positive after hashing* fpa is the expected false positive estimate *after* BF construction:

$$fpa = \rho^k \tag{1}$$

The *fpa-optimized* BF selection was introduced in Sec. 3.2 and is based on finding the set of LITs with the smallest predicted fpa. The *observed false positive probability* is the actual *false positive rate* (fpr) when a set of membership queries are made on the BF:

$$fpr = \frac{\#\text{Observed false positives}}{\#\text{Tested elements}} \tag{2}$$

Note that the fpr is an experimental quantity and not a theoretical estimate. The minimum observed fpr of the d candidate BFs provides a reference lower bound for a specific BF design.

These two metrics form the basic BF-selection criteria. While *fpa-optimized* selection is cheaper in computational terms, the *fpr-optimized* selection will give better results as the actual topology is more precisely considered in this process. However, the *fpr* describes the overall network performance only indirectly. In order to capture better the actual bandwidth consumption due to false positives, we introduce *forwarding efficiency* as a metric to quantify the bandwidth overhead caused by sending packets through unnecessary links:

$$fwe = \frac{\#\text{Links on shortest path tree}}{\#\text{Links during delivery}} \tag{3}$$

In other words, forwarding efficiency is 100% if the packets strictly follow the shortest path tree for reaching the subscribers. Consequently, this metric is representative and useful in the scenarios where larger subscriber sets are reached with multiple smaller delivery trees, or in virtual link scenarios, where false positives may be costly by causing deliveries over multiple hops.

AS	1221	3257	3967	6461	*TA2*
Nodes (#)	104	161	79	138	65
Links (#)	151	328	147	372	108
Diameter	8	10	10	8	8
Radius	4	5	6	4	5
Avg (Max) degr.	2 (18)	3 (29)	3 (12)	5 (20)	3 (10)

Table 1: Graph characterization of a subset of router-level AS topologies used in the experiments.

4.2 Packet level simulations

First, we used the intra-domain AS topologies from Rocketfuel [1] to simulate the protocol behaviour. Though not completely accurate, they are a common (best) practice to experiment with new forwarding schemes in real world scenarios[2]. A second useful data set is SNDlib [28], from where we selected the largest network (*TA2*). The most important properties of these networks are shown in Table 1.

Using ns-3, we implemented a zFilter-based forwarding layer and a simple topology module, which computes zFilters based on publisher and subscriber locations and the actual network map; the selected tree is always defined by the shortest paths between the publisher and each of the subscribers. We set m, the size of the BF to 248 bits; a fair comparison to the IPv6 source and destination fields $(2 \cdot 128)$. We briefly considered $m = 120$ and $m = 504$, but abandoned the former due to poor performance and the latter due to relatively small overall gains compared to the per-packet cost. A more flexible design, allowing m to vary per packet, is left for further study. We investigated the effect of different numbers of forwarding tables (d), the number of subscribers (n), and the different LIT-sets for the nodes (constant $k = 5$, variable $k \in [3, 3, 4, 4, 5, 5, 6, 6]$), as well as different BF-selection strategies.

Stateless forwarding: We present the essence of our simulation results on Tables 2 and 3. Table 2 contains results using the *fpa* selection criteria with the variable distribution

[2]Recent studies [35] have pointed out some limitations in Rocketfuel data, suggesting that the number of actual physical routing elements may be less than inferred by their measurement technique. However, this particular inaccuracy in the present data places more stress on our mechanism than the suggested corrected scheme would place.

Users	AS	Links (#)		Efic. (%)		fpr (%)	
		mean	95%	mean	5%	mean	95%
4	*TA2*	8.6	12.7	99.92	100	0.02	0
	1221	9.7	13.6	98.08	88.89	0.37	2.13
	3257	9.6	13.5	99.83	100	0.02	0
8	*TA2*	15.6	20.0	99.6	94.12	0.2	1.59
	1221	16.8	21.3	97.78	90.89	0.54	2.02
	3257	17.9	22.9	98.95	91.3	0.28	1.25
16	*TA2*	25.7	30.9	97.92	91.67	0.83	2.67
	1221	27.4	31.0	95.51	88.22	1.28	3.17
	3257	31.3	36.7	92.37	79.58	1.76	3.86
24	*TA2*	34.1	38.8	95.2	87.18	1.95	4.63
	1221	36.1	41.0	92.06	83.33	2.65	5.19
	3257	42.2	48.1	82.27	67.69	4.17	6.96
32	*TA2*	41.4	46.0	92.04	84.31	3.46	6.46
	1221	44.0	48.3	88.22	78.95	4.32	7.45
	3257	52.2	57.9	71.47	59.34	7.3	10.41

Table 2: ns-3 results for d=8, variable k-distr.

Users	AS	links	fpr_{fpa} (%)		fpr_{fpr} (%)		Stdrd
		mean	k_c	k_d	k_c	k_d	$k = 5$
8	*TA2*	15.6	0.12	0.2	0	0	0.18
	1221	16.83	0.44	0.54	0.26	0.26	0.55
	3967	17.72	0.28	0.33	0.03	0.03	0.48
	6461	17.18	0.32	0.39	0.06	0.07	0.36
16	*TA2*	25.7	0.54	0.83	0.01	0.03	0.8
	1221	27.37	1.17	1.28	0.36	0.45	1.57
	3967	29.04	1.13	1.29	0.24	0.34	1.48
	6461	29.31	1.55	1.57	0.71	0.83	1.89
24	*TA2*	34.1	1.65	1.95	0.38	0.58	2.03
	1221	36.14	2.48	2.65	1.21	1.33	3.55
	3967	37.65	2.55	2.78	1.31	1.48	3.22
	6461	39.60	3.72	3.79	2.81	2.86	4.86

Table 3: Mean fpr values for different configurations.

of k. The performance appears adequate in all of the topologies, up to 23 subscribers ($\approx$ 32 links); forwarding efficiency is still above 90% in the majority of the test cases. The result is much better than multiple unicast, where the same links would be used multiple times by the same publication. For example, in AS3257 the unicast forwarding efficiency is only 43% for 23 subscribers.

Table 3 sheds light on the difference between *fpa* and *fpr* algorithms. There is an interesting relation between the distribution of k and the optimization strategies: in our region of interest, $k_c = 5$ performs better than the variable k distribution (k_d). As expected, *fpr-optimization* successfully reduces the false positive rate, and outperforms the non-optimised ($d = 1$) approach by 2–3 times in the scenarios with 16 users. The gain of using *fpa* instead of the non-optimised algorithm is clear, although not as significant as with *fpr*. These improvements can be also observed in the sample results of AS6161, see Fig. 5.

Of course, as the link IDs are inserted into the zFilters, delivery trees are only present in the packet headers, and therefore completely independent from each other. Hence, the number of simultaneous active trees does not affect the forwarding performance.

Stateful forwarding: In networks with scale-free properties, a large part of the traffic flows between high-degree hubs. We experimented with the effects of installing virtual

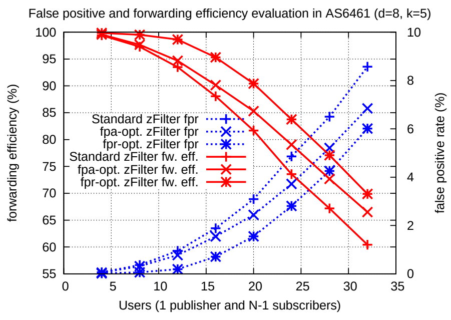


Figure 5: ns-3 simulation results for AS 6461.

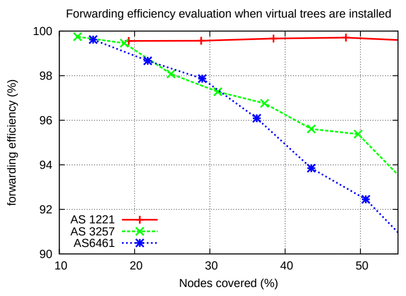


Figure 6: Stateful dense multicast efficiency

links covering different parts of the network. We built virtual links from the publisher towards the core and between the hubs, but that enhanced the performance only slightly, as virtual links substituted only a couple of physical links.

Significant performance enhancements can be reached if we install virtual links rooted at (high-degree) core nodes and covering a set of subscribers, avoiding thereby the presence of many LITs in the zFilter. The results on Fig. 6 show that dense multicast can be supported with more than 92%-95% forwarding efficiency even if we need to cover more than 50% of the total nodes in the network (cf. Table 2).

Forwarding table sizes: Assuming that each forwarding node maintains d distinct forwarding tables, with an entry consisting of a LIT and the associated output port, we can estimate the amount of memory we need for the forwarding tables:

$$FT_{mem} = d \cdot \#Links \cdot [size(LIT) + size(P_{out})] \quad (4)$$

Considering $d = 8$, 128 links (physical & virtual), 248-bit LITs and 8 bits for the outport, the total memory required would be $256Kbit$, which easily fits on-chip.

Although this memory size is already small , we can design a more efficient forwarding table by using a *sparse representation* to store just the positions of the bits set to 1. Thereby, the size of each LIT entry is reduced to $k \cdot log_2(LIT)$ and the total forwarding table requires only $\approx 48Kbit$ of memory, at the expense of the decoding logic.

4.3 Discussion

To support larger trees than we can comfortably address with a single zFilter, two choices can be considered. First, we can create virtual links to maintain the fill factor and to keep the overdeliveries under control. This comes at the price of control traffic and the increase of state in forwarding nodes. Second, we can send multiple packets: instead of building one large multicast tree we can build several smaller ones, thereby keeping zFilters' fill factor reasonable. The packets will follow the desired route with acceptable false delivery rates, but exact copies will pass through certain links where the delivery trees overlap. Depending on the scenario specifics, this can result in more bandwidth waste than in the case of a single larger tree.

So far, we have calculated the performance of zFilters for specific sized subscriber sets. A further step is to estimate the overall performance of the network, where the traffic matrix is consisting of a large variety of different subscriber sets. Here we rely on current systems centered around disseminating information objects. First, according to RSS workload data collected at Cornell, the number of subscribers for different topics follows a Zipf distribution [24]. Second, YouTube video popularity also shows a power-law distribution in a campus network [17]. Third, IPTV channel popularity [11] was measured to have the same characteristics even with a faster drop in the case of unpopular channels than the Zipf-distribution would suggest. Fourth, in typical data centers there is a need for a large number of multicast groups, albeit all contain only a small amount of receivers [5].

Based on these observations, assuming a long tail in the popularity of topics, with $m = 248$ our results confirm that our fabric needs no forwarding state for the large majority of topics and requires virtual links or multiple sending only for the few most popular topics. This is a clear advantage compared to IP multicast solutions, where even the small groups need forwarding states in the routers. Furthermore, as we can freely combine the stateful and stateless methods, we can readily accommodate a number of changes in the popular topics before needing to signal a state change in the network, avoiding unnecessary communication overhead.

4.4 Security

The probabilistic nature of Bloom filters directly provides the basis for most of our security features. Furthermore, as zFilters are location specific, it is unlikely that any given zFilter could induce any usable traffic if used outside of its intended links. Without knowledge of the actual network graph, including the active Link IDs and LITs, it is unpractical trying to guess a zFilter that would reach any particular set of nodes.

In a simple *zFilter contamination attack*, the attacker tries to get a single packet to be broadcasted to all possible links by using a BF containing a large amount of 1's (or even only 1's). A simple countermeasure for such attack, also observed in [44], is to limit the fill factor, e.g., to 50–70%. We have implemented this in hardware, without causing any additional delay. As a result, a randomly generated zFilter will match outgoing links only at the false positive rate resulting from the maximal allowed fill factor.

In a more advanced attack, combining a *LIT learning attack* and a *zFilter re-use attack*, an attacker may first attempt to figure out the LITs of the links nearby it by attempting to lure lots of subscribers from different parts of the network. The attacker learns a number of valid zFilters originating at it and, using AND for the received LITs, guesses the LITs of the next few links. This attack, however, requires a lot of work, and there are a few direct countermeasures. First, the number of parallel LIT's close to the publisher can be increased and the uplink Link IDs can be changed more often. Second, by varying the selection of the Bloom filter (Sec. 3.2), though not optimal, we may increase the probability that the attacker gets a too full zFilter.

More generally, we can avoid many of the known, and probably a number of still unknown attacks, by slowly changing the Link IDs over time. Our on-going work is focusing on hash chains and pseudo-random sequences in this area, meaning that with a shared secret between the individual forwarding nodes and the topology system the control overhead of communicating the changes could be kept at a minimum. The caveat would be that the zFilters being used in the network need to be re-calculated once in a while.

Overall, no forwarding state is created if there aren't a fairly large number of subscribers that have explicitly indicated their interest in data delivery. We thereby avoid the typical problems of multicast routers maintaining state of unnecessary multicast groups, e.g., an attacker joining many low-rate multicast groups.

Finally, consider a situation where an attacker has successfully launched a *DDoS attack*. Initially, the victim can quench the packet stream by requesting the closest upstream node to filter traffic according to the operation defined in Section 3.3.4. After that, the LITs on the forwarding nodes can be changed to extinguish the attack. However, the latter is a slower operation, requiring updates to the topology layer and recalculation of zFilters for affected active subscriptions. Additional future work will consider how legitimate traffic can exploit the multi-path capabilities of the zFilters.

5. FEASIBILITY

We now turn our attention to the overall feasibility of our approach, focusing on the inter-networking aspects. In particular, we consider how our forwarding fabric can be extended to cover inter-domain forwarding. We discuss the efficiency and scalability aspects for the pure pub/sub case. For the IP-based multicast case, described in Section 2.4, we need to use currently existing mechanisms, limiting the breadth of the issues. We also discuss how the proposal is (slightly) better than IP in supporting data-oriented naming and in-network caching.

5.1 Full connectivity abstraction

As mentioned in Sect. 2, the overall architecture we rely on is based on a recursive approach, where each layer provides a full connectivity abstraction. Hence, to implement inter-domain forwarding, we need to attach two forwarding headers into a packet, an intra-domain and an inter-domain one, and replace the intra-domain header at each domain boundary. For IP multicast, the IP header with the IP multicast address takes the place of the inter-domain header.

To provide the full mesh abstraction, a domain provides an *inter-domain Link ID (IdLId)* for each of its neighboring domains. Furthermore, the domain provides a distinct Link ID to be added to packets that have local receivers. Hence, in the inter-domain zFilter of an incoming packet, there is the incoming *IdLId* for the link from the previous to this domain, the outgoing *IdLId*s for the links from this domain to any next domains, and if there are any local receivers, the *IdLId* denoting their existence.

When we receive a packet from outside, we first may verify that the packet is forwarded appropriately, e.g., that the inter-domain zFilter contains the incoming *IdLId*. Af-

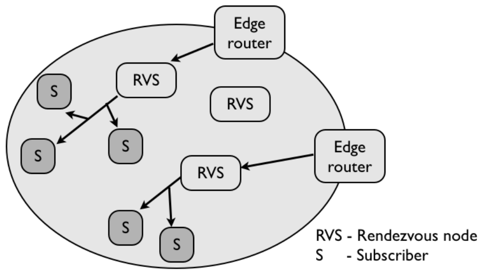


Figure 7: Inter-domain forwarding with distributed RVSs

ter that, we match the zFilter against all outgoing *IdLIds*, simultaneously looking up the corresponding intra-domain zFilters. The intra-domain zFilters can usually be simply merged. If the inter-domain zFilter indicates local recipients, more processing is needed.

We assume that the data topic identifier, carried inside the packet, or other suitable identifier such as an IP multicast address, is used to index the set of local recipients. For IP multicast addresses, it is reasonable to expect the edge nodes to maintain the required state. For the pub/sub case, where the number of active topics may be huge, the subcriber information may be divided between a set of intra-domain rendezvous nodes (see Figure 7), providing load distribution.

Eventually, a rendezvous node looks up the intra-domain zFilter by using the topic identifier. As it takes time to pass the packet to the right rendezvous node, and as the lookup may take some time, the rendezvous nodes can construct cache-like forwarding maps and distribute them to the edge nodes.

5.2 Resource consumption

We now estimate the amount of resources needed to maintain topic-based forwarding tables, needed for the recursive layering in the pub/sub case. To estimate the storage requirements, we consider the number of indexable web pages in the current Internet as a reasonable upper limit for the number of topics subscribed within a domain. In 2005 there was around 10^{10} indexable web pages [18]; today's number is larger and we assume it to be around 10^{11}. Considering that each topic name would take 40 bytes and each forwarding header takes $32 - 34$ bytes, in the order $\approx 10\ TB$ of storage would be needed.

Following the argumentation by Koponen et al [23] and assuming similar dynamics, it is plausible that even a single large multi-processor machine could handle the load. However, a multi-level lookup caching system is needed to reduce per-packet lookup delay to a reasonable level. For example, each edge node could cache a few million most active topics, each rendezvous node could keep in their DRAM a few billion less active topics, and the information about rest could be stored on a fast disk array. If only a small fraction of subscriptions would be active at any given point of time, the suggested multi-level caching may make it possible to handle the typical lookup load with just one or a few large server PCs.

We note that the approach may be problematic for applications where the inter-packet delay is long but latency requirements are strict. If needed, the problem can be solved by introducing explicit signaling that would allow certain topics to be always kept in the cache, even when not actively used[3].

An interesting open problem is to consider potential space saving techniques, such as determining commonalities between inter-domain zFilters, perhaps allowing them to be used as indices. If the topics sharing a single inter-domain zFilter can be distinguished with only a few bits, it may be possible to develop clever data structures for compressing the topic-based forwarding tables.

5.3 Policy compliance and traffic engineering

For the IP case, we expect no real changes to traffic engineering or policies, as the forwarding fabric would be invisible outside of the domain. In the recursive pub/sub case, we have to make sure that the inter-domain zFilters are policy-compliant. As a starting point, each edge node can verify that all traffic is either received from a paying customer or passed to a paying customer. However, due to multicast, there are difficult cases not covered by the typical IP-based policy compliance rules, such as traffic arriving from one upstream provider and destined both to a paying customer and another upstream provider. In general, we will eventually need a careful study of the issues identified by Faratin et al [16]. As observed in [30], it is an open problem how the kind of source routing we propose may change the overall market place and policies.

Considering traffic engineering, sender-based control would be easy. At this point, however, open questions include how the transit operators may affect the paths or how the receivers can express their preferences. We surmise that those aspects have to be implemented elsewhere in the architecture, as our forwarding layer can redirect traffic only by redirecting links.

5.4 Naming and caching

As mentioned earlier, both data-oriented naming and in-network caching are needed for efficient pub/sub. Our stack structure and independence of end-node addresses in zFilter forwarding, make both of these functions simpler compared to IP networks. Our architecture treats data as first class citizens. The focus is on efficient data delivery instead of connecting different hosts for resource sharing. The default choice of multicast brings natural separation of rendezvous (addressing/naming) and routing. The resulting identifier/locator split gives better support for data-oriented naming than the current IP-based architecture, cf. e.g. [2]. Once routing is based on location-independent identifiers, any kind of native naming and addressing on the infrastructure turns out to be a straightforward task.

The zFilter forwarding eases in-network caching by supporting the required decoupling between publishers and subscribers. Publishers can publish data in the network, independent of the availability of subscribers. Packet caching and further delivery from the caches is relatively simple, as node based addressing is not needed. Caching can also be used for other purposes, e.g., enhancing reliability. Combining data-oriented naming and caching, we can turn the traditional packet queues and the sibling recipient memories into opportunistic indexable caches, allowing, for example,

[3]Obviously, such a service would either need strict access controls or an explicit fee structure.

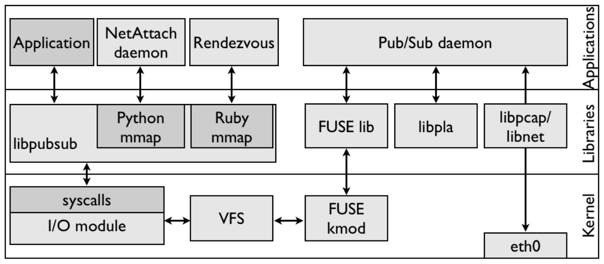


Figure 8: FreeBSD prototype structure

any node to ask for recent copies of any missed or garbled packets; cf. [3, 14]. The fine-grain path control allows us to easily determine those nodes that may have copies of recent packets in their memory. Multicast, in turn, allows local control queries to be sent efficiently. The falling prices of memory compared to bandwidth indicates the economical feasibility of our model.

6. IMPLEMENTATION

There are currently two partial prototypes of the system. The FreeBSD-based end-node prototype consists of some 10000 lines of C code, implementing both the pub/sub subsystem and the forwarding fabric. Our NetFPGA-based forwarding node prototype has currently some 390 lines of Verilog, implementing the main ideas from this paper. In this section, we briefly describe the implementation details and present early measurements of the NetFPGA forwarding module.

6.1 End node

The structure of the end-node prototype is depicted in Fig. 8. The I/O module implements a few new system calls for creating new publications (reserving memory areas), publishing, and subscribing. When allocating memory for a publication, the pager is set to be a vnode pager, and the backing file to be in the Filesystem in Userspace (FUSE)[38]. Hence, each publication is backed up by a virtual file, located in a separate virtual file system running under FUSE.

Currently, forwarding and other network traffic is handled in separate threads running within the Pub/Sub daemon, simply sending and receiving raw Ethernet frames with `libnet` and `libpcap`. While we use Ethernet frames, we always broadcast the frames, basically using each Ethernet cable as a point-to-point link, disregarding any Ethernet bridging or switching.

6.2 Forwarding node

We have implemented an early prototype of a forwarding node using Stanford NetFPGA [25]. Starting from the Stanford reference switch implementation, we removed most of the for-us-unnecessary code in the reference pipeline and replaced it with a simple zFilter switch. At this point, we have implemented the basic LIT and virtual link ideas, and tested it with 4 real and 4 virtual LITs per interface. With this configuration, the total usage of NetFPGA resources for the logic is 4.891 4-input LUTs out of 47.232, and 1.861 Slice Flip/Flops (FF) out of 47.232. No BRAMs are reserved. For the whole system, the corresponding numbers are 20.273 LUTs, 15.347 FFs, and 106 BRAMs.

# of NetFPGAs	Average latency	Std. Dev.	Latency/ NetFPGA
0	$16\mu s$	$1\mu s$	N/A
1	$19\mu s$	$2\mu s$	$3\mu s$
2	$21\mu s$	$2\mu s$	$3\mu s$
3	$24\mu s$	$2\mu s$	$3\mu s$

Table 4: Simple latency measurement results

To get some understanding of the potential speed, we have made some early measurements. The first set of measurements, shown in Table 4, focused on the latency of the forwarding node with a very low load. In each case, the latency of 10000 packets was measured, varying the number of NetFPGAs on the path from zero (direct wire) to three. Packets were sent at a rate of 25 packets/second; both sending and receiving was implemented directly in the FreeBSD kernel.

The delay caused by the Bloom filter matching code is $56ns$ (7 clock cycles), which is insignificant compared to the measured $3\mu s$ delay of the whole NetFPGA processing. With background traffic, the average latency per NetFPGA increased to $5\mu s$.

To get an idea of the achievable throughput, we compared our implementation with the Stanford reference router. This was quantified by comparing ICMP echo requests processing times through a plain wire, our implementation, and the reference IP router with five entries in the forwarding table. To compensate the quite high deviation, caused by sending and receiving ICMP packets and involving user level processing, we averaged over 100 000 samples. The results are shown in Table 5.

While we did not directly measure the bandwidth (due to lack of test equipment to reliably fill the pipes), there are no reasons why the implementation would not operate at full bandwidth. The code is straightforward and should be able to keep the pipeline full under all conditions.

7. RELATED WORK

Related work falls into various categories, which we briefly discuss in the following paragraphs.

Network level multicast: Our basic communication scheme is functionally similar to IP-based source specific multicast (SSM) [19], with IP multicast groups replaced by topic identifiers. The main difference is that we support stateless multicast for sparse subscriber groups, with unicast being a special case of multicast; IP multicast typically creates lot of state in the network if one needs to support a large set of small multicast groups.

In "Revisiting IP multicast" [31], Ratnasamy et al propose source border routers to include an 800-bit Bloom-filter-based shim header (`TREE_BF`) in packets. TREE_BFs represent AS-level paths of the form $AS_a : AS_b$ in the dissemination tree of multicast packets. Moreover, a second type of Bloom filters is used to aggregate active intra-domain multicast groups piggybacked in BGP updates. The presented method uses standard IP-based forwarding mechanisms enriched with the built-in TREE_BF to take the inter-domain forwarding decisions. However, our multicast fabric uses the in-packet Bloom filter directly for the forwarding decisions, removing the need for IP-addresses and proposing Link IDs

Path	Avg. latency	Std. Dev.
Plain wire	$94\mu s$	$28\mu s$
IP router	$102\mu s$	$44\mu s$
LIPSIN	$96\mu s$	$28\mu s$

Table 5: Ping through various implementations

as a generic indirection primitive.

In Xcast [7], source nodes encode the list of multicast channel destinations into the Xcast header. Each router along the way parses the header, partitions the destinations based on each destination's next hop, and forwards a packet appropriately until there is only one destination left where the Xcast packet is unicasted. Our fixed size Bloom filter approach shares the simplicity and stateless operations of Xcast while it avoids costly header re-writings and the destination IP packet header overhead.

Data-center applications and multicast: In [4] Bhargava et al discuss the performance achieved with kernel-level multicast for distributed databases. Due to the problems of IP multicast, such approaches are not commonly used in data-center applications. Recently, there has been some efforts on mapping traditional IP multicast to new models to ease wider use of IP multicast in these applications [42]. Our approach provides some new ground for considering multicast and explicit routing (e.g., middlebox serialization) in data-center environments.

Explicit routing: The simplest form of source routing [37] is based on concatenating the forwarding nodes' network identifiers on the path between senders and receivers. Our approach addresses the main caveats of source routing, including the overhead of having to carry all the routing information in the packet. Moreover, our approach does not reveal node or link identifiers, not even to the sending nodes, nor the sequence or exact amount of hops involved.

GMPLS [26] is being marketed as a solution to provide fast forwarding. By separating control and forwarding planes, it introduces more flexibility and promises performance gains, with the hardware-based fast label switching. However, it does not directly scale for massive multicast due to the limited label space and no capability for label aggregation.

In PoMo [29], Poutievski, Calvert, and Griffioen suggest an approach that trades overdeliveries for reduced state and reduced dependence of node network locators. In [10], the same authors propose an architectural approach with link identities having a pivotal role.

The BANANAS framework [22] is based on encoding each path as a short hash (PathID) of a sequence of globally known identifiers. The focus of BANANAS is on host-centric multipath communications, while ours is centered around non-global, opaque Link IDs and their compact representation. Some of the schemes developed in [22] for route computation and deployability over existing connectionless routing protocols (e.g., OSPF and BGP extensions) may be used to support LIPSIN over legacy networks.

Routing and forwarding with Bloom filters: Multiple flavours of Bloom filters [9] have been proposed to assist the forwarding operations of diverse systems (e.g., P2P, WSN, pub/sub). In the field of content-based pub/sub [21], Bloom filters are employed to represent a conjunction of subscriptions' predicates (`SBSTree`) used at content-based event forwarding time. In comparison, our pub/sub primitives are topic-based and the Bloom filters are built into packets to carry link IDs and not summarized subscriptions stored in network elements. Other forms of in-packet Bloom filters include the loop detection mechanism in Icarus [43], the credentials-based data path authentication in [44], and the aforementioned AS-level path representation for IP multicast [31].

8. CONCLUSIONS

Building on the idea of placing a Bloom filter into data packets, we have proposed a new forwarding fabric for multicast traffic. With reasonably small headers, comparable to those of IPv6, we can handle the large majority of Zipf-distributed multicast groups, up to some 20 subscribers, in realistic metropolitan-sized topologies, without adding any state in the network and with negligible forwarding overhead. For the remainder of traffic, the approach provides the ability to balance between stateless multiple sending and stateful approaches. With the stateful approach, we can handle dense multicast groups with very good forwarding efficiency. The forwarding decisions are simple, energy efficient, parallelised in hardware, and have appealing security properties. All these attributes make our work, in its current form, a potential choicer for data-center applications.

While a lot of work remains, the results indicate that it may be feasible to support Internet-wide massive multicast in a scalable manner. Technically, the main remaining obstacles are related to determining the right local delivery tree for traffic arriving from outside of a domain. Our current proposal scales only linearly. The problems related to the deployment and business aspects are likely to be even harder, but fall beyond the scope of this paper.

From a larger point of view, support for massive multicast is but one component needed for Internet-wide publish/subscribe. The other two components, data-oriented naming and in-network caching, we touched only indirectly. However, we hope that our work allows others to build upon it, allowing experimentation with network architectures that are fundamentally different from the currently deployed ones.

9. ACKNOWLEDGEMENTS

This research was supported by the EU's PSIRP project (FP7-INFSO-IST 216173). The authors thank the SIGCOMM reviewers and our shepherd Jon Crowcroft for the comments that helped to improve the paper. We also thank NomadicLab's implementation team for their efforts.

10. REFERENCES

[1] Rocketfuel ISP topology data. http://www.cs.washington.edu/research/networking/rocketfuel/maps/weights-dist.tar.gz.

[2] B. Ahlgren, L. Eggert, A. Feldmann, A. Gurtov, and T. R. Henderson. Naming and addressing for next-generation internetworks. Technical report, Dagstuhl, 2007.

[3] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral Error Correction for Time-Critical Multicast. In *NSDI' 07*, 2007.

[4] B. Bhargava, E. Mafla, and J. Riedl. Communication in the Raid distributed database system. *Comput. Netw. ISDN Syst.*, 1991.

[5] K. Birman, M. Balakrishnan, D. Dolev, T. Marian, K. Ostrowski, and A. Phanishayee. Scalable Multicast Platforms for a New Generation of Robust Distributed Applications. In *COMSWARE' 07*, 2007.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[7] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms. Explicit multicast (Xcast) concepts and options. IETF RFC 5058, 2007.

[8] R. Briscoe. The implications of pervasive computing on network design. *BT Technology Journal*, 22(3):170–190, 2004.

[9] A. Z. Broder and M. Mitzenmacher. Survey: Network applications of Bloom filters: A survey. *Internet Mathematics*, 2004.

[10] K. L. Calvert, J. Griffioen, and L. Poutievski. Separating Routing and Forwarding: A Clean-Slate Network Layer Design. In *In proc. of the Broadnets Conf.*, 2007.

[11] M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft. On next-generation telco-managed P2P TV architectures. In *IPTPS '08*, 2008.

[12] J. Day. *Patterns in Network Architecture: A Return to Fundamentals.* Prentice Hall, 2008.

[13] S. E. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. on Comp. Syst.*, 1990.

[14] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen. Ditto - A System for Opportunistic Caching in Multi-hop Wireless Mesh Networks. In *ACM Mobicom*, 2008.

[15] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 2003.

[16] P. Faratin, D. Clark, P. Gilmore, S. Bauer, A. Berger, and W. Lehr. Complexity of Internet interconnections: Technology, incentives and implications for policy. In *TPRC' 07*, 2007.

[17] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *ACM SIGCOMM IMC'07.*, 2007.

[18] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *WWW '05*, 2005.

[19] H. Holbrook and B. Cain. Source-specific multicast for IP. RFC 4607. 2006.

[20] J.D.Touch and V.K.Pingali. The RNA metaprotocol. In *ICCCN '08*, 2008.

[21] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS '08*, 2008.

[22] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. Bananas: an evolutionary framework for explicit and multipath routing in the internet. *SIGCOMM Comput. Commun. Rev.*, 2003.

[23] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*, 2007.

[24] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In *IMC'05*, 2005.

[25] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA–an open platform for gigabit-rate network switching and routing. In *MSE '07*, 2007.

[26] E. Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945, 2004.

[27] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *INFOCOM 2004*, 2004.

[28] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *INOC' 07*, 2007.

[29] L. B. Poutievski, K. L. Calvert, and J. N. Griffioen. Routing and forwarding with flexible addressing. *Journal Of Communication and Networks*, 2007.

[30] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma. Incentive-compatible caching and peering in data-oriented networks. In *ReArch'08*, 2008.

[31] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *SIGCOMM'06*, 2006.

[32] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, 2008.

[33] J. Scott, J. Crowcroft, P. Hui, and C. Diot. Haggle: a networking architecture designed around mobile users. In *Annual IFIP Conference on Wireless On-demand Network Systems and Services*, 2006.

[34] A. Sharma, A. Bestavros, and I. Matta. dPAM: a distributed prefetching protocol for scalable asynchronous multicast in P2P systems. In *INFOCOM' 05*, 2005.

[35] R. Sherwood, A. Bender, and N. Spring. Discarte: a disjunctive Internet cartographer. *SIGCOMM Comput. Commun. Rev.*, 2008.

[36] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM'02*, 2002.

[37] C. A. Sunshine. Source routing in computer networks. *SIGCOMM Comput. Commun. Rev.*, 1977.

[38] M. Szeredi. Filesystem in Userspace. *Located at http://fuse. sourceforge. net.*

[39] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *MobiArch '08*, 2008.

[40] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, and T. Bressoud. Opportunistic use of content addressable storage for distributed file systems. In *USENIX' 03*, 2003.

[41] D. Trossen (edit.). Architecture definition, component descriptions, and requirements. Deliverable D2.3, PSIRP project, 2009.

[42] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock. Dr. multicast: Rx for datacenter communication scalability. In *HotNets-VII*, 2008.

[43] A. Whitaker and D. Wetherall. Forwarding without loops in Icarus. In *Proc. of OPENARCH*, 2002.

[44] T. Wolf. A credential-based data path architecture for assurable global networking. In *IEEE MILCOM*, 2007.

[45] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve. Exploring the pubsub routing/forwarding space. In *International Workshop on the Network of the Future*, 2009.

Appendix C

Publication C

A. Zahemszky, A. Császár, P. Nikander and C. Esteve Rothenberg. "Exploring the Pub/Sub Routing & Forwarding Space." In *IEEE ICC, Workshop on the Network of The Future*, Jun. 2009, Dresden, Germany.

Exploring the Pub/Sub Routing&Forwarding Space

András Zahemszky, András Császár, Pekka Nikander
Ericsson Research
Email: {firstname.lastname}@ericsson.com

Christian Esteve Rothenberg
University of Campinas (UNICAMP)
Email: chesteve@dca.fee.unicamp.br

***Abstract*—We envision an information-centric future Internet where the network is built around named pieces of data instead of explicitly addressable hosts. One clear way of implementing information-centric networking is using publish and subscribe (pub/sub) operations instead of the send and receive primitives. Internet-like pub/sub networking requires completely different routing protocols and forwarding mechanisms compared to those that are extensively used today. Consequently, we are facing a clean-slate design exercise, where we should start our adventure by exploring the new design space. We identify four key metrics (signalling overhead, state in nodes, information in packets and routing stretch) to help us evaluating the different proposals. We present a general five-step approach for routing in pub/sub networks. The presented approach is recursive, so it can be repeated as many times as necessary until we reach manageable sized problem instances. The final part of the mechanism is to glue together the created and assigned forwarding structures to the publication to ensure that all interested subscribers at any domains in the network will get the requested data.**

I. Introduction

In addition to the prominent World Wide Web, new forms of Internet usage have appeared rapidly in the last years. People read news feeds, listen to webradios, watch (RT or non-RT) video streaming, and download huge amount of data using BitTorrent. A common characteristic of these applications is that the users indicate their wish to get some specific pieces of information and they are not interested in the sources of data as long as the data comes in its original, unmodified form.

The first one to prominently advertise this shift in the application space from connections to information was Van Jacobson in his talk in 2006 [1]. Since that, it has become a common direction to propose clean-slate solutions for solving the problems of IP [2]. Researchers are aiming for new inter-networking solutions that have native support for mobility, multi-homing, privacy, accountability, or other clearly missing features. Furthermore, many of the proposed solutions use data instead of the hosts as the basis for the design [3].

As a specific example, the EU FP7 PSIRP project [4] has a goal of building a pub/sub-based network, where the architecture uses pieces of data as the first-class citizens. With the *publish* operation, an endpoint can indicate that it wants to associate a document or a one-way channel with the given (possibly randomly looking) identifier. With the *subscribe*, an endpoint can signal its desire to get (read only) access to the named document or channel. Based on the subscriptions, the network is responsible for delivering the document or any data appearing on the channel, to all the subscribers. As typical to pub/sub-, multicast is the natural mode of communication.

In this paper, we explore the new design space we are facing when attempting to design the routing and forwarding components of the PSIRP architecture [4]. First, we discuss the problem of designing a scalable routing mechanism for multicast and multicast-based publish/subscribe (Sec. II). Next, we briefly overview the RTFM architecture [5], the variant of the PSIRP architecture that our work is based on, including various possible forwarding components (Sec. III). The core contribution of this paper is our Divide&Conquer approach to tackle the problems of scalable routing (Sec. IV). We divide the routing problem in two dimensions: first, vertical and second, horizontal. The former is hierarchical division and the latter is dividing the overall problem of multicast routing in each area of the hierarchy to smaller subproblems, each of which is easily manageable as such. Finally, we discuss related work (Sec. V) and provide our tentative conclusions (Sec. VI).

II. The problem

Our aim is to create an information-oriented network based on the pub/sub paradigm. The data transmission with unicast in such networks is clearly not optimal w.r.t. transport efficiency, while the same data is delivered over the same connections multiple times. Thus, we have selected multicast as the primary transport mechanism. As a consequence, it is more natural to consider delivery trees rather than forwarding paths.

A forwarding tree for a given publication contains the publisher and all subscribers. It may be optimal w.r.t. an appropriate metric, e.g. delay. We call such trees as *ideal* trees.

It appears that the routing&forwarding solution space for multicast tree implementations is 4-dimensional, as they can be evaluated by four rather orthogonal metrics. The first is the transport efficiency measured in *stretch*, meaning that the packets follow the shortest possible paths (in any appropriate metric). The second concerns the amount of *network state* needed in the forwarding nodes. This must grow sublinear w.r.t. the number of nodes and subscribers and strongly sublinear w.r.t. the number of publications. The third aspect involves the number of bits included in the *packet headers*, encoding the information that helps the forwarding node to determine the outgoing interface. The final one is the *signalling overhead* caused by routing and other related control protocols.

Assume two examples: a) *source routing* requires a high amount of information in the packet headers but relatively little state in the forwarding nodes. The downside is the signaling overhead in scenarios under subscribers' churn, as the sender of the data should be always up-to-date on the information encoded in the headers. Moreover, the transport utilization depends to a large extent on the algorithms the overall system uses to compute the forwarding trees for data delivery.

In the second example, b) *network state* installed to describe directly the ideal trees into the forwarding nodes provides high levels of transport efficiency and minimal number of bits in packet headers (just a tree identifier). Potentially, each forwarding node may need to have a separate entry for each single active publication tree in the globe. Moreover, every change in any tree would need to be signalled to a potentially large number of forwarding nodes.

To avoid potential state explosion (as our estimation total number of possible publications $\approx 10^{15}$), we are looking for solutions that use *good enough* forwarding trees, i.e., which are close to ideal but require only a reasonable amount of state in the network. When trying to save state, we may end up with a little higher overhead, for example in terms of unnecessarily sent packets. However, this may be remedied by other means, for instance by utilising *opportunistic caching*.

In this paper, our goal is to work towards a solution that finds a good balance among the contradictory aspects. To get there, we first present the overall PSIRP architecture, and then briefly discuss our ongoing work in the forwarding domain.

III. The PSIRP components

We start by briefly outlining the RTFM architecture [5], an early design choice from the PSIRP project. Our present contribution can be regarded as an evolutionary step building upon RTFM. The RTFM consists of recursive functional building blocks:

1) A *Rendezvous* system is in charge of matching subscriptions to publications within information scopes.
2) The *Topology* system, our main focus in this paper, is responsible for collecting and managing network topology information, as well as creating and maintaining the required multicast delivery trees both proactively (establishment, optimization) and re-actively (on-demand).
3) The *Forwarding* functions perform the actual datagram delivery, based on the forwarding identifiers in the packet headers and the state installed in the forwarding nodes by the topology system.
4) Finally, *More* refers to the additional data mediation functions at forwarding time, such as opportunistic caching or network coding.

At the bottom of the architecture we have the forwarding layer. Above it, the structure can be divided into a data and control plane (see Fig. 1). The functions are present seperately in different network levels (e.g. in domain(AS)-level, core network level). The *control plane* consists of the topology and rendezvous systems. At the *data plane*, in addition to the traditional transport functions, we envision a number of new network functions, e.g. opportunistic caching and lateral error correction. The transport functions will work in concert, utilising each other in a *component wheel* [4], similar to the way Haggle managers are organised [6].

In our design, the basic communication scheme is functionally similar to IP-based Source Specific Multicast (SSM) [7], with the IP multicast groups having been replaced by an identifier identifying the publication, similar to the topic identifier described in [8].

Each publication, be it a single packet, a one-way multicast channel or a document, is identified with a Rendezvous Identifier (RId). More precisely, each publication is identified with a $< SId, RId >$ pair, where the Scope Identifiers $< SId >$ are just specific RIds, identifying *scopes*, which help the rendezvous system to scale and to organise the publications.

When a publisher wants to publish a new piece of information, it picks up a RId and hands the publication data to the system. Correspondingly, subscribers acquire the RIds through application-specific means and ask the system to arrange the data to be received. Once the rendezvous system has identified a publication that has both a publisher (or an up-to-date cache) and one or more subscribers, the topology system is requested to build a forwarding tree from the present location of the data to the subscribers. The high-level operations of this routing function are the main concern of this paper.

A. Forwarding components

First, we outline four solution components: using in-packet Bloom filters to encode delivery trees, using Merkle trees to represent partial forwarding trees, and scaling these approaches up to more dense trees through installing explicit state within the forwarding nodes, and finally, we briefly outline how the latter approaches could work together.

1) Encoding delivery trees with Bloom filters: Bloom filters (BFs) [9] are data structures for representing subsets of a given maximum size without listing the individual elements. BFs are applicable in any situation where a small number of false positives is acceptable. When used to take forwarding decisions, false positives are translated into packets being transmitted over additional links than the ones originally programmed. As long as the false positive rate is low enough, we consider that acceptable due to active caching and the decreasing probability of concatenated false positives over multiple hops; for the details, see [10].

To encode delivery trees, we form a set L of directed links. That is, for the forwarding nodes A and B, we represent the

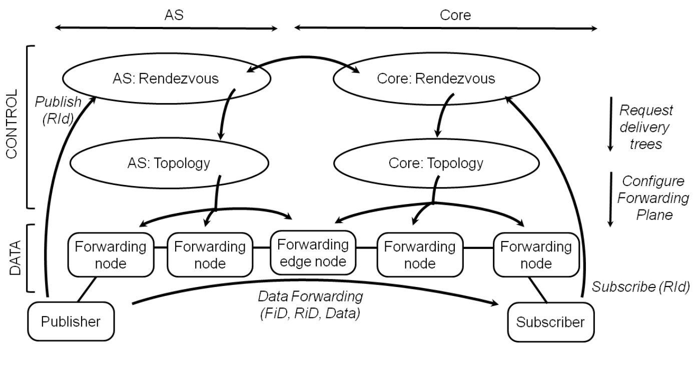


Fig. 1: High level arquitectural overview.

link from A to B as $\overrightarrow{AB}$, and the link from B to A as $\overleftarrow{AB}$. So, any forwarding tree can be seen as a set of unidirectional links. For example, encoding a simple tree with links from A to B, and from B to C, we should add $\overrightarrow{AB}$ and $\overrightarrow{AC}$ into the BF.

We place the BF describing the delivery tree into the packet header. Checking the header, each forwarding node tests which of its outgoing links are included in the set. Since this is a simple binary `and` operation, the checks can be done parallel in hardware, producing a very fast forwarding plane.[1]

As we show in [10], using 248-bit BFs, it is possible to encode up to about 40 link names (out of trillions) into a single BF, while still having an acceptable false positive rate (of around 4%.) That means that we can forward unicast packets over 40 links, well over the maximum practical hop count in the current Internet. For multicast, if we assume Internet-like AS topologies, we can use a single BF to address up to 20 receivers scattered all over a large AS topology. If more is needed, state can be added into the network in the form of virtual links as shown below.

Hence, using BFs in packets seems to provide us with an efficient way of source routing in arbitrary trees over Internet-like topologies.Our approach aims to work in intra- and inter-domain scenarios over any transport technology.

2) Merkle tree representation: Merkle trees [11] are data structures that contain summary information about a larger tree-like data structure. We propose to use them as an alternative compact form to represent forwarding trees and to derive chained forwarding labels.

Every node in the Merkle tree has a view of its vicinity and knows about the active trees it participates in. The active tree structures need to be consistent among the domains participating in the inter-domain routing, requiring the existence of a routing protocol to maintain the tree structures. The actual labels forming the hash trees can be constructed for each active tree. The node compares the received label with the compiled root hashes of its active network trees to resolve the next hop(s) and include the updated forwarding identifiers. Hence, the chained hashing mechanism of Merkle trees provide an uniform way to derive compact labels that aggregate paths recursively.

3) Utilising subtrees as virtual links: When the capacity of a single BF is not sufficient, the system can be extended by representing some subtrees as a *virtual link*. Basically, a virtual tree is a subtree that has a distinct name and associated state in the network nodes. That is, while we use BFs to encode the links of a tree in the packet header, we can also include trees as virtual links in this Bloom-filter-based representation.

Given the scale-free characteristics of the Internet, a major part of the data travels through a very small hub (tier-1). With virtual links, we can define forwarding subtrees that take advantage of the common path towards the center of the network that can be assumed for the vast majority of the traffic from any source. By defining virtual links spanning over multiple hops towards the domain edges, more stable fast forwarding paths can be installed explicitly in the network.

Hence, with this construction, the problem of mapping rendezvous identifiers to forwarding identifiers can be reduced, deeper in the network, to a smaller mapping problem of active flows (partly defined by virtual links), which is well within the scope of feasibility.[2]

4) Approximate fast stateful edge switching: False-positive-prone fast forwarding decisions can also be performed through stateful operations between domain boundaries. At the edges, making a switching or mapping decision between the large rendezvous identifier space and the smaller forwarding identifier space needs to be efficient both in space (small amounts of high-speed memory) and time (few computation cycles per packet).

The *SPSwitch* in [2] is a promising approach to solve the problem: with only a few bits per entry (40-50 bits), switching of packets identified by long flat identifiers (e.g., 256-bit) can be performed in a fast and efficient way. Again, the price is a fairly small amount of traffic delivered over unrequested paths.

B. Putting the components together

As described above, we have different forwarding mechanisms (in-packet BFs and Merkle-trees). The idea of virtual links combines both approaches by allowing explicit stateful trees to be included into the in-packet Bloom filters. Finally, the SPSwitch approach tackles the mapping problems we are likely to have at domain boundaries.

We need to combine these mechanisms so that we create state on-demand and only where required, adapting to the actual traffic patterns. This is achieved by relying on semi-stable subtrees represented by virtual links spanning multiple hops. In addition, by employing false-positive-prone forwarding schemes over suboptimal trees, we achieve state reduction and line speed at the cost of some extra delivery. It is a fair and currently necessary trade-off to reach the required levels of scalability. Indeed, a clever design can convert this apparent bandwidth waste into a strength of the architecture (e.g. opportunistic caching, resilience etc.).

IV. Divide and conquer

We present our Divide&Conquer-based approach to tackle the scalability problem of massive amount of overlapping multicast trees for efficient data delivery. We split the problem space along two dimensions: we use hierarchies in the network to achieve scalability and we divide the problem into smaller, more manageable steps.

A. Hierarchical aggregation

As a first approach, we use the traditional and successful way for achieving scalability: hierarchical aggregation. Our requirement can be explained in what we call *scalability*

[1]There is a small possibility that this simple method would create forwarding loops. For loop prevention, see [10].

[2]Consider as a reference the current performance of MPLS/VPN devices under route reflector operations handling up to few millions of flows.

principle: In any given domain *A*, the amount of state corresponding to any remote domain *B* should not depend on the number of subscribers in domain B. The key of this principle is that any node in domain *A* should see domain *B* as only one subscriber, even if there are many real subscribers within *B*. Changes inside domain *B*, therefore, should only cause change in the forwarding states within domain *B*. Topology hiding, like in today's Internet, not only meets the operators' business interest but also helps achieving scalability.

Hierarchy can be implemented recursively on different levels, e.g. consider OSPF areas, autonomous systems (ASes), AS confederations etc. in today's networks. We should, however, not restrict ourselves to this division, as the introduction of the new paradigm may not only change the current AS structure but also the fundamental policies [12] that define what autonomous system means. We should not introduce a constraint in the number of levels to be used: when an operator feels necessary it could introduce a new hierarchy level at any time. For the sake of easier explanation, in the following we will work only with two-level hierarchies: intra-domain and inter-domain levels.

B. A five-step approach

Now we turn our attention towards the other dimension of the division. Our approach can be described as taking a few steps, one after each other. All these operations should take place within each single area that we have in the hierarchy. The steps are as follows:

1) Compute an ideal tree.
2) Determine the gaps between the ideal tree and any existing trees.
3) Select tree-creation strategies or gap filling strategy for each gap.
4) Compute the needed changes according to the strategies.
5) Apply the changes to the network.

1) Ideal tree: The first step is to *compute an ideal forwarding tree* that connects the publisher to all the subscribers. We envision that this will be performed in the vicinity of rendezvous and topology layers, in a (possibly) distributed manner. The ideal forwarding tree is the best tree regarding any appropriate metrics (delay, hop count, etc.). The resulting forwarding tree will usually span several different domains. By introducing hierarchies, we can reduce the problem of creating one overall ideal forwarding tree to creating several intra-domain forwarding trees and an inter-domain (AS-level) forwarding tree connecting them.[3] These ideal forwarding trees will guide us as a reference when we assign the actual trees for the ultimate data delivery.

2) Gaps: In the second step, the topology layer *compares the existing forwarding trees to the ideal ones* and selects the best matching existing trees in each domain. There are no restrictions on the number of existing trees used, logically combining several existing trees is allowed. The resulting best-matching tree either covers all the subscribers in the domain, and therefore can be utilized for publication delivery if other criteria are met (e.g delay requirements) or does not cover some subscribers. In the latter case, it cannot be used for publication delivery in its current form, before it gets expanded. However, we may be able to avoid the latter situation by default including trivial broadcast and short-range unicast trees in each domain.

3) Tree-creation and gap filling strategies: The third step is to *select a tree-creation strategy*, separately in each domain, for creating the actual forwarding tree. If the combination of the existing trees already covers all the subscribers inside a domain, then this combination could be used as the actual delivery tree. However, if any of the determined further constraints are not met, more work is needed. For example, if the best-matching tree contains too many subtrees where there are no interested subscribers, an action must be taken, either immediately or later. In general, it seems a viable strategy to first use the existing, non-optimal tree, and build one or more new trees in parallel, and switch over to the new trees once they are ready. An alternative may be to modify some existing trees, provided that the modified trees still fulfill the requirements of their other use. The exact details, however, depend on how the trees are built and installed, i.e. how much the solution is based on source-routing like approaches and how much on actually representing the trees with explicit state in the forwarding nodes. Our present solution is largely flexible here, as it became apparent from the description of the solution components in Section III-A.

If the combination of the existing trees does not cover all the subscribers, the network must *select a gap filling strategy*. For example, it may decide *to modify existing trees or create new ones* in order to reach all the subscribers and meet other constraints. Note that the compulsory requirement for the new combination of forwarding trees is that it must cover all the subscribers for the given publication.

4) Compute the changes: After the routing entity has selected the strategies, it should *compute the new/modified trees* accordingly. This process is eventually mapping the internal representation of the tree to the representation that is used in the network. Here, different forwarding solutions yield different actions: either installing elements to Bloom filters, or encoding tree structures into Merkle trees, or more possibly combining both of them. Finally, the routing entity should identify the places (network elements) where state injection or update is needed.

5) Apply changes: As the last step, the *changes must be signaled* to all affected entities. This involves sending control messages ordering nodes to install or modify forwarding states, as well as notifying the source of the data delivery if a new forwarding identifier is to be placed into the packet header.

C. An example scenario

Figure 2 illustrates the hierarchical aggregation. The entities contain nested entities (e.g. today ASes contain areas and areas contain routers). Consider that on a level of hierarchy Entity

[3]Note that the resulting overall tree may not be ideal but that it will be policy compliant and that each subtree will be ideal.

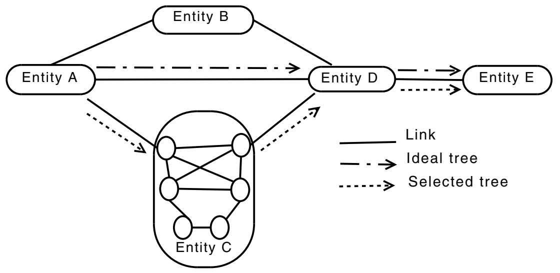


Fig. 2: Example of hierarchical aggregation and tree selection

A is the publisher (which means that it contains the publisher host) and Entity D and E are the subscribers. From the hop count point of view, the tree A-D-E is the ideal. Assume that there exists another publication which has the tree A-C-D-E. The routing system therefore can assign this tree for the new publication as well, as it contains all subscribers. Consequently, we should not add new forwarding state as we have not created any new tree on that level. Finally, when the publication reaches e.g. E, it should switch to an internal tree to reach the subscribers residing inside. However, there will not be any internal forwarding tree for this publication in Entity C. By caching the publication, Entity C may as well increase the network performance by supporting e.g. faster error correction (implementing the network coding *More* functions envisioned in RTFM). Of course, if some policy dictates that the ideal tree should be used or any requirements are not met, the delivery of the publication can be switched to it as soon as all the necessary states are installed in the network.

D. Towards a pub/sub routing protocol

A key to implement the five-step mechanism is to have a (distributed) entity in the network that is topology-aware. We suggest a GMPLS Path Computation Element (PCE)-like [13] topology manager, adopted into our pub/sub world.

The pub/sub PCE collects the topology information of the network domain by subscribing to link advertisements coming from the nodes (this requires to have a default path between any node and the PCE). The content of the link advertisements may vary depending on the actual forwarding mechanism used. Finally, the pub/sub PCE will have a full view of the topology of the domain. Assuming that publish and subscribe messages collect the path between the PCE and the publishers/subscribers, the PCE can locate all the entities and can compute the trees.

The difference of the current practice is the absence of link advertisement flooding, as only the PCE should build the network graph and other participants do not need to subscribe, meaning that changes trigger less control messages than today. Considering the inter-domain case, the cooperation of pub/sub PCEs results in the appropriate inter-domain forwarding structure.

E. Challenges of the division

The operations at the hierarchical boundaries (e.g., edge router mapping of trees) represent the main scalability challenge of our Divide&Conquer approach. Obviously, a full source-route would circumvent the edge-mapping problem. However, strict source-routing has its limitations and price (see Sec. II). Therefore, we need to consider suitable label swapping and stacking mechanisms with an optimal balance of the size of network state and packet headers, and signaling overhead.

During the tree construction phase, the appropriate set of forwarding trees (potentially multi-domain) has to be chosen so that all domain internal subscribers are covered. At the domain boundaries, when an edge forwarding node receives a packet over an inter-domain tree, the edge node must determine the right internal tree; similarly, after receiving packet on an intra-AS tree, it must determine the right inter-domain tree. Obviously, there will be no one-to-one mapping between the trees at different levels..

To change between the hierarchy levels, one possibility is to look inside the packet, using the publication-level identifiers. This approach seems unfeasible from a scalability point of view considering the potential amount of *active* rendezvous identifiers travelling an edge during a certain time window (publication data delivery time).

To benefit this fact that potentially many publication identifiers will require the same mapping of trees, we propose the use of a special set of ”non-routable” link IDs with the goal of triggering the mapping. Typically, we need at least one virtual link from each edge node in the AS to the rest of the edge nodes. The amount of intra-domain virtual links “bridging” edge routers and providing a fast path to each neighbouring AS will be in the order of a few hundreds.[4] On packet reception, the edge node checks for the presence of this special link idenitifiers in the Bloom filter and will push the appropriate forwarding label for intra-domain delivery avoiding the publication identifier look-up.

Another solution space of the mapping problem is determined by the notion of information scoping. While scope identifiers (SIds) are meant to guide the pub/sub directives to the suitable rendezvous points in the network for matching, we consider to include the SId in the actual data packets to take edge filtering decisions at this granularity. Since scopes are aggregating data in a semantic layer, it makes sense to think of re-using this aggregation for communal transport services at the forwarding layer.

We are aware of additional challenges that deserve more attention, including extensive evaluation of the trade-offs, Bloom filter check performance, implications of the delay in the multi-step approach, and specific issues w.r.t. content-orientation of inter-domain routing policies. Each of these challenges will be the subject of upcoming papers.

[4]Given the power-law distribution of an AS degree. In practice, maximal 1024 AS neighbours can be assumed [14], [15].

V. Related Work

We are certainly not the first to work with routing&forwarding problems in future networks. However, to the best of our knowledge no prior work has proposed the adoption of the pub/sub paradigm throughout the stack in Internet-scale, and described its relation to routing.

TRIAD [16] was among the first proposals of a *content-based routing* design in the sense that it routes on URLs by mapping fully qualified domain names (FQDN) to next-hops. While similar in the spirit of data-centrism, our work is more ambitious and aims at a finer granularity of content, namely individual pieces of information objects (pub/sub channels or documents).

IP *multicast* can be viewed as a special case of a data-oriented networking service. In practice, a multicast address is a name (cf. a pub/sub topic) rather than a true network identifier. IP multicast issues w.r.t. complexity and deployability incentives have been widely discussed [15]. The authors of [15] revisit the case of IP multicast and propose Bloom filters to aggregate the active multicast groups inside a domain, piggybacking this information in BGP updates. Their IP multicast design also includes a Bloom-filter- based shim header in packets to represent AS-level paths of multicast packets. Our work handles links as more general destination information than IP prefixes and explores more dimensions of the routing&forwarding space.

ROFL [17] proposes Internet-scale routing scheme on flat host identifiers based on neat DHT constructs, but suffers from policy-compliancy issues [12] and larger stretch. AIP [18] is based on a two-level (domain and host) routing architecture with self-certifying domain and host identifiers to account on an Internet scale. The future internetworking proposal in [19] also combines the notion of separating of routing and forwarding using generic link identities.

DONA [3] employs flat self-certifying labels for data objects operated by *find/register* primitives over the legacy IP network. The main difference in our work is that we do not assume underlying IP forwarding. Besides policy and incentive compatibility issues, DONA suffers from scalability problems [12] near Tier-1 operators as they require an entry for each single registered publication in the global network. This suggests that a new data plane design is required to support global data-oriented internetworking.

VI. Conclusion and Future Work

This paper explored the routing options and problems of a new (inter-)networking layer based on the publish/subscribe paradigm. We moved a step forward towards pure pub/sub routing by dividing the problem in two dimensions. First, we use hierarchical aggregation; second we describe five steps that together solve the problem of routing: (1) Construct the best possible (ideal) forwarding tree on the known topology. (2) Examine the existing forwarding trees and combine them to best match the ideal tree. (3) If it does not reach all subscribers, or it does not meet any additional requirements, select tree-creation and gap-filling strategies, then (4) compute the modifications. Finally, (5) push the necessary information to the affected entities and update the source of the data delivery, if needed.

The ultimate goal we strive to achieve is to find a solution, which has (A) reasonable signalling overhead in dynamic condititions and (B) fairly low stretch, and which minimizes (C) the unnecessarily used network resources and (D) the per-packet overhead. Finding a right balance will be crucial for future work. Our plan for future work also includes validating the overall solution with implementation (NetFPGA routers, FreeBSD nodes) and ns-3 simulation works.

Acknowledgment

We would like to thank Petri Jokela, Jarno Rajahalme and Somaya Arianfar for their valuable comments. This work was supported by the EU FP7 PSIRP project under grant INFSO-ICT-216173.

References

[1] V. Jacobson, "If a clean slate is the solution what was the problem?" Stanford "Clean Slate" Seminar., Feb 2006.

[2] C. Esteve Rothenberg, F. Verdi, and M. Magalhães, "Towards a new generation of information-oriented internetworking architectures," in *First Workshop on Re-Architecting the Internet*, Madrid, 2008.

[3] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM CCR*, vol. 37, no. 4, pp. 181–192, 2007.

[4] D. Trossen (ed.), "Conceptual Architecture of PSIRP Including Subcomponent Descriptions (D2.2)," http://psirp.org/publications, June 2008.

[5] M. Särelä, T. Rinta-aho, and S. Tarkoma, "RTFM: Publish/subscribe internetworking architecture." ICT Mobile Summit, Stockholm., 2008.

[6] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton, "Haggle: Seamless networking for mobile applications," in *Ubicomp*, 2007, pp. 391–408.

[7] H. Holbrook and B. Cain, "Source-Specific Multicast for IP," RFC 4607, August 2006.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[10] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," http://www.psirp.org/files/Deliverables/PSIRP-TR09-0001-LIPSIN.pdf, PSIRP EU FP7 project, Tech. Rep., 2009.

[11] R. C. Merkle, "A certified digital signature," in *CRYPTO '89: Proceedings on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1989, pp. 218–238.

[12] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented networks," in *First Workshop on Re-Architecting the Internet*, Madrid, Dec 2008.

[13] A. Farrel, J.-P. Vasseur, and J. Ash, "A Path Computation Element (PCE)-Based Architecture," RFC 4655, Aug. 2006.

[14] S. Jiang, "An addressing independent networking structure favorable for all-optical packet switching," *SIGCOMM CCR*, vol. 37, no. 1, pp. 17–28, 2007.

[15] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting ip multicast," in *Proceedings of ACM SIGCOMM'06*, Pisa, Sep. 2006.

[16] D. R. Cheriton and M. Gritter, "Triad: A new next-generation internet architecture," http://www-dsg.stanford.edu/triad/, July 2000.

[17] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, "ROFL: Routing on flat labels," in *Proceedings of ACM SIGCOMM'06*, Pisa, Italy, Sep. 2006.

[18] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable internet protocol (aip)," in *ACM SIGCOMM*, 2008.

[19] L. B. Poutievski, K. L. Calvert, and J. N. Griffioen, "Routing and forwarding with flexible addressing," *Journal Of Communication and Networks*, vol. 9, pp. 383–393, Dec. 2007.

Appendix D

Publication D

C. Esteve Rothenberg, P. Jokela, P. Nikander, M. Särela and J. Ylitalo. "Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters." In *5th European Conference on Computer Network Defense (EC2ND)*, Nov. 2009, Milan, Italy.

Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters

Christian Esteve Rothenberg
School of Electrical and Computer Engineering
University of Campinas, Brazil
chesteve@dca.fee.unicamp.br

Petri Jokela, Pekka Nikander,
Mikko Sarela, Jukka Ylitalo
Ericsson Research, NomadicLab, Finland
{firstname.lastname}@ericsson.com

Abstract—**In this paper, we propose and analyze an in-packet Bloom-filter-based source-routing architecture resistant to Distributed Denial-of-Service attacks. The approach is based on forwarding identifiers that act simultaneously as path designators, i.e. define which path the packet should take, and as capabilities, i.e. effectively allowing the forwarding nodes along the path to enforce a security policy where only explicitly authorized packets are forwarded. The compact representation is based on a small Bloom filter whose candidate elements (i.e. link names) are dynamically computed at packet forwarding time using a loosely synchronized time-based shared secret and additional in-packet flow information (e.g., invariant packet contents). The capabilities are thus expirable and flow-dependent, but do not require any per-flow network state or memory look-ups, which have been traded-off for additional, though amenable, per-packet computation. Our preliminary security analysis suggests that the self-routing capabilities can be an effective building block towards DDoS-resistant network architectures.**

I. Introduction

An old question is whether routing should happen in a hop-by-hop manner or as source routing. With the proliferation of the Internet, hierarchical hop-by-hop routing won the day. Unfortunately, that came with a price: the network serves more the sender than the receiver. The hop-by-hop approach makes its best to deliver a packet to the destination, whether the receiver wants to receive it or not, opening a venue for unwanted traffic. Various remedies, such as firewalls, deep packet inspection (DPI), and explicit capabilities, have been proposed to address the problem, with variable success.

Network capabilities, as introduced by Anderson et al. [1], are architectural approaches that enable secure statements attached to packets, allowing routers to easily check if a packet has been approved by the receiver. They are typically based on cryptographic approaches that enable routers to verify packets in a stateless way, though some statements, such as those related to a maximum bandwidth, do require state [24]. When capabilities are required, any prospective sender must first retrieve a suitable capability, either directly from the receiver (using explicit bandwidth reserved for that), out of band, or through a trusted third party [19].

In this paper, we present a system where there is no need to have capabilities separate from forwarding identifiers; i.e., where the capabilities work as a forwarding identifier and vice versa. Building upon LIPSIN [10], a native multicast forwarding method based on *in-packet Bloom filters (iBF)*, we introduce a DDoS resistant forwarding service. We construct a system where having separate capabilities is unnecessary, as with our iBF-based forwarding identifiers it becomes computationally hard to extract path information for constructing new capabilities, without insider help.

Addressing Denial-of-Capability (DoC) attacks [2], which aim at preventing new (legitimate) capability-setup packets from reaching the destination by overwhelming the system with capability requests, is out of scope of this work. Recent work has shown effective means to mitigate DoC attacks. For instance, in addition to treating capability request packets as best-effort packets [1], [22], the system can allocate scarce link bandwidth for connection establishment packets based on per-computation fairness (cf. Portcullis puzzle system [14]), or by allowing a well provisioned third party service to act as capability service [19]. Depending on the needs of the service to be contacted, such third party may require cryptographic identity verification, monetary payment or a guarantee, or require a user to solve a hard AI problems e.g. CAPTCHAs [18].

Our approach differs from existing capability-based systems in that (1) the capability has a *fixed size,* independent of the number of hops,[1] (2) the capability acts also as a *forwarding identifier* in a stateless fashion with no forwarding table lookups, and (3) the system is *multicast* friendly.

The rest of this paper is organized as follows. In Sec. 2, we briefly recap the LIPSIN forwarding approach and discuss associated DoS vulnerabilities. In Sec. 3, we describe our proposed DDoS-resistant enhancements. Sec. 4 contains a statistical analysis of the DDoS-resistance properties. In Sec. 5, we briefly discuss related work, concluding the paper in Sec. 6.

II. Background

Using the approach of [16], we divide networking into three components: rendezvous [17], topology [25], and forwarding [10]. The rendezvous is responsible for matching sources and sinks and for instructing the rest of the system. From the security point of view, it acts as a capability distribution center [19], [5]. The division between the topology and the forwarding components is similar to those of the routing as a service proposal [12] and the direct network control approaches, such as 4D [23]. We believe such a division can

[1]There is a practical upper limit (of ≈ 40) for the number of hops; see [10] for a detailed analysis.

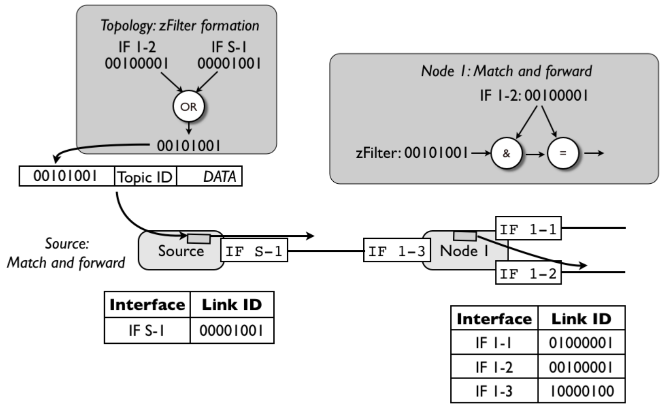


Fig. 1. zFilter creation and forwarding in LIPSIN.

be achieved with a (distributed) topology service, similar to the Path Computation Entity (PCE) [8] in (G)MPLS.

Within this networking model, there are three main avenues for Distributed Denial-of-Service (DDoS) attacks. (1) An attacker may try to attack 'legitimately', i.e. through rendezvous (gaining one or more forwarding identifiers). (2) An attacker may try to overload the rendezvous system with excess requests. (3) The attacker may try to guess or construct a forwarding identifier so that it can overload the target, without receiving help from the rendezvous or topology components. In this paper, we focus on the last one, relying on existing work on capability-based systems, over provisioning, and contractual relationships to solve the former two [1], [22], [14]. Additionally, we exclude insider threats (e.g., Byzantine routers), leaving them for future work.

A. The LIPSIN forwarding mechanism

The LIPSIN [10] forwarding solution does not name nodes or interfaces. Instead, links are named, separately in each direction. Consequently, each forwarding identifier is essentially a set of Link IDs, denoting a delivery tree or a path compressed as a Bloom filter [4] called zFilter.

In practise, each Link ID is an m-bit long string with k bits set to one, with $k \ll m$, and m relatively large. This makes Link IDs statistically unique. For instance, with $m = 256$ and $k = 5$, we get $\approx m!/(m-k)! \approx 10^{12}$ different Link IDs.

The Link IDs are used at two distinct instances. First, to construct a zFilter for a given delivery tree T, the topology component takes a binary OR over the IDs of the links forming the tree (see Fig. 1). The resulting zFilter Z_T is then passed to the source, allowing it to send packets along the delivery tree using the zFilter as the forwarding identifier. Second, when a forwarding node receives a packet, it needs to determine where to forward the packet to. For each outgoing link o, the node checks if the zFilter Z_T contains 1s in those bit positions where the Link ID L_o does. If so, the node forwards the packet along that link; i.e., if $(Z_T \wedge L_o) \equiv L_o$, then forward the packet over o. If the zFilter contains multiple outgoing Link IDs, then the packet is forwarded to each of them, resulting in multicast. Also, as is well known, using Bloom filters introduces the possibility of false positives; their probability rises as more links are included in the iBF.

The Link ID Tag (LIT) mechanism, also described in [10], provides control over the false positives by defining d different names for each outgoing link. Consequently, any given delivery tree can be described with d different iBFs, each of them having different bit patterns. This allows iBF selection based on different criteria, such as fewer false positives.

B. Remaining forwarding vulnerabilities

There appears to be a few vulnerabilities that the basic LIPSIN approach, and any naive source routing forwarding scheme, does not protect from. First, while a given zFilter works only from its source to its sink(s), the same zFilter can be used also for other traffic that what it was meant for. We call this a *zFilter replay attack*. Second, while the used encoding helps to hide the link identifiers, correlation between iBFs is still possible, creating a *computational attack*; see below. Third, while each zFilter is directly usable only by the source and any *en-route* nodes, if an attacker can figure out another zFilter that passes through any of the en-route nodes, it can *inject* traffic to the delivery tree.

In the computational attack, an attacker collects valid, related zFilters and analyses them. Wherever the bit patterns are similar among a group of zFilters, it is likely that any reoccurring bits represent a partial graph common to those zFilters. Hence, knowledge over a large number of $\langle$source, sink(s), zFilter$\rangle$ triples may allow an attacker to create valid zFilters towards a target. By merging correlation pairs from multiple sites (e.g., using bots), DDoS attacks might well be possible. While the introduction of the LIT construction makes this attack computationally more expensive, especially when d is large, the attack appears to remain practical.

To analyse the difference between the original zFilter proposal and our proposal, we introduce a formal security model in Sec. IV. There we evaluate a few attacks in terms of success probabilities and associated costs.

III. Secure in-packet Bloom Filters

To address the above-described security problems (and potentially other, still undiscovered ones), we now propose a system in which the link names are periodically changed and tied to the path and to an upper layer flow identifier. By this we mean an identifier that upper layer, e.g. transport, uses to identify packets belonging to different applications. The link names (and consequently the zFilters) are tied to upper layer flow identifiers to ensure that only packets belonging to a flow requested or approved by some application is delivered. The flow identifier can, but does not need to, be based on IP addresses. Any identifier that upper layers decide to use suffices e.g., topic ID in pub/sub systems.

Instead of relying on a set of pre-defined (but perhaps time-dependent) names for each link, our solution is based on dynamically computing link names depending on the packet contents, the path the packet is taking, and perhaps other context-dependent parameters.

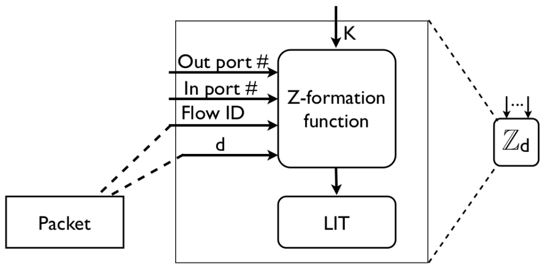


Fig. 2. z-Formation function.

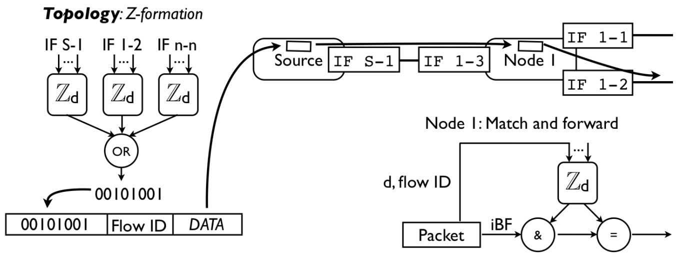


Fig. 3. z-Formation creation and check in the forwarding node.

A. z-Formation

The key idea of the z-Formation is to enable forwarding decisions in a completely dynamic, computational fashion where the iBF, the packet content, and the processing context is used to determine where the packet should be forwarded, if at all. Instead of maintaining a fixed forwarding table containing the Link IDs (or LITs) for each outgoing interface, the *z-Formation* dynamically computes the names of the candidate links on a packet-basis. A function Z computes the LITs using (i) some in-packet information I (a Flow ID), (ii) a periodically changing secret K, (iii) the incoming and outgoing interface indices (In, Out), and (iv) the Link ID Tag index d (see dynamic LIT computation in Fig. 2). As in LIPSIN, also here each LIT $O = Z(I, K(t_i), In, Out, d)$ is a Bloom mask of m bits. As the zFilter is now constructed using these dynamic LITs instead of static LITs, the resulting zFilter becomes additionally bound to the Flow ID, a specific time period, and the input port. Especially, having the Flow ID I as an input parameter ties the given zFilter to only those packets carrying the specified Flow ID, which, for example, makes reactive filtering an easier task (cf. Sec. IV-D).

To construct the time-bound shared secrets K, each forwarding node i shares a master key K_i^m with the topology manager. For any time period t, forwarding nodes compute $K_i(t) = F(K_i^m, t)$, where F is cryptographically secure pseudo-random function. For example, t may be a seeded counter or wall time clock at a coarse enough granularity; in either case, the forwarding nodes and the topology manager need to have loosely synchronized clocks [6]. The topology manager always uses the current valid value of t while forwarding nodes also accept j (one or a few previous) values. In this way, if t is advanced every Δt seconds, even if $K_i(t)$ is compromised for a specific t, the attack is limited to the single forwarding node using the key and to the maximum time of $j\Delta t$.

Finally, as Z takes in both the *outgoing* and *incoming* interface indices as inputs, any given zFilter is tightly bound to the corresponding forwarding path or delivery tree. That is, this feature blocks the injection attack, preventing off-path attackers from sending data towards a delivery tree even if they know both the Flow ID and the zFilter. Additionally, including the incoming interface index as an input-parameter allows us to introduce *virtual interfaces* within forwarding nodes, thereby enabling on-path services.

Secure iBF generation: Identical to the LIPSIN approach, we expect the topology component to generate a zFilter as a result of a path formation request. Using the Flow ID I from the request, the current secret keys $K_i(t)$ and the required interfaces from the nodes in the computed network graph (cf. [8], [12]), the topology component applies Z for each d. Note that as partial results can be easily combined, this operation may be distributed along multiple (e.g., per-domain) topology components. Given the final d candidate iBFs, the topology component picks the best one and hands it over to the source. Section III-B elaborates on how zFilters updates can be pre-computed, requested by receivers, and redistributed to sources every Δt seconds.[2]

Secure iBF forwarding: When a data packet arrives at a forwarding node, the node extracts d and I from the packet. With I, d, the incoming interface index, and the current $K_i(t)$ (plus optional j older) value(s), it computes the LIT for each outgoing link. If the iBF matches the on-the-fly generated LIT, the packet is forwarded along the interface. Dynamic LIT computation can be easily done in parallel for each interface. Note that forwarding nodes are freed from storing any per-flow state or traditional FIB table lookups. Only the *seed* of the secret K and the current accepted values need to be maintained.

Z-function implementation: The Z-function can be implemented as a stream-cipher-like construction, tailored to give constantly out $\approx k$ 1-bits instead of the usual average of $\approx m/2$ 1-bits. Internally, the function may resemble a keystream generator, initialized with a combination of the values K, I, d, In, Out. As typical stream ciphers can be implemented in hardware with only a few shift registers and logic gates (see e.g. [9], [21]), we surmise that the needed circuit could work at full OC-768 line speed.

B. Updating zFilters

If a flow is valid for longer than $(j-1)\Delta t$, the source needs a new zFilter once the old one is about to expire. However, we defer the specific description and evaluation of updating zFilters to future work and merely note that there are at least two methods for doing so. The sink can indicate its willingness to continue receiving by responding to the traffic and instructing the intermediate forwarding nodes to construct an up-to-date zFilter en-route. This can be embedded in upper layer messages, e.g. in transport protocol acknowledgements.

[2]Due to the ability to combine partial results, each part of the network may have a different Δt.

Alternatively, a source can obtain a new zFilter from the rendezvous system. This is similar to other rendezvous-based solutions (e.g., HIP RVS servers [11], mailboxes in [7]). It enables both the sender and the receiver to express their interest in extending their communication. The topology system needs to be instructed to construct the new zFilter, typically using the same path but with the current value for each K_i along the path. As $K_i(t+1)$ can be generated locally, zFilter updates can be easily pre-computed and conveyed to the source (even before the oldest valid key expires), which can use it as the new routing capability for data delivery.

C. Applicability to IP networks

We believe that the z-Formation could be used with IP networks, though there are still many open issues related to NATs and other middle-boxes, partial deployment, and security, to name a few. The 5-tuple $\langle IP_{src}, IP_{dst}, P_{src}, P_{dst}, prot.\rangle$ can serve as the flow ID and the z-Filter will be used for routing decisions within those ASes already deployed. The best place for the z-Filter is likely to be as an extension header after the IPv6 header (or as an IP protocol on top of IPv4).

A group of interconnected ASes can use zFilters to route all traffic within and coming to the group. If a packet without a zFilter comes, it will be sent to the rendezvous system, which will determine whether an approval in the form of zFilter tied to the used 5-tuple will be given to the sender. Support for zFilters can be implemented as a shim layer between IP and transport, or as a separate proxy. Assuming, the victim does not respond to attack traffic, even a single, or a few ASes can reduce the severity of attack by several orders of magnitude as shown in Section IV-B. Additionally, attacks based on copying a single zFilter to multiple attackers are simple to filter, as they require the same 5-tuple in all packets.

IV. Analysis

Preventing an attacker from injecting large numbers of packets without approval via rendezvous is a necessary, though not sufficient condition for DDoS resistant architecture. We now evaluate the effectiveness of the z-Formation forwarding mechanism when malicious nodes try to compromise the network availability by injecting unwanted traffic. First, we introduce the mathematical framework to describe the forwarding model. Then, we use probabilistic methods to quantify the achievable levels of DoS protection. Later on, we discuss the effects our architecture has on replay attacks and computational attacks.

A. Label-based forwarding model

Let $G = (V, E)$ be a network graph, in which all edges are named with Link IDs, i.e. directional bit vectors $e = \{0,1\}^m$, where the number of 1s in each bit vector $e(v_i, v_j) \in E, v_i, v_j \in V$ is exactly k and the 1s are randomly distributed. Let $p(v_0, v_n)$ be a path in the network such that $\langle v_i, v_{i+1}\rangle \in E, \forall i < n$.

Define zFilter $z = \{0,1\}^m$ as an m-bit long string with maximum density ρ_{max} such that zFilter $z_1 \wedge z_2 = z_1 \iff z_1 \subseteq z_2$ and edge $e(v_i, v_j) \in z, \iff e \wedge z = e$. Thus, a

TABLE I
Capacity in terms of edges before reaching the maximum allowed iBF density

ρ_{max}	m = 256		m = 196		m = 128		$fpr_{k=5}$
	#e	$\#e_{opt}$	#e	$\#e_{opt}$	#e	$\#e_{opt}$	
0.45	30	34	23	27	15	17	1.85%
0.50	35	39	27	31	18	20	3.13%
0.55	40	44	31	35	20	22	5.03%

path is encoded in the zFilter if each edge within that path is encoded in the zFilter. We write $p(v, w)$ to denote the smallest zFilter that encodes path p, i.e. $p(v_0, v_n) = \bigcup_{i=1}^{n} e(v_{i-1}, v_i)$.

Assumptions: No forwarding node on any path is hostile. The Link IDs are random and uniformly distributed, that is the secret value K is random and the Z-formation has the property that if K is random, $O = Z(K, I, ...)$ is sufficiently random. The z-Formation produces a different Link ID for each link depending on the incoming interface of the packet. This effectively adds a single hop to the length of the path that the attacker must guess. We use the above described formalism with constant Link IDs in the analysis.

B. Attack description and evaluation

We assume the attacker v_m knows a legitimate zFilter $z(p(s,t))$ between a non-malicious source v_s and target v_t and show that it is difficult for the attacker to create a valid zFilter from malicious nodes to target with it. $V_p = (v_s, ..., v_t)$ is the set of nodes on path p.

Given z with a fill rate of ρ, the number of possible edges e included is $\binom{\rho \cdot m}{k} \approx 3 * 10^8$ for $m = 256, \rho = 0.5$ and $k = 5$. With no way to test off-line for the validity of single edges within z, the first attack strategy consists of randomly trying a set of $z_i \supseteq z$ to see if it can deliver packets through, i.e. if $p(v_m, v) \in z_i, v \in V_p$. Hence, the attack model is based on *brute force* and is equivalent to a randomly generated z causing false positives along the path(s) toward the set of nodes V_p (as the path V_p is included in each zFilter z_i). However, if a known z is used at its maximum capacity ρ_{max}, the attacker cannot set additional bits to include extra edges in z.

In both cases, we can assume that z has $\rho_{max} * m$ bits set to 1. The threat consists of $p(v_m, v), v \in V_p$ being encoded in z. The probability of a non-included edge having its k bits set to one in any z depends only on the fill factor and is equal to ρ_{max}^k. Table I shows the estimated false positive rate and the average number of edge labels, with and without the $d = 8$ LIT optimization, that can be inserted before reaching ρ_{max}.

When an attacking node is h hops away from any node in V_p, the attacker needs to cause h false positives to get the packet forwarded to and through the valid path. Thus, the probability of a successful attack, i.e. $z \wedge p(v_m, v) = p(v_m, v), v \in V_p$, is equal to $Pa = \rho_{max}^{k \cdot h}$. Figure 4 shows, at the left axis, the estimates of Pa for different maximum fill factors and attack path lengths h. An attack path length of one means that the attacker is a neighbor of some node on path. In this case, the probability of falsely forwarding a forged packet

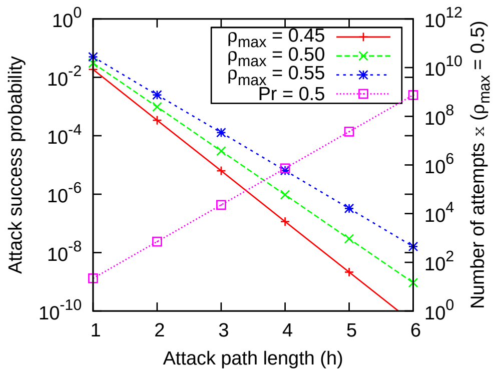


Fig. 4. On the left axis, attack success probability for different ρ_{max}. On the right axis, the line with square points represents the attempts required to success with probability 1/2.

is around 3%. When the malicious node v_m is more hops away from the valid path, this probability sinks drastically, i.e. 10^{-9} for v_m to guess a working label path over 5 hops.

Finally, we can determine how expensive it is to guess a zFilter from v_m to any other target node v_t. We estimate how many attack attempts (x), consisting of randomly generating maximally filled z, are required to have some probability Pr of obtaining at least one valid path label to a destination v_t h hops away from v_m (the right axis on Fig. 4 shows $Pr = 0.5$):

$$x = \log_{1-Pa}(1 - Pr) \qquad (1)$$

C. Discussion

Tying the zFilter to a set of periodically changing keys, one for *each* forwarding node, makes replay attacks less severe, as they can only be used during the joint lifetime of the keys. It also makes computational attacks more difficult. Even if the attackers know the full topology of the network, tying the zFilter to each forwarding node's secret key reduces the best attack strategy to a brute force attack consisting of generating random labels and hoping that at least one of them reaches the target(s). At the same time, the seed-based re-keying scheme is local and introduces low communication overhead (low frequency seed exchanges) between forwarding nodes and distributed topology instances, which can easily anticipate to zFilters update requests.

Assuming that the attacking node is capable of injecting 10^6 packets per second (e.g., 1Gbps edge link and 1000 bits per packet), a malicious node will need over 40 minutes to guess, with probability 1/2, a working label for a 5-hop path. If the receiver of the traffic does not answer to such packets, then the system reduces the magnitude of attack traffic by the percentage of packets filtered enroute. An attacking host sending packets 2 hops away from the target, with $\rho_{max} = 0.5$, would only be able to get (approximately) 0.1% of the attack traffic into the path. Thus, changing keys as slow as once every 20 minutes, the forwarding plane can be protected for paths longer than 4 hops with very high probability.

Under some circumstances, we may want to shorten the time for which any host can receive unwanted traffic, empowering the receiver by means of the capabilities renewal mechanisms. For instance, a re-keying frequency of around 1 minute would be short enough to (i) protect very short paths and to (ii) limit the duration of DDoS attacks based on the misuse of legitimate zFilters. Typically, an expiration interval in the order of a few dozens of seconds is long enough to complete average transactional traffic without requiring zFilter renewal.

Note that we have not only assumed a very high and constant attack traffic injection rate (10^6 pps) but also the existence of a return channel to know whether randomly generated z reach the intended victim(s). As a final remark, the DDoS protected forwarding plane only complements additional security measures at the end node higher level stacks similar to end-host firewall implementations where only solicited (subscribed) data flows are allowed and processed.

D. Attack detection and mitigation

By virtue of the time-based re-keying mechanism, a forged path lasts only for $j\Delta t$ in the worst case. After that, a malicious node would need to re-initiate the attack process. As the most efficient attacks we are aware of require excessive probing, an attack can be detected early by the sudden increase on false positives caused by the falsely labeled packets injected by the attacking node(s). Hence, a blacklist mechanisms can be used to block or shape down any suspicious traffic. By definition of an in-packet flow identifier (I), each attack needs to be tailored for a specific I. This does not only limit the scope of an attack but also eases any blacklisting mechanisms based on I.[3]

V. Related Work

Anderson et al. [1] were the first to propose in-packet capabilities. More generally, [1], [5], [7], [24] are all close to our approach in the sense that for sending the sender needs a permission from the receiver. Compared to our work, the main difference is that in our case the forwarding identifier (bound to an upper layer flow identifier), acts by itself as the capability; additional capability fields or cryptographic end-to-end schemes are not needed.

SANE [5] was, to our knowledge, the first proposal to combine centralized computed source routes and capabilities together. It achieves that, by encrypting each hop in layers, with a big routing identifier of $10 + 14 \cdot hops$ bytes; e.g. a 13-hop path would require 1536 bits. Therefore it cannot be considered scalable to Internet-wide scales.

Our secure fast-path forwarding mechanism is close to the stateless path pinning service provided by SNAPP [13], where IP forwarding decisions are cached into the flow into the flow initiating packets. The chain of securely constructed forwarding directives is returned to the sender who is now able to use them as packet headers enabling fast switching decisions and additional benefits from the separation of routing from forwarding, including sender-controlled paths, expensive

[3]Of course, if the underlying link technology offers any trustful node identification mechanism, the filter rule can be built on the network node identity of the malicious node.

route lookups, sender anonymity, and accountability. A key difference of our solution is using an in-packet Bloom filter to encode the pinned path, whith the main performance benefits of a fixed header size independent from the number of hops at the cost of some amount of false positives. While our presented networking model involves a rendezvous service using topology information, an en-route capability formation similar in spirit to IP switching could be easily supported.

Also advocating for the separation of routing and forwarding, Platypus [15] proposes a capability-based system to enable authenticated and policy-compliant IP source routing. Functionally similar to our Z-function is their usage of a distributed temporal secret to verify flow binding capabilities at line speed. However, our capabilities are not transferable per design as they securely embed the routing information.

Ballani et al. [3] were the first to use in-network Bloom filters for pro-actively filtering distributed denial-of-service attacks. Our forwarding plane attains a similar off-by-default behaviour but does this by matching the communication interest of sources and sinks over a slow-path rendezvous network instead of propagating over the network routing infrastructure explicitly reachability directives signaled by end hosts.

Phalanx [7] combines capabilities with a multi-path-aware overlay and shows that Bloom filters can be used to reduce state requirements while providing probabilistic guarantees for in-network security. Yang et al.propose TVA [24] , a capability approach that shows how the router state can be bounded even when routers enforce bandwidth limits for capabilities.

In [20], Wolf presented a mechanism where packet forwarding is dependent on credentials. If the packet does not contain a correct credential, it will not be forwarded. The credentials are packed into a small in-packet Bloom filter and each router verifies that its credentials are included. Compared to ours, in their work routing is based on traditional IP addresses and the credentials are issued for a specific IP 5-tuple flow.

VI. Conclusion

In this paper, we have proposed a source-routing-based packet forwarding mechanism that is highly resistant to distributed denial-of-service attacks. In essence, in our approach the hosts have no names; only links are named. To be able to send, any prospective sender needs to acquire a forwarding identifier that simultaneously acts as a path capability. By delegating capability creation to a rendezvous component that explicitly considers both the senders' and receivers' interest, we shift power from senders to receivers. In our approach, receivers are in control of what they want to receive.

To our knowledge, our approach is the first that combines forwarding identifiers and capabilities in an efficient way, thereby creating *self-routing capabilities*. By virtue of the Bloom-filter-based construction of the capabilities, even the links names remain statistically undisclosed. By binding the routing capabilities to specific flows, time periods, and network paths, we create a high barrier for attackers, making it hard to forge valid capabilities.

While the forwarding operation in our scheme is computationally heavier than in LIPSIN, we surmise that it still can be implemented in line-speed hardware, as no memory lookups are required, the number of logic gates appears to be moderate, and the required dynamic per-packet state can be stored in simple shift registers. The verification of this assumption remains as future work.

VII. Acknowledgements

Writing of this paper was supported by TEKES (Finnish Funding Agency for Technology and Innovation) as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *Hotnets II*, pages 39–44, 2004.
[2] K. Argyraki and D. Cheriton. Network capabilities: The good, the bad and the ugly. *ACM HotNets-IV*, Jan 2005.
[3] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default. *ACM HotNets IV*, Aug 2005.
[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.
[5] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *USENIX-SS'06*, Berkeley, CA, USA, 2006.
[6] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, 1981.
[7] C. Dixon and T. Anderson. Phalanx: Withstanding multimillion-node botnets. *Usenix NSDI*, 2008.
[8] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (PCE)-based architecture. IETF RFC 4655, 2006.
[9] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj. Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. SASC'08, Feb 2008.
[10] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander. LIPSIN: Line speed publish/subscribe inter-networking. In *Proceedings of ACM SIGCOMM'09, Barcelona, Spain*, Aug. 2009.
[11] J. Laganier and L. Eggert. Host identity protocol (HIP) rendezvous extension. Technical report, RFC 5204, April 2008.
[12] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a service, 2004.
[13] B. Parno, A. Perrig, and D. Andersen. Snapp: stateless network-authenticated path pinning. In *ACM ASIACCS '08*, 2008.
[14] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Sigcomm*, 2007.
[15] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. *SIGCOMM CCR*, 34(4):167–178, 2004.
[16] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, 2008.
[17] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *MobiArch '08*, 2008.
[18] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. *LNICS*, pages 294–311, 2003.
[19] D. Wendlandt, D. Andersen, and A. Perrig. Fastpass: Providing first-packet delivery. *Technical report CMU cylab*, 2006.
[20] T. Wolf. A credential-based data path architecture for assurable global networking. In *IEEE MILCOM*, 2007.
[21] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State-of-the-art implementations and attacks. *ACM Trans. Embed. Comput. Syst.*, 3(3):534–574, 2004.
[22] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate DDoS flooding attacks. *Security and Privacy*, 2004.
[23] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4D network control plane. In *NSDI'07*, 2007.
[24] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting network architecture. *IEEE/ACM Trans. on Networking*, 16(6):1267–1280, 2008.
[25] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve. Exploring the pubsub routing/forwarding space. In *ICC FutNet09*, 2009.

Appendix E

Publication E

C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Zahemszky. “Data center networking with in-packet Bloom filters.” In *28th Brazilian Symposium on Computer Networks* (SBRC), May 2010, Gramado, Brazil.

Data center networking with in-packet Bloom filters

Christian Esteve Rothenberg,[1] **Carlos A. B. Macapuna,**[1]
Fábio L. Verdi,[2] **Maurício F. Magalhães,**[1] **András Zahemszky**[3]

[1]FEEC – University of Campinas (Unicamp)
Caixa Postal 6101 – 13.083-970 – Campinas – SP – Brazil

[2]Federal University of São Carlos (UFSCar)
Campus Sorocaba – SP – Brazil

[3]Ericsson Research NomadicLab / HIIT
Helsinki, Finland

{chesteve, macapuna, mauricio}@dca.fee.unicamp.br

verdi@ufscar.br, andras.zahemszky@ericsson.com

Abstract. *This paper describes a networking approach for cloud data center architectures based on a novel use of in-packet Bloom filters to encode randomized network paths. In order to meet the scalability, performance, cost and control goals of cloud infrastructures, innovation is called for at many areas of the data center environment, including the underlying switching topology and the packet forwarding paradigms. Motivated by the advent of high-radix, low-cost, commodity switches coupled with a substrate of programmability, our proposal contributes to the body of work re-thinking how to interconnect racks of commodity PCs at large. In this work, we present the design principles and the OpenFlow-based testbed implementation of a data center architecture governed by Rack Managers, which are responsible to transparently provide the networking and support functions to cost-efficiently operate the DC network. We evaluate the proposal in terms of state requirements, our claims of false-positive-free forwarding, and the load balancing capabilities.*

1. Introduction

With the advent of Internet cloud services, the underpinning data center networks (DCN) have become a matter of intense research to raise their scale, performance, and cost-efficiency to unprecedented levels [Greenberg et al. 2009a]. In order to meet these goals without sacrificing service quality, innovation is called for at many areas of the data center environment, including the hosting infrastructure itself (e.g., energy management, wiring) and the network and system engineering (routing, virtualization, monitoring, etc.)

Recent research in re-architecting data center networks has spurred creative designs to interconnect servers at large, including shipping-container-tailored designs with servers acting as routers and switches as crossbars (BCube [Guo et al. 2009]), commoditized fat-tree topologies [Al-Fares et al. 2008], forwarding on position-based pseudo MAC addresses (Portland [Niranjan Mysore et al. 2009]), or load-balanced switching clouds providing the illusion of a single virtual layer 2 (VL2 [Greenberg et al. 2009b]). Traditional DCN architectures consist of a tree of networking elements (L2/L3 switches) with progressively more specialized and expensive equipment moving up the network hierarchy. Unfortunately, even when scaling up, resulting topologies may only offer a fraction of the aggregate capacity available at the end hosts, with reported over-subscription rates as high as 1:240 [Greenberg et al. 2009a].

While diverging in their architectural approach (e.g., server-centric vs. network-centric), every next generation DCN design proposal aims at providing a scalable, cost-efficient networking fabric to host Web, cloud and cluster applications. Many of these applications require bandwidth-intensive, one-to-one (e.g., video coding/streaming), one-to-several (e.g., distributed file systems), one-to-all (e.g., application data broadcasting), or all-to-all (e.g., MapReduce) communications among servers. Non-uniform bandwidth among data center nodes complicates application design and limits the overall system performance, turning the inter-node bisection bandwidth the main bottleneck in large-scale DCNs. Recent data center traffic characterization studies [Benson et al. 2009, S. Kandula and Patel 2009] have shed some light on the nature of DCN traffic, concluding that traffic demands are unpredictable and highly bursty, two factors that hamper traditional traffic engineering solutions (e.g., VLAN QoS). A closely related issue is the necessity of avoiding the fragmentation of resources (i.e., available servers and network paths) throughout IP subnets and VLAN domains. In highly virtualized cloud DCs, network *agility* is key to achieve high levels of server utilization and let virtual machines (VM) be dynamically instantiated (and live migrated) in any available physical server. An example of a job demanding agility might be to accommodate VMs on-demand to host Web services dedicated to the World Cup football championship during two months. In order to have an agile and unfragmented DCN, ideally, the underpinning interconnection fabric should behave like a big Ethernet domain that exploits the path diversity and scales sub-linearly to the number of addressable endpoints. Unfortunately, the flat routing nature of Ethernet does not scale beyond certain boundaries due to the lack of aggregation capabilities, the constraints of MAC-based forwarding tables, and the ARP flooding.

In this paper, we present SiBF (Switching with in-packet Bloom filters), a DCN proposal motivated by the changes in networking driven by the advent of high-radix, low-cost, commodity switches coupled with a substrate of programmability (e.g., OpenFlow [McKeown et al. 2008]). Our design borrows characteristics from a few new generation DCN designs, for instance building upon proven interconnection topologies (e.g., Clos networks) and reliance on logically centralized controllers in spirit of 4D [Greenberg et al. 2005]. Compared to related work, our key difference is the forwarding approach based on an in-packet Bloom filter (iBF) expedited by what we call a new entity in the data center: the Rack Manager (RM). The RM follows a direct network control approach to transparently provide the networking functions (address resolution, route computation) and support services (topology discovery, monitoring, optimization) to unmodified (physical and virtual) servers behind Top-of-Rack (ToR) switches.

Forwarding in SiBF takes on the idea of moving network state to the packet headers in form of a compact, multicast-friendly source route representation amenable to low-cost, high performance networking gear [Jokela et al. 2009]. Basically, SiBF efficiently interconnects any pair of communicating nodes within the DCN by compactly representing the packet's source route into a Bloom filter carried in the Ethernet MAC fields. Design goals include conserving the IP semantics and yield a false-positive-free forwarding fabric by leveraging DC's topological properties and exploiting the multiple paths available. We address the issue of having a system with two mutually conflicting requirements: 1) flat (non-hierarchical) L2 addresses, and 2) aggregation. Our approach is to open another vector of the design space, namely potential efficiency penalties due to false positives resulting in some packets using unnecessary links. The proposed solution makes better use of the 96-bit space of source and destination MAC fields, avoiding thereby encapsulation and shim-header overheads, and at the same time, conserving the nice plug and play properties of the Ethernet MAC addressing. The iBF-based fine control over the path traveled by packets enables multiple load balancing schemes to avoid hot-spots, for instance, by bouncing off traffic flows to intermediate switches.

The rest of the paper is organized as follows. Section 2 introduces background information on the rationale behind rethinking DCN architectures and outlines highlights of related work. Section 3 presents the design principles adopted for our solution and describes the key functional blocks. In Section 4, we detail the prototype implementation and the testbed environment. Section 5 evaluates SiBF in terms of network state requirements, false positive performance, and load balancing capabilities. Finally, Section 6 concludes the paper and outlines the future work.

2. Background

Current efforts towards low-cost powerful computing facilities span from large-scale (geo)-distributed application programming, innovation in the DC infrastructure, and re-thinking how to interconnect commodity PCs at large. Our work is focused on the latter. In this section, we first introduce networking requirements of the cloud, and then provide a snapshot of two remarkable new generation DCN proposals. Finally, we present the Bloom filter data structure, which is at the heart of our proposed forwarding mechanism.

2.1. Networking requirements of cloud data centers

The existing DCN literature seems to agree that efficiently networking the cloud DC calls for re-thinking the underpinning architecture to meet a reviewed set of requirements, which we have summarized as follows:

Resource Pooling: Offering the illusion of infinite computing resources available on demand requires means for elastic computing and agile networking. Such degree of DCN agility is possible (i) if IP addresses can be assigned to any VM within any physical server, and (ii) if all network paths are enabled and load-balanced.

Scalability: Networking (dynamically) a large pool of location-independent IP addresses (i.e., in the order of millions of VMs) requires a large scale Ethernet forwarding approach. Unfortunately, ARP broadcasts, MAC forwarding table sizes, and spanning tree limitations place a practical limit on the size of the system.

Performance: Available bandwidth should be high and independent from the end-points' location, which requires congestion-free routing for any traffic matrix in addition to fault-tolerance (i.e., graceful degradation) to link and server instabilities.

Middlebox support: An ordered sequence of middlebox services (e.g., firewalls, WAN optimizers, load balancers) is commonly required to be (transparently) placed on the network paths of DCN traffic. Conventional solutions (e.g., SPT, VLAN, OSPF) turns the overall configuration into a costly and tedious operation, besides unnecessary resource and performance inefficiencies [Joseph et al. 2008].

2.2. Related work

There is a large body of work tackling the cloud DCN research issues resulting in a collection of customized architectural proposals. We briefly outline the essence of two proposals which have inspired parts of our design.

PortLand proposes a scalable Ethernet-like layer 2 routing and forwarding protocol for data centers with three-tiered hierarchical topologies [Niranjan Mysore et al. 2009]. The approach to overcome the scalability limitations of Ethernet is based on modifying the control plane of the network, leaving the switch hardware and end hosts untouched. The main idea behind PortLand is the locator/identifier split, where nodes are identified by their actual MAC (AMAC) address, and located by a pseudo MAC (PMAC) address, which encodes hierarchical location information in its structure. Mapping between the two addressing spaces is done by the edge switches after querying a central fabric manager, which is responsible for tracking each correspondence of IP to pseudo MAC address within the discovered topology. Edge switches perform AMAC-PMAC rewriting for outgoing and incoming traffic.

VL2 provides a scalable Virtual Layer 2 to empower huge data centers with uniform high capacity between servers, performance isolation, and Ethernet semantics [Greenberg et al. 2009b]. Building upon existing technologies, in order to support agility, VL2 uses flat addresses in the IP layer to separate names from locators. VL2 yields uniform high capacity and traffic fairness by virtue of Valiant Load Balancing to randomize the traffic throughout the 3-tiered switching fabric using IP-in-IP encapsulation and Equal Cost Multi-Path (ECMP). Address resolution (i.e., application IP to location IP) is done modifying the end-systems and querying a scalable directory service.

2.3. Bloom filters

The Bloom filter is a popular data structure capable of answering questions of the form "is element x in set S?", with some tunable probability of returning false positives, i.e., claiming that x belongs to S even when this is not true. A typical implementation consists of a bit array of size m and k independent hash functions used to set/check bit positions when inserting/querying elements, which in our case are going to be switch MAC addresses forming a source route. The probability of false positives after inserting n elements is commonly approximated as (cf. [Bose et al. 2008]):

$$p^k = \left[1 - \left(1 - \frac{1}{m}\right)^{k*n}\right]^k \tag{1}$$

3. Design

The data center, as an interconnection network to perform distributed processing tasks, has three key dominant elements that determine its performance: (1) the network architecture (i.e. naming, address resolution, etc.), (2) the routing scheme, and (3) the interconnection topology. In this section we describe the design principles adopted to address (1) and (2) which can be summarized as an identifier/locator separated approach where IP addresses act solely as identifiers and oblivious routing is provided by forwarding based on in-packet Bloom filters (iBF) encoding randomly selected routes between the communicating endpoints. As for (3), the interconnection topology, in line with the existing literature, we assume a 3-tier topology with a lower layer of ToR switches, an intermediate layer of p_1-port Aggregation (AGGR) switches, and an upper layer of p_2-port CORE switches (see Fig. 1). Our solution is not restricted to a particular topology, and works on e.g., 3-level fat-trees with identical p-port switches (like Portland) or 3-tier 5-stage Clos arrangements (with $p_1 \neq p_2$ like VL2). Moreover, we note that other scale-out topologies could be considered (e.g., DHT-like rings, Hypercubes, Torus, etc.), as long as they offer large path diversity and low diameter.

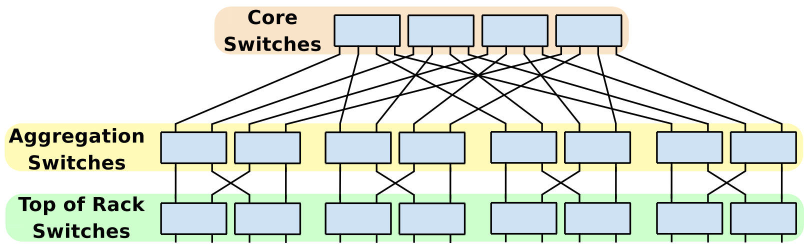


Figure 1. A 3-tier fat tree using 4-port switches.

3.1. Design Principles

We adopted the following principles for the proposed data center network architecture:

Separating Names from Locations: Identifier-locator split is the fundamental capability to enable resource pooling of IP addressable services, which can expand or contract their footprint in the DC as required (*agility*). IP addresses are used to identify physical servers (and VMs) within the DC. That is, no topological constraints are imposed on how IP addresses are assigned or translated in case of communications towards external networks (i.e., public Internet). In this context, IP addresses are not meaningful for packet routing, which is solely based on a revisited source-routing capable Ethernet layer.

Source explicit routing with zero-overhead: Leveraging the small diameter of data center topologies, our approach to meet the scalability goals is based on *strict source routing*. Routing in 3-tier DCN topologies is fairly simple, as any route between two ToRs, has an upward phase towards a common CORE switch and then a downward path to one AGGR switch connected to the destination ToR. Forwarding is based on an iBF containing only three elements, namely the Bloomed MAC identifiers of $\langle CORE_i, AGGR_{down}, ToR_{dst}\rangle$ switches. Source ToRs encode the iBFs in the MAC fields of outgoing packets which are sent to a next hop $AGGR_{up}$ switch. Hence, three iBF-based forwarding decisions are taken, one at the first AGGR, one at the CORE and one at the down-path AGGR. The destination ToR needs to re-write the source and destination MAC fields before delivering the packet to the destination server. By carrying the iBF in the 96 bit space of the MAC fields and re-writing packets at ToRs, we avoid encapsulation techniques or additional shim headers. Source routing not only minimizes FIB requirements of intermediate switches but also eases the inclusion of middleboxes.

Direct network control and logically centralized directory: SiBF embraces the 4D [Greenberg et al. 2005] philosophy of simplifying the data plane and centralizing the control plane to enforce the data center goals. We introduce the role of a Rack Manager (RM) to take the routing decisions and program the state of programmable switches. In order to construct source routes, two pre-requisites are required: (1) *topology information*, and (2) *server location*. We surmise that a directory service to track host locations and the underlying switching topology are implementable and able to scale to the envisioned DCN demands as shown by related work (e.g., Tesseract, Bing's Autopilot, VL2).

Load Balancing through path randomization: The approach to provide load balancing is based on *oblivious routing*, i.e., traffic independent randomized packet routing [Yuan et al. 2007]. More specifically, like VL2, we implement valiant load balancing (VLB) using the routing iBFs to "bounce off" flows at random intermediate switches.

Unmodified endpoints and plug-and-play: The forwarding fabric does not rely on end-host modifications. Legacy servers, operating systems and applications are supported off-the-shelf. Moreover, the plug-and-play behaviour of Ethernet is to be conserved, with auto-configuration of end-hosts and switches being part of the solution.

3.2. False-positive-free forwarding on Bloomed MAC identifiers

The key innovation comes when "switching" in the AGGR and CORE layers. Forwarding tables of switches are initially empty and get filled with one flow entry per neighboring switch detected. A flow entry is wildcarded except for the 96 bits of the source and destination MAC fields. However, instead of traditional exact matching of MAC fields, each flow entry contains a 96-bit mask generated from k hashes of the neighbouring switch unique MAC address. Similar to Link IDs in [Jokela et al. 2009], a *Bloomed MAC ID* is a 96-bit vector where only k bits set, but with the key difference that it is *not* directional, i.e., generated on a network interface pair basis. Forwarding decisions are trivial. On packet arrival, only the k 1s of each Bloomed MAC ID are checked for presence in the Ethernet MAC fields carrying the iBF. Upon match, the packet is forwarded.

Generation of Bloomed MAC identifiers: Instead of making k independent hashes of the neighboring switch MAC address, we make only one hash using a cryptographic function (e.g., MD5) and concatenate the output with the least significant 24-bits of the MAC address (unique per Ethernet vendor). Thereby, we obtain a randomly generated $128 + 24$ bit vector, which we slice in 7-bit segments to obtain k "pseudo" hashes that determine the bit positions in the 96-bit Bloom filter:

$$iBF[i] = (MAC_{24:48}|MD5(MAC))[7i : 7(i+1)]mod96 \tag{2}$$

Bloomed MACs IDs generated this way are still statistically unique e.g., $m!/(m-k)! \approx 10^{13}$ for $m = 96$ and $k = 10$. The algorithm defined by Eq. 2 is a system wide parameter that can be changed or optimized for a given set of MAC addresses (if required).

False positives: The well-known caveat of Bloom filters is the possibility of returning false positives to set membership queries, i.e., returning true when a set element was not inserted. In our case, this means that in addition to the explicitly inserted next hop switch, additional switch(es) appear(s) as next hop candidate(s). The resulting conflict can be solved either (i) by multi-casting the packet along all matching interfaces, or (ii) by picking only one. In any case, we require *iBF forwarding completeness*, i.e., loop-free and guaranteed delivery of packets to the intended destination(s). After a careful analysis of every false positive case of our implementation choice, we claim to have a loop-free, high solution that circumvents any potential issue arising from false positives. The factors that contribute to this result are multi-fold, some of them are due to an iBF-forwarding design tailored for multi-rooted tree topologies, and the remaining are implementation-specific, i.e. forced/enabled by our OpenFlow implementation choice. To start with, note that due to the high bit per element ratio ($m/n \approx 30$ for $m = 96$ and $n = 3$), false positives are extremely rare i.e. in the order of 10^{-7} (see details in Sec. 5.2).

Our strategy to avoid the potential effects of false positives is to exploit the notion of *power of choices* along two dimensions: (1) multiple paths, and (2) multiple iBF representations. That is, we compute the iBFs of the multiple available paths, and for each we generate d additional candidates using different sets of hash functions. Using the topology information, the routing service can easily check *a priori* whether any candidate iBFs is prone to false positives along the path. If so, those candidate iBFs are discarded from the random path selection. For the sake of brevity, we omit some details of the OpenFlow implementation and the analysis of why some false positives are self-healed by virtue of the multi-rooted topology. In a nutshell, false positives result in multiple flow entries matching the wildcarded k bits in the iBF. Since only the actions associated to one entry can be executed (per OpenFlow specification), a packet may be wrongly forwarded to a switch not included in the source route. Such packets lacking of matching flow entries are forwarded to the RM, which computes an alternative path and installs the required flow entries to temporarily fix the issue. However, recall that our strategy to avoid false positives is to detect them *prior* to their use. With knowledge of the topology, the RM pre-computes and maintains a source to destination ToR matrix filled only with false-positive-free iBFs for the multiple available network paths. In Sec. 5.3 we experimentally quantify the penalties on path multiplicity, which we anticipate to be insignificant.

3.3. Tree and Role Discovery Protocol

Topology knowledge is a prerequisite to allow source routing. A point which is not so evident and trivial is how to correctly infer the tree topology and the role of each switch (i.e., ToR, CORE or AGGR.), more critically at bootstrap time, since one of our requirements is to mimic the Ethernet plug & play behavior to avoid any manual intervention. This feature does not only reduce operational efforts to e.g., replace misbehaving switches, but is critical for the correct (and optimized) routing of packets. To this end, we have designed a Role Discovery Protocol (see Algorithm 1) that automates the inference of the switching tree by simply extending the link layer discovery protocol (LLDP) with an extension TLV to include the discovered role. We note that Portland faced a similar challenge in order to switches discovering their specific location within the DCN hierarchy to form a pseudo MAC address of the form *pod.position.port.vmid*. Our protocol is fairly simpler and requires only to identify the layer in which it is located.

4. Prototype implementation and testbed

Implementation of the iBF-forwarding mechanism is based on OpenFlow switches [McKeown et al. 2008], and the Rack Manager (RM) has been implemented as an application on top of the NOX controller [Gude et al. 2008]. In the following, we describe the key issues of the implementation work and the testbed environment. For details on the prototype implementation and on how to replicate our testbed we refer to

Algorithm 1: Role Discovery Protocol.

```
begin switch_join
    ROLE ← UNDEFINED;
    SendAllPorts(llpd, ROLE);
end

begin arp_receive_server
    if $ROLE\ != TOR$ then
        ROLE ← TOR;
    end
end

begin lldp_receive_neighbors
    NBROLE ← neighbors.ROLE;
    if $NBROLE = (CORE$ or $TOR)$ then
        ROLE ← AGGR;
    else if $NBROLE = AGGR$ then
        ROLE ← CORE;
    end
end
```

the publicly available source files and how-to instructions.[1]

4.1. OpenFlow

An OpenFlow (OF) switch separates the fast packet forwarding (data path) from the high level routing decisions (control path) of a router or switch. While the data path portion still resides on the switch and runs using the same underlying hardware, high-level packet handling decisions (i.e. routing) are moved to a separate controller. OF-enabled devices and the controller(s) communicate via the OF protocol, which defines messages, such as `packet-received`, `send-packet-out`, `modify-forwarding-table`, and `get-stats`.

The disruptive aspect of OF is to define a clean interface in form of a flow table abstraction with entries containing a set of packet fields to match from the 10-tuple: $\langle inport, Eth_{src}, Eth_{dst}, VLAN, EthType, IP_{proto}, IP_{src}, IP_{dst}, TCP_{src}, TCP_{dst}\rangle$, and a list of hardware-supported actions, i.e., `send-out-port`, `modify-field`, or `drop`. When an OF switch receives a packet for which it has no matching flow entry, it sends this packet to the controller, which in turn decides on how to handle the packet. The decision is sent to the switch, which can be instructed to cache the decision for some period of time by adding a flow entry to handle upcoming packets at line rate.

In order to support the iBF-based forwarding, only a minor modification was required to the current OpenFlow reference implementations (v.0.89rev2 and v.1.0). The key of iBF-based forwarding is the Bloomed MAC identifier which is a wildcarded *bit-mask* with only k arbitrary bits set to one. Thus, we needed to add this special behavior support to the OpenFlow datapath implementation. Fortunately, this required only changes in two lines of code[2] of the fast path flow matching function.

[1]To be published in http://www.dca.fee.unicamp.br/ chesteve/dcn/

[2]function `flow_fields_match` in `openflow1.0.0/udatapath/switchflow.c` or `openflow0.9.0/datapath/flow.c`

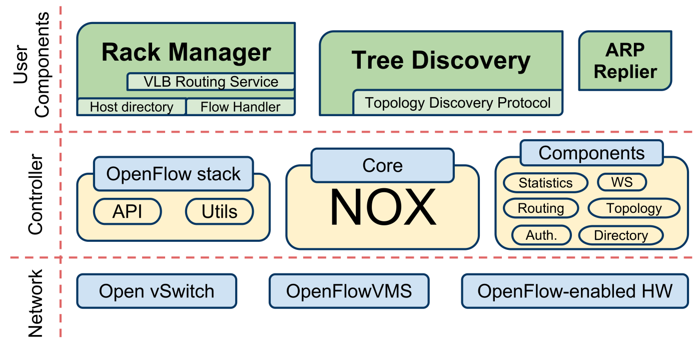


Figure 2. Component Architecture.

4.2. Rack Manager

The Rack Manager acts as a controller of OF switches, and as such, the natural implementation is as an application on top of the open source OF controller named NOX [Gude et al. 2008]. In a nutshell, NOX's programmatic interface is built upon *events*, triggered by NOX core components, thrown by user-defined applications, and generated directly from OF messages like `packet-in`, `switch join`, `switch leave`, etc. Figure 2 depicts our implementation of the RM functionality, which we have divided into three separate NOX user components.

4.3. Message sequence

The packet flow diagram of Fig. 3 shows how communications happen in the prototype implementation. Regular arrows are single data packets and the dotted arrows represent OF protocol messages. A server's network activity starts by sending an ARP request to some destination IP$_x$ (Step 0), for instance, to resolve the address of the DNS server. The ARP request reaches the ToR, which has no matching entry and informs the controller.

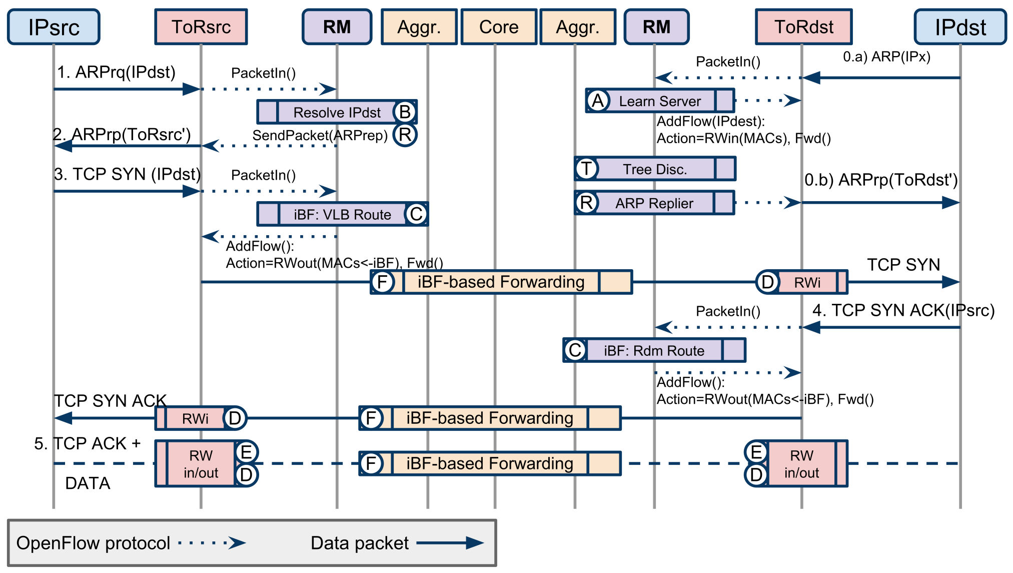


Figure 3. Packet flow sequence in an OpenFlow-based SiBF instantiation.

The *packet-in* event is passed to all the modules which have expressed interest in ARP packets. The RM learns the server location and registers it as being attached to a port of the OF switch triggering the event (Action A). It then sends a *flow-mod* command to install a semi-permanent flow entry for future incoming packets containing the destination IP equal to the server IP, and the associated actions set to (i) re-write the MAC fields with the ToR and server original MAC addresses, and (ii) forward to the attached port. The Tree Discovery identifies the switch as a ToR and updates the state of the Role Discovery Protocol (Action T). The ARP replier responds with a "fake" ARP reply containing the ToR MAC (Action R). In Step 1, a server sends an ARP request for a destination IP, and like any originating ARP, it acts as a trigger for the server discovery actions described in Step 0. After receiving the ARP reply (Step 2), the source node sends a TCP SYN packet which hits the ToR switch and is accordingly forwarded to the controller (Step 3). The RM picks one iBF towards the destination ToR (Action C), and orders the installation of an OF entry (10 sec. soft-expiration) to re-write packets belonging to the fully specified 10-tuple flow. Packets within this flow description get the iBF written in the MAC fields and are forwarded at line rate across the AGGR and CORE layers based on the iBF source route (Action F). When the iBF-labeled packet hits the destination ToR, it matches the flow entry installed in Step 0 (Action A) and is delivered to the destination server after re-writing the MAC headers (Action D). In Step 4, the destination server replies with a TCP SYN ACK which lacks of a flow entry and is delivered to the RM (Action C). After iBF selection and the installation of the flow entry (Action C), the TCP SYN ACK is forwarded based on the iBF. Upon reception at the originating server, the 3-way handshake can be completed (Step 5) and both entities can exchange data at line rate.

4.4. Testbed

The testbed consists of 5 physical nodes, one hosting the NOX controller with the RM components and the remaining 4 were partitioned into 9 virtual machines: 5 instantiating an OF switch each, and 4 hosting linux-based VMs. Figure 4 shows the testbed environment, where the solid lines represent direct links between virtual machines and the dashed lines represent the connections between VMs from different physical machines. The topology on each physical machine is configured with OpenFlowVMS, which in-

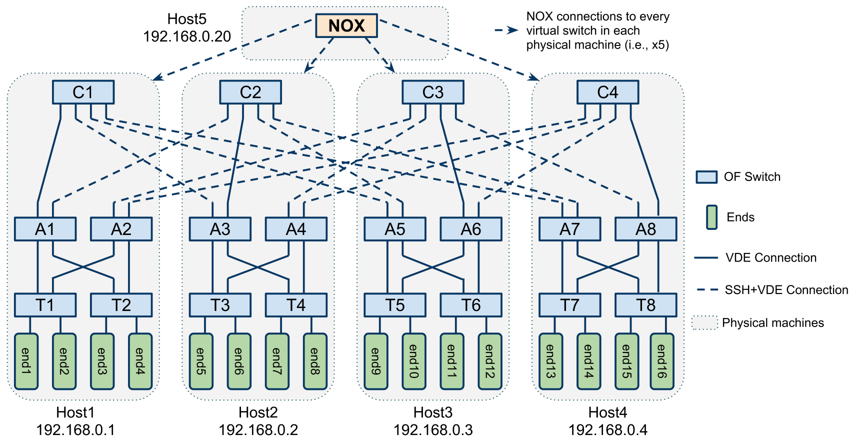


Figure 4. Testbed environment.

Table 1. Evaluation of the state requirements in terms of entries at switches.

Physical hosts	2.880			23.040			103.608		
Racks	144			1152			5184		
Aggr. Switches	24 ($p_1 = 24$)			96 ($p_1 = 48$)			144 ($p_1 = 144$)		
Core Switches	12 ($p_2 = 24$)			24 ($p_2 = 96$)			72 ($p_2 = 144$)		
	VL2	Portland	SiBF	VL2	Portland	SiBF	VL2	Portland	SiBF
Entries at ToR	200	120	120	1292	120	120	5420	120	120
Entries at AGGR	180	24	24	1272	48	48	5400	144	144
Entries at CORE	180	24	24	1272	96	96	5400	144	144

cludes a useful set of scripts to automate the creation of networked VMs using QEMU and VDE. Additional scripts were developed to distribute the environment across different physical machines using *ssh* connections and virtual dumb-switches based on VDE. Our extended script set enables to quickly define a target topology and automate the bootstrapping of the virtual nodes and OF switches, including the IP configuration, the creation of data-paths, the start-up of OF modules and the connection to the controller.

5. Evaluation

After validating the prototype implementation by verifying the full connectivity among the pool of servers (16 VMs), the next question is to evaluate the iBF-based forwarding fabric in terms of (i) state requirements, (ii) potential effects of false positives, and (iii) the load balancing capabilities. Due to the limitations of a virtualized testbed, performance aspects like goodput and flow completion times are left out of scope here.

5.1. State analysis

We start by comparing analytically the state requirements of SiBF with VL2 and Portland. The network setup is a 3-tier Clos topology, with ToRs connecting to 20 servers via 1 Gbps ports and to two AGGRs via 10 Gbps links. The p_1 ports of AGGRs are used to connect to $p_1/2$ ToRs and $p_1/2$ COREs equipped with p_2 high speed ports. In line with related work [Tavakoli et al. 2009], we assume an average of 10 concurrent flows per server (5 in and 5 out). Table 1 presents the scalability requirements for different switch configurations. By virtue of strict source routing, SiBF requires minimal state at COREs and AGGRs, namely only one entry per interfacing neighbor. Moreover, scaling-out the DCN does not impact the number of flow entries in the switches which is constant and equal to the number of neighbors. At ToRs, the amount of flow entries grows with the number of concurrent outgoing flows plus a constant amount of entries, one for each hosted server in order to re-write terminating flows. By comparison, VL2 requires forwarding entries in proportion to the total number of switches in order to route packets along the two-levels of IP encapsulation: $\langle LA_{CORE}, LA_{ToR} \rangle$. On the other hand, Portland has the same state requirements as SiBF, namely only one forwarding entry per interface, sufficient to perform the hierarchical forwarding on PMACs.

5.2. False positives

Now, we turn our attention to the practical false positive performance of small 96-bit Bloom filters when holding only 3 elements, namely the three Bloomed MAC addresses. More than the theoretical estimates (i.e., Eq. 1), what practitioners are really interested is in the observed false positive rate (fpr) after the iBF is queried for elements. Therefore,

Table 2. Evaluation of the false positive rate of the 96-bit iBF.

k	5	6	7	8	9	10	11	13	15	17	19	21
Theor. Eq 1 ($\cdot 10^{-6}$)	64.89	25.7	11.68	5.95	3.33	2.03	1.32	0.68	0.42	0.31	0.25	0.23
fpr ($\cdot 10^{-4}$)	2.41	1.81	1.5	1.7	1.83	2.23	3.09	4.92	7.17	11.46	16.09	21.07
fpr_{min} ($\cdot 10^{-6}$)	0.93	0.58	1.74	1.85	2.78	5.56	9.72	28.6	95.1	182	355	591

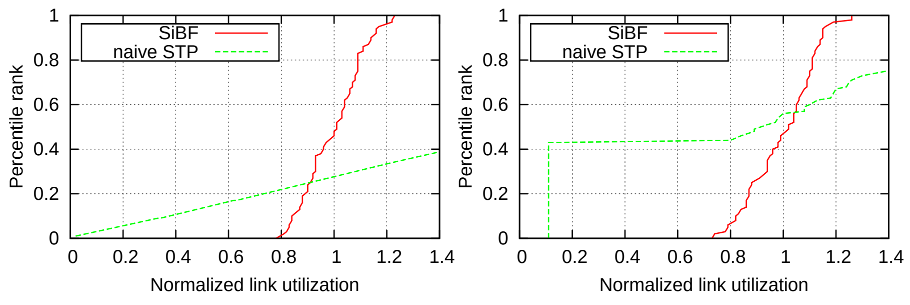
from a pool of 1M unique, randomly generated 48-bit values, on each experiment round (10.000 in total) we randomly insert 3 of them into a 96-bit BF using the Bloom MAC ID algorithm (Eq. 2) and test for presence of 432 (= 144 * 3 hops) randomly selected MACs. Table 2 shows the observed fpr for basic BFs constructs and when the power of choice optimization (with $d = 4$, $m' = 94$) is used (fpr_{min}). In theory, the optimal number of hash functions ($k_{opt} = \frac{m}{n} ln2$) that minimizes the false positive probability would be as many as 22. However, in our practical setup, the lowest fpr was obtained for k around 7. The deviation from the theoretical estimates can be explained by the accurate equation and bounds for small size Bloom filters by Bose et al. [Bose et al. 2008, Theorem 3]. Even without the d-candidate extension, only a few false positives per 10.000 queries were observed in plain 96-bit iBFs, which suggests that the effect of a false positive (if any) could be easily handled on a per-case basis.

5.3. False-positive-free forwarding on large-scale DCN topologies

We now evaluate the viability and efficiency of our false positive avoidance strategy based on discarding false-positive-prone iBF candidates prior to their use. Our thesis is that, given the low fpr of the 96-bit iBF data structure, there are plenty of false-positive-free paths between any two communicating nodes. In this experiment, we use an ns-3 implementation to explore the fpr performance on large-scale DCN topologies by sending an iBF for each of the available path between every ToR. Following the approach described in Sec. 5.1, we generate a topology with 48-port AGGRs and COREs to interconnect 576 ToRs, enough to host 11.520 physical servers. Testing every combination of $\langle ToR_{src} - ToR_{dst} \rangle$ (i.e., 331.200 ToR pairs) along each available path, results in over 30M iBFs sent and accounted for false positives. The summary results are as follows: 74% of the ToR pairs were false-positive-free for every available shortest path. Among those with some false positive (26%), the average number was 3 out of the 96 multiple paths. The maximum number of false positive paths for any ToR combination was 10. As a result, only 0.92% of all network paths exhibited some false positive and should be kept out of the pool of iBFs used for load balanced routing. Based on these results, we may conclude that false-positive-free forwarding comes at an affordable cost (less than 1%) in reduced path multiplicity. Moreover, considering the d-candidate optimization with e.g., $d = 4$, we could, with very high probability, get rid of the remaining 1% of false-positive iBFs by choosing alternative bit representations, and thereby utilize every available path.

5.4. Load balancing capabilities

Now, we investigate the load balancing capabilities of implementing VLB with iBFs over our testbed environment. Given a traffic matrix (TM), the goal is to evaluate how well the traffic is spread among the available links. We compare the link utilization of our VLB implementation with a vanilla Spanning Tree (SPT) implementation over the same topology. Two types of TMs were tested, one to mimic the all-to-all characteristics of DC applications like MapReduce, and one with random communicating endpoints. We used ITG [Avallone et al. 2004] as the traffic generator configured with TCP flows to last for 10s, with exponentially distributed payload sizes around 850 Bytes, which are reasonable assumptions for the majority of the reported DCN traffic. Figure 5 shows the normalized link utilization after ten experiment runs. As expected, SPT under- and over-utilizes the network links, whereas SiBF spreads traffic remarkably well, with the maximum and minimum normalized utilization of any link deviating only around 20% from the ideal value, i.e., 1. In the case of randomly chosen endpoints (Fig. 5(b)), the conclusion is the same, VLB using iBFs achieves a nice utilization of the available links in a TM-independent manner. The distribution of the normalized link utilization is comparable to the numbers reported in the VLB implementation of VL2, with min values (0.78 vs. 0.46) and max values (1.23 vs 1.2) [Greenberg et al. 2009b, Fig. 15]. The divergence of the min values can be explained by the nature of the operational traffic in VL2 compared to our synthetic TMs.



(a) All to all. (b) Random.

Figure 5. Evaluation of the load balancing behaviour. CDFs of the link utilization.

6. Conclusion

We have presented SiBF, a data center network architecture based on a simple data plane layer below IP that forwards packets based on the contents of an in-packet Bloom filter. SiBF embraces the (upcoming) category of commodity switches leveraged with a flow-oriented API extending the next frontier in data center networks from “commoditization” to “customization.” The DCN proposal presents many appealing characteristics such as not requiring any modification of end-hosts, reusing the Ethernet packet header bit space, minimal FiB consumption, and a fine control over the packet routes across the data center. The evaluation on a small-scale virtualized testbed implementation not only provides a proof of concept that helped to feedback the design cycles, but also shed light on the actual capacity of providing load balancing with randomized iBFs. In future implementation rounds, the prototype will be improved (e.g., to handle failure cases) and extended with additional features like distributed database management (e.g., topology and host directory) and transparent middlebox traversal, making it all together a real candidate to be deployed as an in-house cloud DCN playground.

Acknowledgements

This work is partly funded by CNPq, Capes, FAPESP and Ericsson Research.

References

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM CCR*, 38(4):63–74.

Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-itg distributed internet traffic generator. In *QEST '04*. IEEE Computer Society.

Benson, T., Anand, A., Akella, A., and Zhang, M. (2009). Understanding data center traffic characteristics. In *WREN '09*. ACM.

Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M., and Tang, Y. (2008). On the false-positive rate of Bloom filters. *Information Processing Letters*, 108(4):210–213.

Greenberg, A., Hamilton, J., Maltz, D. A., and Patel, P. (2009a). The cost of a cloud: research problems in data center networks. *SIGCOMM CCR*, 39(1).

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009b). VL2: a scalable and flexible data center network. *SIGCOMM CCR*, 39(4):51–62.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5):41–54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *SIGCOMM CCR*, 38(3).

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09*. ACM.

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. (2009). LIPSIN: line speed publish/subscribe inter-networking. In *SIGCOMM '09*. ACM.

Joseph, D. A., Tavakoli, A., and Stoica, I. (2008). A policy-aware switching layer for data centers. *SIGCOMM CCR*, 38(4):51–62.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74.

Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*. ACM.

S. Kandula, Sudipta Sengupta, A. G. and Patel, P. (2009). The nature of data center traffic: Measurements and analysis. In *ACM SIGCOMM IMC*.

Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. (2009). Applying NOX to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*.

Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2007). Oblivious routing for fat-tree based system area networks with uncertain traffic demands. *SIGMETRICS PER*, 35(1).

Appendix F

Publication F

C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi and M. F. Magalhães. “The Deletable Bloom Filter: A new member of the Bloom family.” In *IEEE Communication Letters*, June 2010, Volume: 14 Issue:6, On page(s): 557 - 559

The Deletable Bloom Filter: A New Member of the Bloom Family

Christian Esteve Rothenberg, Carlos A. B. Macapuna, Fábio L. Verdi, and Maurício F. Magalhães

***Abstract*—We introduce the Deletable Bloom filter (DlBF) as a new spin on the popular data structure based on compactly encoding the information of where collisions happen when inserting elements. The DlBF design enables false-negative-free deletions at a fraction of the cost in memory consumption, which turns to be appealing for certain probabilistic filter applications.**

***Index Terms*—Bloom filter, deletions, packet forwarding.**

I. Introduction

The Bloom filter (BF) [1] is a popular data structure capable of answering questions of the form "is element x in set S?", with some tunable probability of returning false positives, i.e., claiming that x belongs to S even when this is not true. Due to its simplicity and wide applicability, BFs have become very interesting objects of study and a daily aid of system implementations. The 40-year-old hash-based data structure is beloved by theoreticians due to the mathematics that underpin the randomized flipping of 0s into 1s, and is beloved by practitioners as a powerful ally when aggregating data sets. BFs turn resource-intense (memory, computation) operations into simple, resource-friendly set membership problems. The Bloom domain spans from hardware implementations, all the road up the system stack to the software application domain, where it first saw the light to perform space- and time-efficient dictionary look-ups.

The design of BFs is fundamentally about tradeoffs, i.e., striking the right balance between memory, computation and (false positive) performance. Several variations have been proposed to modify the behavior of the standard Bloom filter (SBF) beyond its natural limits, for instance, sacrifying its zero false negative characteristic in favor of less false positives, e.g., [5]. Due to its broad scope of applications, such metamorphoses are commonly needed to meet application-specific requirements or render additional features like frequency queries, deletions, coding values, security, and so on.

In this letter, we give birth to a new BF spin-off: The Deletable Bloom filter (DlBF). The DlBF inherits the plainness of its progenitor and introduces only a simple yet powerful idea, namely keeping record of the bit regions where collisions happen. The proposed design tradeoff turns out to be useful for applications with the following requirements:

R1:	Probabilistic guarantees of element deletability.
R2:	No false negatives upon element deletion.
R3:	Fixed memory allocation.
R4:	Low impact on the false positive rate (fpr) i.e. comparable to a SBF of the same bit size m.

Like other Bloom scions in the past, our needs for another Bloom variant come from a specific networking application (see in-packet BF examples in Sec. V). However, the DlBF is well suited for other use cases where re-constructing the filter upon set membership changes is either unfeasible or too costly. For standalone applications, removal of element fingerprints is commonly desirable for functionality or optimization purposes. For distributed applications, a deletable filter can be thinned out as queried elements are processed in order to (i) avoid repeated matches upfront, (ii) reduce false positives, and/or (iii) enable fresh bit space for new additions.

II. Related Work

The first Bloom descendant with genes for deletability is the Counting Bloom filter (CBF) [6], which basically extends the 1-bit cells to c-bit counters. Unfortunately, this c-fold reduction of practical bit space, typically 3 or 4 bits to avoid counter overflows, is a price too high in memory consumption (e.g., on-chip memory). Bloom relatives that improve this waste of space include the Spectral Bloom filter [4], and "an optimal Bloom filter replacement" [9]. While proven by theory to be more space-efficient, both alternatives come with a non-negligible complexity overhead, missing thereby the implicit requirement of *simplicity*, a critical factor for actual implementations. The d-left CBF (dlCBF) [2] is probably the best alternative construction for a CBF. Based on d-left hashing and element fingerprints, the dlCBF is simple, and given a target fpr, it requires about half the bit-space m of a 4-bit CBF. However, aiming at a fpr comparable to a SBF (R4), we can not afford around $2m$ for construction.

Closest to our design, is the Bloom filter with variable-length signatures (VLF) by Lu et al. [8], which presents an elegant solution to the deletion problem by resetting only a fraction of the k bits. Unfortunately, the main caveat of the VLF is that it is prone to false negatives, missing thereby R2. To the best of our knowledge, there is no Bloom filter variation which simultaneously satisfies all requirements R1-R4.

III. Design

The DlBF is built on the simple idea of tracking where bit collisions occur when inserting elements and exploits that bits set by only one element can be safely deleted. The proposed amendment consists of compactly encoding the regions of deletable bits using a fraction of the filter memory. An element can be effectively removed if at least one of its bits is reset,

Manuscript received March 1, 2010. The associate editor coordinating the review of this letter and approving it for publication was J. Holliday.

C. E. Rothenberg, C. A. B. Macapuna, and M. F. Magalhães are with the University of Campinas (Unicamp), Brazil (e-mail: {chesteve, macapuna, mauricio}@dca.fee.unicamp.br).

F. L. Verdi is with the Federal University of São Carlos (UFSCar), Brazil (e-mail: verdi@ufscar.br).

This work is partly funded by CNPq, Capes, FAPESP, and Ericsson Research.

Digital Object Identifier 10.1109/LCOMM.2010.06.100344

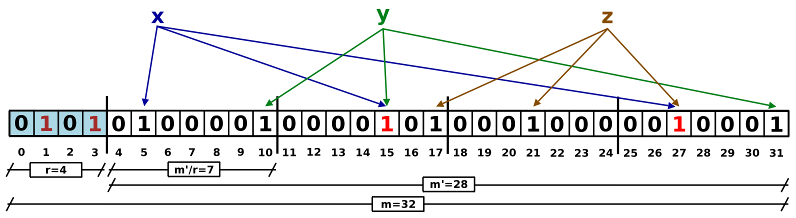


Fig. 1. Example of a DlBF with $m = 32$, $k = 3$ and $r = 4$, representing $S = \{x, y, z\}$. The 1s in the first r bits indicate collisions in the corresponding regions and bits therein cannot be deleted. All elements are deletable as each has at least one bit in a collision-free zone.

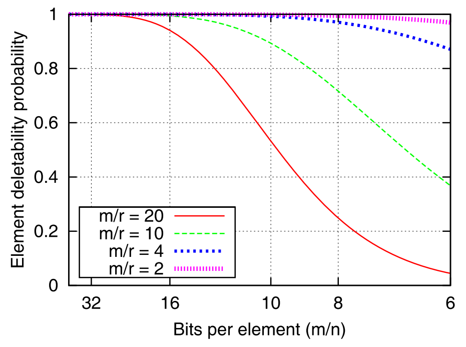


Fig. 2. Deletability estimate as function of the filter density m/n for different collision bitmap sizes r.

i.e., located in a collision-free-region. We divide a bit array of size m into r regions of $\lceil m'/r \rceil$ bits each, where m' is the original m minus the bits required to code the information of the collisions. A straightforward approach to compactly represent this information is a bitmap of size r to code with 0 a collision-free region and with 1 otherwise (see Fig. 1). Element insertion and lookup are the same as in a traditional BF. In addition to adding and maintaining a collision bitmap of size r, the DlBF adds an element removal primitive:

- *Insert(x)* maps an element x to k bit positions determined by a set of independent hashes. If one bit cell happens to be already set (collision), the corresponding region is marked in the r bitmap as non-deletable.
- *Query(x)* returns *true* if the k bit positions are set to 1.
- *Remove(x)* clears only those bit positions among k which are located in collision-free zones.

False-negatives are avoided at the cost of some elements not being deletable and accounting now as false positives, which are acceptable by the Bloom filter principle. Orphaned (non-removable) bits contribute to a larger fill factor, which, in turn, deviates the observed fpr from the expected value if all parameters were optimized. Consequently, one limitation of the DlBF appears in dynamic applications with frequent deletions and insertions where orphaned bits may fill the filter until collisions have happened in every region, hampering future deletions and increasing the residual fpr.

Since element removal is only probabilistic, a key design issue is choosing the value r and quantifying its impact on (i) the capacity to remove elements, and (ii) the false positive behavior (before and after elements are removed). First, we provide the mathematical model for the element deletability probability and then we estimate the false positive penalties.

A. Element deletability probability

Consider a bit array of size $m' = m - r$ with $\lceil m'/r \rceil$ bit cells per region. The probability that a given cell has at least one collision is $p_c = 1 - p_0 - p_1$, where p_0 denotes the probability that a given cell is set to 0 and p_1 is the probability that a given cell is set to 1 only once after inserting n elements:

$$p_0 = (1 - 1/m')^{kn} \text{and } p_1 = (kn)(1/m')(1 - 1/m')^{kn-1}$$

Then, the probability that a m'/r bit region is collision-free is given by $(1 - p_c)^{m'/r}$. Finally, for $r \geq k$ and $m >> k$, the probability of an element being deletable (i.e., with one of its k bits in a collision-free region) can be approximated to:

$$p_d = (1 - (1 - p_c)^{m'/r})^k \tag{1}$$

Figure 2 plots p_d against the filter density m/n for different memory to regions ratios m/r, confirming the intuition that increasing r results in a larger portion of deletable elements. As more elements are inserted (lower m/n), the number of collisions increase and consequently the deletion capabilities are reduced. Hence, the parameter r can be chosen by defining a target element deletion probability p_d and estimating the upper bound of the set cardinality n. For instance, allocating only 5 % of the available bits ($m/r = 20$) to code the collision bitmap, we can expect to remove around 90 % of the elements when the bits per element ratio m/n is around 16.

B. False positive probability

The false positive impact of consuming r bits from m can be estimated by updating m in the well-known false positive probability of a BF:

$$p_r{}^k = \left[1 - \left(1 - \frac{1}{m - r}\right)^{k*n}\right]^k \tag{2}$$

Obviously, the false positive degradation is driven by the ratio m/r. With r being only a fraction of m, the false positive increase is controllable and arguably comparable to a SBF, satisfying thus $R4$ ($fpr_{m'=m-r} \approx fpr_m$).

IV. Practical Evaluation

We now evaluate via simulation the practical performance of the DlBF in terms of *deletability* and observed fpr. We answer the questions of (1) how many elements can be safely removed in practice, and (2) how many false positives are observed before and after elements are removed.

Due to space limitations, we present only the experimental results for the case where $m = 240$ and $k = 5$, which corresponds to the configuration of the in-packet BF application [7] that motivated the DlBF design (see Sec. V). On every

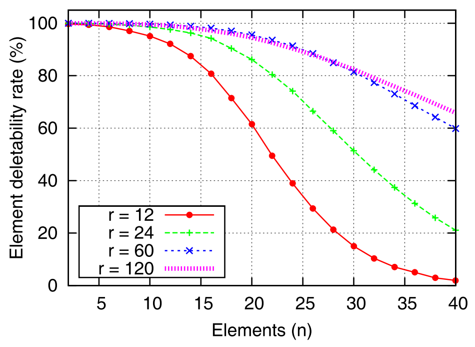


Fig. 3. Experimental deletability rate (mean values) of a 240-bit DlBF with $k = 5$ for different number of regions r.

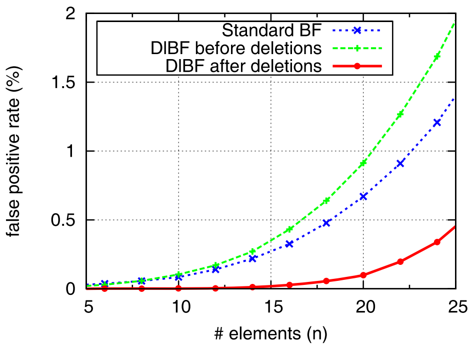


Fig. 4. Observed average fpr of a 240-bit DlBF with $r = 24$ and $k = 5$, before and after removing elements, compared to a 240-bit SBF.

trial (2000 per parameter set), we insert n elements randomly chosen from the American dictionary ($\approx$ 145K entries). We then (i) quantify how many inserted elements can be deleted (Fig. 3), and (ii) count for false positives (Fig. 4) by testing 500 randomly chosen elements (before and after deletions).

The observed deletability rate behaves as predicted by theory, but with relatively lower values (noteworthy as r tends to $m/2$ and for high m/n ratios). This can be explained by the assumption in Eq. 1 of perfectly random hash functions, an issue which can be more significant in small BFs [3]. Taking as an example the case where 10% of the memory is used to code the bitmap ($r = 24$), under a reasonably utilization ($n = 22$), on average, 80% of the elements could be removed (compared to 90% predicted by theory) by resetting around 40% of the bits (not shown in Fig. 3). Interestingly, doubling r from 60 to 120 only improves the number of deleted bits but not the actual element deletability. As expected, the price in fpr (Fig. 4) is an affordable increase before elements are removed, and a potential improvement when element bits are deleted. For other parameters (m, r, k), we could verify the adherence to theory, with the above noted divergences, too.

V. Example Applications and Future Work

We now give a snapshot on two networking applications to illustrate the practical use of the DlBF when placed into fixed-length packet headers. In LIPSIN [7], the inserted elements are unidirectional link identifiers (LID). A 256-bit source routing BF can be constructed by including the LIDs of a multicast delivery tree. Being able to remove already processed LIDs enables (1) avoiding loops, and (2) deleting special LIDs like multi-hop virtual links or control messages.

In a second scenario, we are exploring the DlBF in a data center environment to compactly represent a sequence of middlebox services (e.g., firewall, load balancer, DPI) which a packet needs to transverse. Relying on a substrate of switch programmability (OpenFlow), the content of the DlBF is used to transparently forward packets upon match on Bloomed Service IDs, which are removed after leaving the middlebox.

Future work includes exploring *dynamics* along two axes. First, understanding the practical limits if we keep doing insertions and deletions. Second, investigating a dynamic adaptation of the amount and the bit range of the deletable regions in function on how collisions happen. An open question is if there are other compact and more flexible ways to code the information of the collision-free regions. Finally, the *power of choices* at hashing time may introduce another interesting interplay. For instance, creating d DlBF candidates with different sets of hash functions and selecting the best in terms of fpr or guarantees that certain elements are deletable.

VI. Conclusion

This letter introduces the deletable Bloom filter (DlBF), a new Bloom engenderment based on the idea of compactly encoding the information of where collisions happen when inserting elements. This allows safely (i.e. without introducing false negatives) elements removal. Depending on how much memory space one is willing to invest, different rates on element deletability and false positives can be achieved. The DlBF is simple and can be easily plugged to existing BFs. We briefly presented two packet forwarding applications benefiting from the DlBF, which we believe could be a good fit for existing (and upcoming) friends of the Bloom principle.

References

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[2] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “An improved construction for counting bloom filters,” in *Proc. ESA’06*, pp. 684–695.
[3] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, “On the false-positive rate of Bloom filters,” *Inf. Process. Lett.*, vol. 108, no. 4, pp. 210–213, 2008.
[4] S. Cohen and Y. Matias, “Spectral bloom filters,” in *Proc. SIGMOD ’03*, pp. 241–252.
[5] B. Donnet, B. Baynat, and T. Friedman, “Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives,” in *Proc. CoNEXT ’06*, New York, NY, USA, 2006.
[6] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” *IEEE/ACM Trans. Networking*, vol. 8, no. 3, pp. 281–293, 2000.
[7] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: line speed publish/subscribe inter-networking,” in *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.
[8] Y. Lu, B. Prabhakar, and F. Bonomi, “Bloom filters: design innovations and novel applications,” in *43rd Annual Allerton Conference*, 2005.
[9] A. Pagh, R. Pagh, and S. S. Rao, “An optimal Bloom filter replacement,” in *Proc. SODA ’05*, Philadelphia, PA, USA, 2005, pp. 823–829.

Appendix G

Publication G

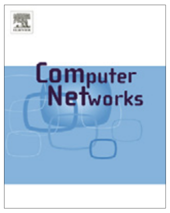
C. Esteve Rothenberg, C. A. Macapuna, F. L. Verdi, M. F. Magalhães and A. Wiesmaier. "In-packet Bloom filters: Design and networking applications.." In *Computer Networks*, In Press, Corrected Proof, Available online 19 December 2010, ISSN 1389-1286, DOI: 10.1016/j.comnet.2010.12.005.

Computer Networks xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



In-packet Bloom filters: Design and networking applications

Christian Esteve Rothenberg [a,*], Carlos Alberto Braz Macapuna [a], Maurício Ferreira Magalhães [a], Fábio Luciano Verdi [b], Alexander Wiesmaier [c]

[a] *University of Campinas (Unicamp), School of Electrical and Computer Engineering, Brazil*
[b] *Federal University of São Carlos (UFSCar), Campus Sorocaba, Brazil*
[c] *Technische Universität Darmstadt/CASED, Germany*

A R T I C L E   I N F O

Article history:
Received 19 January 2010
Received in revised form 11 October 2010
Accepted 13 December 2010
Available online xxxx
Responsible Editor: K. Kant

Keywords:
Bloom filter
Algorithms
Distributed systems
Packet forwarding
Inter-networking

A B S T R A C T

The Bloom filter (BF) is a well-known randomized data structure that answers set membership queries with some probability of false positives. In an attempt to solve many of the limitations of current network architectures, some recent proposals rely on including small BFs in packet headers for routing, security, accountability or other purposes that move application states into the packets themselves. In this paper, we consider the design of such in-packet Bloom filters (iBF). Our main contributions are exploring the design space and the evaluation of a series of extensions (1) to increase the practicality and performance of iBFs, (2) to enable false-negative-free element deletion, and (3) to provide security enhancements. In addition to the theoretical estimates, extensive simulations of the multiple design parameters and implementation alternatives validate the usefulness of the extensions, providing for enhanced and novel iBF networking applications.

1. Introduction

Since the seminal survey work by Broder and Mitzenmacher [1], the Bloom filter (BF) [2] has increasingly become a fundamental data aggregation component to address performance and scalability issues of very diverse network applications, including overlay networks [3], data-centric wireless networks [4], traffic monitoring, and so on. With the caveat of one-sided errors, the use of Bloom filters turns memory and computational expensive operations into simple, resource-friendly set membership problems (e.g. "*is* $x \in S$?").

In this work, we focus on the subset of distributed networking applications that use packet-header-size Bloom filters to share some state (i.e. information set S) among network nodes. The specific state carried in the Bloom filter varies from application to application, ranging from secure credentials [5,6] to IP prefixes [7] and link identifiers [8], with the shared requirement of a fixed-size packet header data structure to efficiently verify set memberships. The commonality of recent inter-networking proposals [5–10] is relying on Bloom filters to move application state to the packets themselves in order to alleviate system bottlenecks (e.g. IP multicast [7], source routing overhead [8]), enable new in-network applications (e.g. security [5,6,9]) or stateless protocol designs [11].

We refer to the BF used in this type of applications as an in-packet Bloom filter (iBF). In a way, an iBF follows a reverse approach compared to a traditional standalone BF implementation: iBFs can be issued, queried, and modified by multiple network entities at packet processing time. These specific needs benefit from additional capabilities like element removals or security enhancements. Moreover, careful design considerations are required to deal with the potential effects of false positives, as every packet header bit counts and the actual performance of the distributed system is a key goal.

* Corresponding author.
E-mail addresses: chesteve@dca.fee.unicamp.br (C.E. Rothenberg), macapuna@dca.fee.unicamp.br (C.A.B. Macapuna), magalhaes@dca.fee.unicamp.br (M.F. Magalhães), verdi@ufscar.br (F.L. Verdi), wiesmaier@cased.de (A. Wiesmaier).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.12.005

In this paper, we address common limitations of naive iBF designs and provide a practical foundation for networking application designs requiring to solve set-membership problems on a packet basis (Section 3). Our main contribution consists of assembling and evaluating a series of practical extensions (i) to increase the system *performance*, (ii) to enable false-negative-free *element deletion*, and (iii) to provide *security-enhanced* constructs at wire speed (Section 4). Via extensive simulation work, we explore the rich design space and provide a thorough evaluation of the observed trade-offs (Section 5). Finally, we relate our contributions to previous work on Bloom filter designs and briefly discuss the applicability of the iBF extensions to existing applications (Section 6).

2. Networking applications

iBFs are well suited for applications where one might like to include a list of elements in every packet, but a complete list requires too much space. In these situations, a hash-based lossy representation, like a BF, can dramatically reduce space, maintaining a fixed header size, at the cost of introducing false positives when answering set-membership queries. From its original higher layer applications such as dictionaries, BFs have spanned their application domain down to hardware implementations, becoming a daily aid in network applications (e.g., routing table lookups, DPI, etc.) and future information-oriented networking proposals [12]. As a motivation to our work and to get some practical examples of iBF usages, we first briefly survey a series of networking applications with the common theme of using small BFs carried in packets.

2.1. Data path security

The credential-based data path architecture [5] proposes the following network router security feature. During the connection establishment phase, routers authorize a new traffic flow request and issue a set of credentials (aka capabilities) compactly represented as bit positions of a BF. The flow initiator constructs the credentials by including all the router signatures into an iBF. Each router along the path checks on packet arrival for presence of its credentials, i.e., the k bits resulting from hashing the packet 5-tuple IP flow identifier and the routers (secret) identity. Hence, unauthorized traffic and flow security violations can be probabilistically avoided in a stateless, per hop fashion. Using 128 bits only, for typical Internet path lengths, the iBF-based authorization token reduces the probability that attack traffic reaches its destination to a fraction of a percent.

2.2. Wireless sensor networks

A typical attack by compromised sensor nodes consists of injecting large quantities of bogus sensing reports, which, if undetected, are forwarded to the data collector(s). The statistical en-route filtering approach [6] proposes a detection method based on an iBF representation of the report generation (collection of keyed message authentications), that is verified probabilistically and dropped en-route in case of incorrectness. The iBF-based solution uses 64 bits only and is able to filter out 70% of the injected bogus reports within 5 hops, and up to 90% within 10 hops along the paths to the data sink.

2.3. IP traceback

The packet-marking IP traceback method proposed in [9] relies on iBFs to trace an attack back to its approximate source by analyzing a single packet. On packet arrival, routers insert their mark (IP mask) into the iBF, enabling a receiver to reconstruct probabilistically the packet path(s) by testing for iBF presence of neighboring router addresses.

2.4. Loop prevention

In Icarus [10], a small iBF is initialized with 0s and then filled as forwarding elements add their Bloomed interface mask (setting k bits to 1). If the `OR` operation does not change the iBF, then the packet might be looping and should be dropped. If the Bloom filter changes, the packet is definitely not looping.

2.5. IP multicast

Revisiting the case of IP multicast, the authors of [7] propose inserting an iBF above the IP header to represent domain-level paths of multicast packets. After discovering the dissemination tree of a specific multicast group, the source border router searches its inter-domain routing table to find the prefixes of the group members. It then builds an 800-bit shim header by inserting the path labels (AS_a: AS_b) of the dissemination tree into the iBF. Routers receiving the iBF check for presence of next hop autonomous systems and forward the packet accordingly.

2.6. Source routing & multicast

The LIPSIN [8] forwarding fabric leverages the idea of having interface identifiers in BF-form (m-bit Link ID with only k bits set to 1). A routing iBF can be constructed by ORing the different Link IDs representing a source route. Forwarding nodes maintain a small Link ID table whose entries are checked for presence in the iBF to take the forwarding decision. In a typical WAN topology and using 256-bit iBFs, multicast trees with around 40 links can be constructed to reach up to 24 users while maintaining the false positive rate ($\approx$3%) and the resulting forwarding efficiency within reasonable performance levels.

3. Basic design

The basic notation of an iBF is equivalent to the standard BF, that is an array of length m, number of independent hash functions k, and inserted elements n. On insertion, the element is hashed to k hash values and the corresponding bit positions are set to 1 (see example in Fig. 1). On element check, if any of the bits determined by the hash outputs is 0, we can be sure that the element

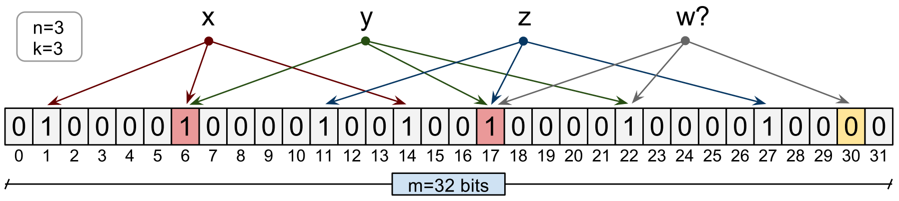


Fig. 1. Overview of the Bloom filter probabilistic data structure.

was not inserted (no false negative property). If all the k bits are set to 1, we have a probabilistic argument to believe that the element was actually inserted. The case of collisions to bits set by other elements causing a non-inserted element to return 'true' is referred to as a *false positive*. In the example of Fig. 1, a false positive for w would be returned if all three hashes would map to 1s.

For the sake of generality, we refer simply to *elements* as the objects carried in the iBF. Depending on the application, elements may take different forms such as interface names, IP addresses, certificates, and so on. False positives manifest themselves with different harmful effects such as bandwidth waste, security risks, computational overhead, etc. Thus, a system design goal is keeping false positives to a minimum.

3.1. False positive estimates

The *a priori* false positive estimate, *fpb*, is the expected false positive probability for a given set of parameters (m,n,k) *before* actually adding the elements. Let $p = 1 - (1 - 1/m)^{kn}$ be the probability that a particular bit is set to 1. Then,

$$fpb = (1 - (1 - 1/m)^{kn})^{k}. \tag{1}$$

The number k that minimizes the false positive probability can be obtained by setting the partial derivative of *fpb* with respect to k to 0. This is attained when $k = m/n \ln 2$, and is rounded to an integer to determine the optimal number of hash functions to be used [1].

While Eq. (1) has been extensively used and experimentally validated as a good approximation, for small values of m the actual false positive rate is larger. Recently, Bose et al. [13] have shown that *fpb* is actually only a lower bound, and a more accurate estimate can be obtained by formulating the problem as a balls-into-bins experiment:

$$p_{k,n,m} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {kn \atop i} \right\}. \tag{2}$$

According to [13, Theorem 4], Eq. (2) can be lower- and upper-bounded as follows:

$$p^k < p_{k,n,m} < p^k \cdot \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right). \tag{3}$$

Hence, the difference between the observed false positive rate and the theoretical estimates can be significant for small size BFs, a fact that we (and others) have empirically observed (see evaluation in Section 5.1.2). Thus, small iBFs are prone to more false positives than larger iBFs for equivalent m/n ratios.

Both Eqs. (1) and (2) do not involve knowing exactly how many bits are actually set to 1. A more accurate estimate can be given once we know the fill factor ρ; that is the *observed* fraction of bits that are actually set to 1 after elements have been inserted. We can define the posterior false positive estimate, *fpa*, as the expected false positive probability *after* inserting the elements:

$$fpa = \rho^k. \tag{4}$$

Finally, the *observed* false positive rate (*fpr*) can be obtained after testing for the presence of elements:

$$fpr = \frac{\text{Observed false positives}}{\text{Tested elements}}. \tag{5}$$

Note that the *fpr* is an experimental quantity obtained via simulation or system measurements and not a theoretical estimate. Hence, the *fpr* is the key performance indicator we want to measure in a real system, where every observed false positive will cause some form of degradation. Therefore, practitioners are less interested in the asymptotic bounds of the hash-based data structure and more concerned with the actual false positive rates, especially in the case of space-constrained iBFs.

3.2. Naming and basic operations

A nice property of hash-based data structures is that they do not depend on the form of the inserted elements. Independent of its size or representation, every element carried in the iBF contributes with at most k bits set to 1. In order to meet the line speed requirements of iBF operations, one design recommendation is to have the elements readily in a pre-computed BF-form (m-bit vector with k bits set to 1), avoiding thereby any hashing at packet processing time. Element *insertion* becomes a simple, parallelizable bitwise `OR` operation. Analogously, an iBF element *check* can be performed very efficiently in parallel via fast bitwise `AND` and `COMPARE` operations.

A BF-ready element name, also commonly referred to as element *footprint*, can be stored as an bit vector of size m or, for space efficiency, it can take a *sparse representation* including only the indexes of the k bit positions set to 1. In this case, each element entry requires only $k\log_2 m$ bits.

4. Extensions

In this section, we describe three useful extensions to basic in-packet Bloom filter designs in order to address the following practical issues:

(i) **Performance**: *Element Tags* exploit the notion of power of choices in combining hashing-based element names to select the best iBF according to some criteria, for instance, less false positives.
(ii) **Deletion**: *Deletable regions* introduce an additional header to code collision-free zones, enabling thereby safe (false-negative-free) element removals at an affordable packet header bit space.
(iii) **Security**: *Secure constructs* use packet-specific information and distributed time-based secrets to provide protection from iBF replay attacks and bit pattern analysis, preventing attackers from misusing iBFs or trying to infer the identities of the inserted elements.

4.1. Element tags

The concept of *element Tags* (eTags) is based on extending BF-compatible element naming with a set of equivalent footprint candidates. That is, instead of each element being identified with a single footprint, every element is associated with d alternative names, called eTags, uniformly computed by applying some system-wide mapping function (e.g., $k \cdot d$ hash functions). That allows us to construct iBFs that can be optimized in terms of the false positive rate and/or compliance with element-specific false positive avoidance strategies. Hence, for each element, there are d different eTags, where d is a system parameter that can vary depending on the application. As we see later, a practical value of d is in the range of multiples of 2 between 2 and 64.

We use the notion of *power of choices* [14] and take advantage of the random distribution of the bits set to 1 to select the iBF representation among the d candidates that leads to a better performance given a certain optimization goal (e.g., lower fill factor, avoidance of specific false positives). This way, we follow a similar approach to the Best-of-N method applied in [15], with the main differences of (1) a distributed application scenario where the value d is carried in the packet header, and (2) the best candidate selection criterion is not limited to the least amount of bits set but may include other optimization criteria (e.g., Section 5.2 bit deletability), including those that involve counting false positives against a training set (e.g. Section 4.1.2 *fpr*-based selection).

The caveats of this extension are, first, it requires more space to store element names, and second, the value d needs to be stored in the packet header as well, consuming bits that could be used for the iBF. However, knowing d at element query time is fundamental to avoid checking multiple element representations, which would traduce in potentially more false positives (cf. [14]). Upon packet arrival, the iBF and the corresponding eTag entries can be ANDed in parallel.

4.1.1. Generation of eTags

To achieve a near uniform distribution of 1s in the iBF, k independent hash functions per eTag are required. In general, k may be different for each eTag, allowing to adapt better to different fill factors and reducing the false positives of more sensitive elements. Using the *double hashing* technique [16] to compute the bits set to 1 in the d eTags, only two independent hash functions are required without any increase of the asymptotic false positive probability. That is, we rely on the result of Kirsch and Mitzenmacher [16] on linear combination of hash functions, where two independent hash functions and can be used to simulate i random hash functions of the form:

$$g_i(x) = [h_1(x) + i \cdot h_2(x)] \bmod m. \tag{6}$$

As long as $h_1(x)$ and $h_2(x)$ are system wide parameters, sharing $i = d \cdot k$ integers is only required to derive the eTags for any set of elements. For space efficiency, another optimization for the sparse representation of the candidates consists of defining the d eTags by combinations among $k + x$ iBF positions, i.e., $d = \binom{k+x}{k}$.

4.1.2. Candidate selection

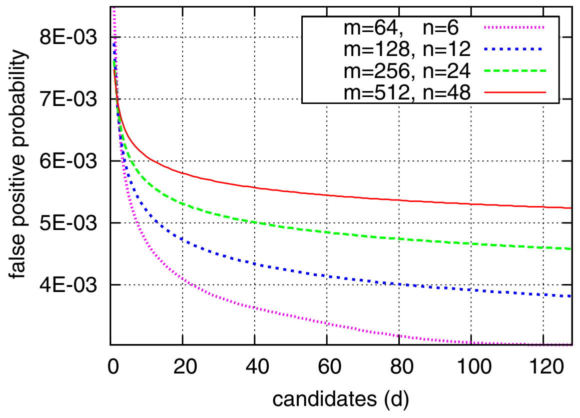
Having "equivalent" iBF candidates enables to define a selection criteria based on some design-specific objectives. To address *performance* by reducing false positives, we can select the candidate iBF that presents the best posterior false positive estimate (*fpa-based selection*; Eq. (4)). If a reference test set is available to count false positives, the iBF choice can be done based on the lowest observed rate (*fpr-based selection*; Eq. (5)). Other types of selection policies can be specified to favor the candidate presenting less false positives for certain "system-critical" elements (*fp-element avoidance selection*).

4.1.3. False positive improvement estimate

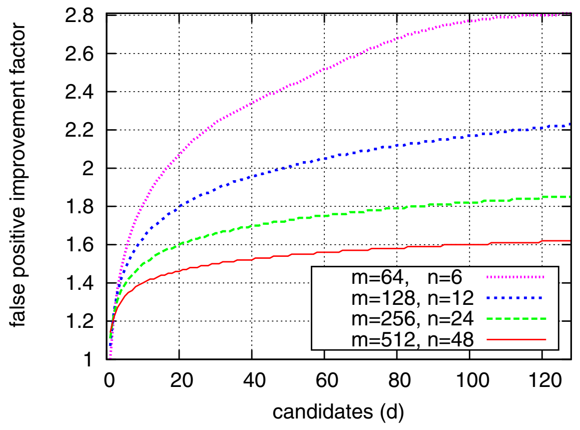
Following the same analysis as in [15], the potential gain in terms of false positive reduction due to selecting the iBF candidate with fewer 1s can be obtained by estimating the least number of bits set after d independent random variable experiments (see Appendix A for the mathematical details). Fig. 2 shows the expected gains when using the *fpa-based selection* after generating d candidate iBF for a given element set. With a few dozen candidates, one can expect a factor 2 improvement in the observed *fpr* when selecting the candidate with fewer ones. Note that the four iBF configurations plotted in Fig. 2 have the same m/n ratio. In line with the theoretical predictions [13], smaller bit vectors are subject to slightly larger false positive probabilities. However, as shown in Fig. 2(b), the *fpr* improvement factor of smaller iBFs due to the d-eTag extension is larger. Hence, especially for small iBFs, computing d candidates can highly improve the false positive behavior, a fact that we have validated experimentally in Section 5.

4.2. Deletable regions

Under some circumstances, a desirable property of iBFs is to enable element deletion as the iBF packet is processed along network nodes. For instance, this is the case if some inserted elements are to be processed only once (e.g., a hop within a source route), or, if bit space is required to add more elements upfront. Unfortunately, due to its compression nature, bit collisions hamper naive element removals unless we allow introducing false negatives into the



(a) A priori false positive estimate of the iBF candidate with the lowest fill factor.



(b) Potential false positive improvement when being able to choose among d iBFs.

Fig. 2. False positive probability gains of the power of choices extension.

system. To overcome this limitation (with high probability), so-called counting Bloom filters (CBF) [17] were proposed to expand each bit position to a cell of c bits. In a CBF, each bit vector cell acts as a counter, increased on element insertion and decreased on element removal. As long as there is no counter overflow, deletions are safe from false negatives. The main caveat is the c times larger space requirement, a prohibitive price for the tiny iBFs under consideration.

The key idea of the deletable region extension is to keep track of where the collisions occur at element insertion time. By using the property that bits set to 1 by just one element (collision-free bits) are safely deletable, the proposed extension consists of encoding the deletable regions as part of the iBF header. Then, an element can be effectively removed if at least one of its bits can be deleted.

Encoding the deletable region information should consume a minimum of bits from the allocated iBF space. A straightforward coding scheme is to divide the iBF bit vector into r regions of m'/r bits each, where m' is the original m minus the extension header bits. As shown in Fig. 3, this extension uses r bits to code with 0 a collision-free region and with 1 a non-deletable region. The probability of element deletion, i.e., the chances of an element having at least one bit in a collision-free region, can be approximated to (see Appendix B for the mathematical details):

$$p_d = (1 - (1 - p_c)^{m'/r})^k. \tag{7}$$

Fig. 4(a) plots p_d against the number of regions r and confirms the intuition that increasing r results in a larger proportion of elements being deletable. As more elements are inserted into the iBF, the number of collisions increases and the deletion capabilities (i.e., bits in collision-free regions) are reduced (see Fig. 4(b)). As a consequence, the target element deletion probability p_d and the number of regions r establish a practical limitation on the capacity n_{max} of a deletable iBF.

Fig. 4(b) plots p_d against the filter density m/n for different memory to regions ratios m/r. As expected, increasing r results in a larger portion of deletable elements. As more elements are inserted (lower m/n and more collisions), the deletion capabilities are reduced. Hence, the parameter r can be chosen by defining a target element deletion probability p_d and estimating the upper bound of the set cardinality n. For instance, allocating only 5 % of the available bits ($m/r = 20$) to code the collision bitmap, we can expect to remove around 90 % of the elements when the bits per element ratio m/n is around 16.

From a performance perspective, enabling deletions comes at the cost of r bits from the iBF bit space. However, removing already processed elements decreases the fill factor and consequently reduces the probability of false positives upfront. Later in Section 5.2 we explore the trade-offs between the overhead of coding the deletable regions, the impact on the *fpr*, and the implications of the candidate selection criteria.

4.3. Secure constructs

The hashing nature of iBFs provides some inherent security properties to obscure the identities of the inserted elements from an observer or attacker. However, there are

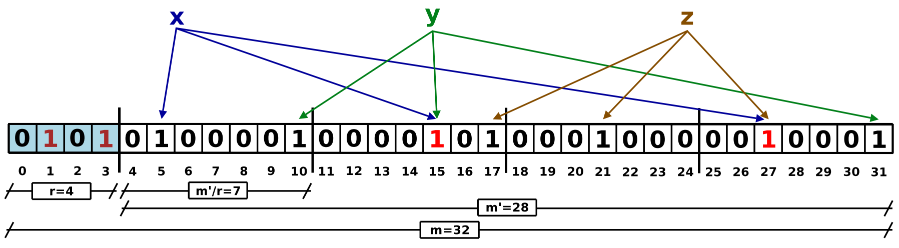
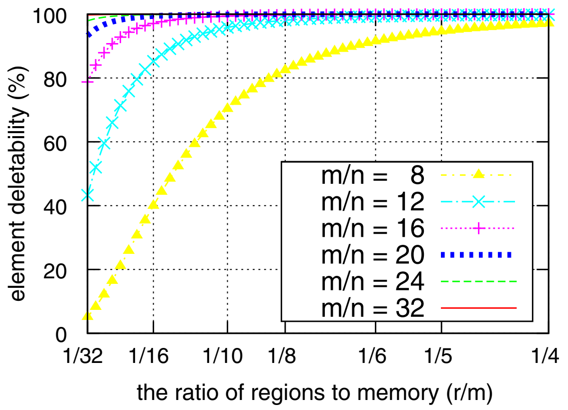
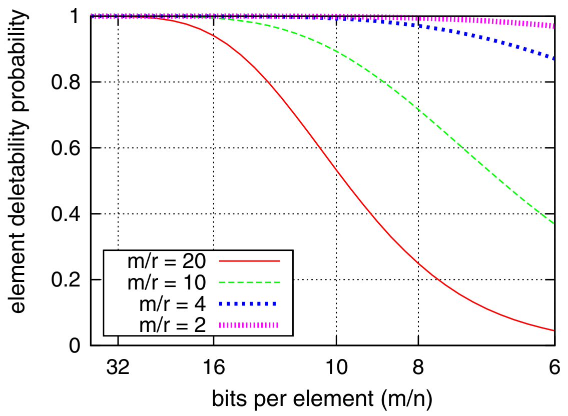


Fig. 3. An example of the DlBF with $m = 32$, $k = 3$ and $r = 4$, representing the set x,y,z. The 1s in the first r bits indicate that a collision happen in the corresponding region and bits therein cannot be deleted. Since each element has at least one bit in a collision-free zone, all of them are deletable.



(a) Deletability as function of number of regions.



(b) Deletability as function of inserted elements.

Fig. 4. Element deletability probability ($m = 256$).

a series of cases where improved security means are desirable. For instance, an attacker is able to infer, with some probability, whether two packets contain an overlapping set of elements by simply inspecting the bits set to 1 in the iBFs. In another case, an attacker may wait and collect a large sample of iBFs to infer some common patterns of the inserted elements. In any case, if the attacker has knowledge of the complete element space (and the eTags generation scheme), she can certainly try a dictionary attack by testing for presence of every element and obtain a probabilistic answer to what elements are carried in a given iBF. A similar problem has been studied in [18] to secure standalone BFs representing a summary of documents by using keyed hash functions. Our solution follows the same approach i.e. obscuring the resulting bit patterns in the filter by using additional inputs to the hashes. However, our attention is focused to the specifics of distributed, line-speed iBF operations.

The main idea to improve the security is to bind the iBF element insertion to (1) an invariant of the packet or flow (e.g., IP 5-tuple, packet payload, etc.), and (2) system-wide time-based secret keys. Basically, the inserted elements become packet- and time-specific. Hence, an iBF gets expirable and meaningful only if used with the specific packet (or authorized packet flow), avoiding the risk of an iBF replay attack, where the iBF is placed on a different packet.

4.3.1. Binding to packet contents

We strive to provide a lightweight, bit mixing function $O = F(K,I)$ to make an element name K dependent on additional in-packet information I. For this extension, an element name K is an m-bit hash of the element and not the eTag representation with only k bits set to 1. The function F must be fast enough to be done at packet processing time over the complete set of elements to be queried by a node processing the iBF. The output O is the k bit positions to be set/checked in the iBF. Using cryptographic hash functions (e.g., MD5, SHA1) for F becomes unpractical if we want to avoid multiple (one per element) cycle-intense hashing per packet.

Algorithm 1: Secure iBF element set/check algorithm

```
Input: element name $K$, packet-specific id. $I$, number
    of choices $d$
Output: $k$ bit positions to be set/checked
1. Let $O = k \otimes I$
2. Divide $O$ into $k$ segments of $m/k$ bits:
$O_1, O_2, \ldots, O_k$
3. Divide each $O_j$ into a $clog_2m$ bit matrix:
$Oj_1, Oj_2, \ldots, Oj_c$ where $c = \lceil \frac{m}{klog_2m} \rceil$
4. for each $O_j \in [O_1, \ldots, O_k]$ do
    Set/check bit position $i$ in the iBF where:
    $i = O_{j1} \otimes O_{j2} \otimes \cdots \otimes O_{jc}$
    $i \ll d$
end
```

As an example resource-efficient implementation of F, we propose the lightweight Algorithm 1 to mix each element K with a fixed bit string I. Taking I as an input, the algorithm runs in parallel on each element K and returns the k bit positions in the iBF to be set or checked. After an initial bitwise XOR operation (Step 1), the output O is divided into k segments of m/k bits (Step 2). To build the folding matrix in Step 3, each segment is transformed into a matrix of $\mathrm{c}\log_2 m$ bits.[1] For instance, with $m = 256$ and $k = 4$, each segment O_k would be a 64-bit bit vector transformed into a 8×8 matrix. Finally, each of the k output values is computed by XORing the rows of each matrix into a $\log_2 m$ bit value that returns the bit position to be set/checked (Step 4). The d-bit shifting enables the power of choices.

We are faced with the classic trade-off between security and performance. An heuristic evaluation suggests that the proposed F provides a good balance between performance and security. First, F involves only bit shifting and XOR operations that can be done in a few clock cycles in parallel for every K. Second, the k bit positions depend on all the bits, within an m/k bit segment, from the inputs I and K. The security of F depends on how well I and K are mixed. For security sensitive applications, the XOR operation in Step 1 should be replaced with a more secure transformation $P(K,I)$ i.e., using lightweight hash functions or line-speed stream ciphers (see e.g. [19]). The final choice of F

[1] Note that depending on the values of m and k, some padding bits (e.g., from within segments) may be required to complete the matrix.

should take the application specifics into account (e.g., nature of K, computation of I per-packet) and the target security level.

4.3.2. Time-based keyed hashing

A more elaborate security extension consists of using a *keyed* element name construction, and change the secret key $S(t)$ regularly. We can define $S(t)$ as the output of a pseudo random function $S_i = F(seed, t_i)$, where *seed* is the previous value and t a time-based input. Then, we can include the current S value in the algorithm for element check/insertion e.g., $O = F(K, I, S(t))$. Thereby, we obtain a periodically updated, shared secret between iBF issuers and iBF processing entities, with the benefit that an iBF cannot be re-utilized after a certain period of time or after an explicit re-keying request. Moreover, by accepting S_i and S_{i-1} the system requires only loose synchronization similar to commercial time-coupled token generators. At the cost of initial synchronization efforts and computational overhead, this method provides an effective means to make iBF applications secure (e.g., forwarding availability [20,21]).

4.4. Density factor

Finally, a basic security measure for iBFs, also proposed in [5], is to limit the percentage of 1s in the iBF to 50%–75%. A *density factor* ρ_{max} can safely be set to $k \cdot n_{max}/m$, as each legitimate element contributes with at most k bits. Then, the probability of an attacker guessing a bit combination that causes a single false positive can be upper bounded by ρ_{max}^k.

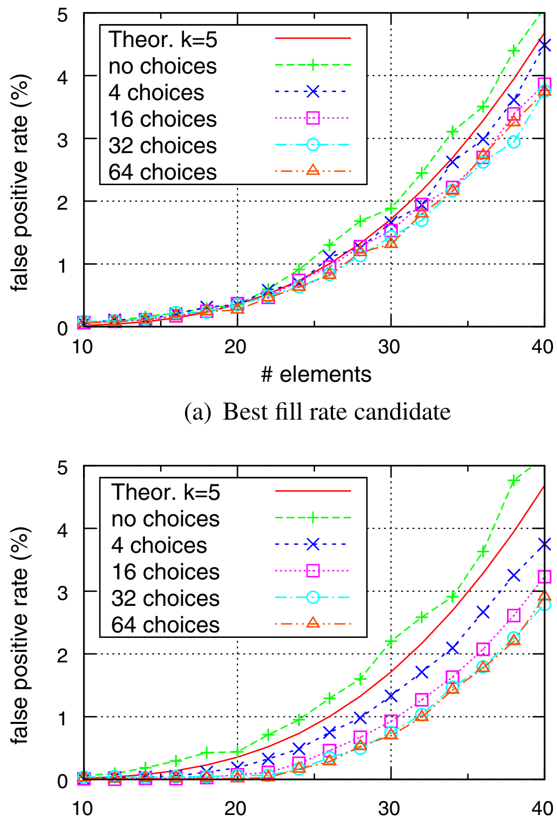
5. Practical evaluation

We now turn our attention to the practical behavior of the iBF in function of the multiple design parameters and carry out extensive simulation work to validate the usefulness of the three extensions under consideration. For these purposes, we use randomly generated bit strings as input elements and the double hashing technique using SHA1 and MD5. The section concludes exploring the potential impact of different types of iBF elements (flat labels, IP addresses, dictionary entries) and the hash function implementation choice.

5.1. Element tags

We are interested in evaluating the gains of the power of choices that underpins the element Tag extension (Section 4.1), where any element set can be equivalently represented by d different iBFs, different in their bit distribution but equivalent with regard to the carried element identities. We first explore the case where $k = 5$ and then the impact of using a distribution around 5 for candidate naming.[2]

[2] We choose $k = 5$ to have a probabilistically sufficient footprint space for the eTags $(m!/(m-k)! \approx 10^{12}$ with $m = 256)$ when targeting an m/n of about 8 bits per element.



(a) Best fill rate candidate

(b) Best observed fpr

Fig. 5. Power of choice gains ($m = 256$, $k = 5$).

5.1.1. Power of choices (d)

We run the simulations varying d from 2 to 64 and updating m accordingly to reflect the overhead of including the value d in the packet header. Fig. 5 compares the observed *fpr* for different values of d. In accordance with the theoretical predictions (Section 4.1.3), increasing d and choosing the candidate iBF just by observing its fill factor after construction (Fig. 5(a)) leads to better performing iBFs. In the region where the iBF is more filled (30–40 elements), the observed *fpr* drops between 30% and 50% when 16 or more candidate iBFs are available. Another interpretation is that for a maximal target *fpr* we can now insert more elements. As expected, the performance gain is more significant if we consider the best performing iBF after testing for false positives. Observing Fig. 5(b), the number of false positives is approximately halved when comparing the best iBF among 16 or more against a standard 256-bit iBF.

We also note that the observed *fpr* is slightly larger than the commonly assumed theoretical estimate (Eq. (1)), confirming thus the findings (Eq. (4)) by [13]. As shown in Table 1, this difference is more noticeable for smaller m, becoming negligible for m larger than 1024.

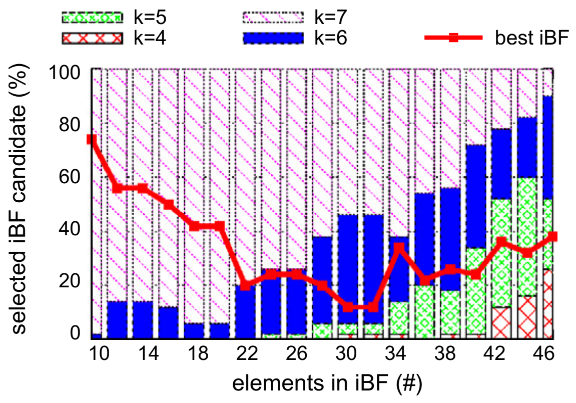
5.1.2. Distribution of the number of hash functions (k)

Now, we explore allowing a different number of bits k per candidate. For instance, with $d = 8$ the distribution of

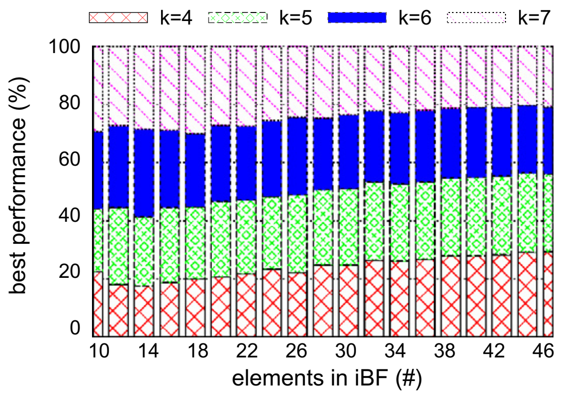
Table 1
Observed *fpr* for iBFs with 16 eTag choices.

m	n	Std. (%)		fpa-opt. (%)		fpr-opt. (%)	
		Th.	*fpr*	k_{cte}	k_{dst}	k_{cte}	k_{dst}
128	6	0.04	0.16	0.14	0.19	0.04	0.05
	12	0.75	1.12	0.88	0.86	0.37	0.32
	18	3.33	4.39	2.80	3.10	2.18	2.37
256	12	0.04	0.09	0.08	0.08	0.01	0.03
	24	0.74	0.95	0.74	0.71	0.26	0.30
	36	3.31	3.63	2.69	2.75	2.07	2.15
512	24	0.04	0.08	0.07	0.04	0.01	0.01
	48	0.74	0.83	0.64	0.64	0.22	0.25
	72	3.29	3.46	2.87	3.05	2.09	2.21

k among the candidates could be {4, 4, 5, 5, 6, 6, 7, 7}. Intuitively, this naming scheme adapts better to the final number of elements in the iBF (as k_d closer to $k_{opt} = m/n\ln(2)$). The *fpa-based selection* criterion (Section 4.1) is now choosing the candidate with the lowest estimate $min\{\rho_0^{k_0}, \ldots, \rho_d^{k_d}\}$. Fig. 6(a) shows the distribution of the selected 256-bit iBFs for the case of $d = 16$ and k evenly distributed between 4 and 7. The line shows the percentage of times that the selected iBF actually yielded the best performance among the candidates. Disregarding the scenarios with fewer elements, the *fpa-based selection* strategy succeeded to choose the optimal candidate in about 30% of the times.



(a) Best fill rate candidate



(b) Best observed fpr

Fig. 6. Distribution of iBF candidates for different number of hash functions k. ($d = 16$, $m = 256$).

Fig. 6(b) shows the percentile distribution of the best performing iBF after *fpr* testing. As expected, in more filled iBFs scenarios, setting less bits per element is beneficial. However, the differences are relatively small. As shown in Table 1, the observed *fpr* in the case of $k_{const.} = 5$ is practically equivalent (if not slightly better) to the case where k is distributed. We can also observe what the theory in Section 2 predicts with regard to smaller iBFs: (i) inferior *fpr* performance for the same m/n ratio, and (ii) larger potential to benefit from the power of choices extension.

5.1.3. Discussion

Based on our experimental evaluation, having more than 32 candidates per element is not compelling in terms of additional proportional *fpr* benefits beyond approximately a factor 2 depending on the specific parameters. The results are consistent with the theoretical estimates in Section 4.1.3. However, if the system design choice is based on selection criteria optimized for the non occurrence of specific false positives (i.e. element-avoidance Section 4.1), increasing the number of choices d allows complying to a larger set of false positive avoidance policies. The practical limitations would be how much space the application designer is willing to pay to store the candidate element representations in the nodes and code the index d in the packets.

5.2. Deletion

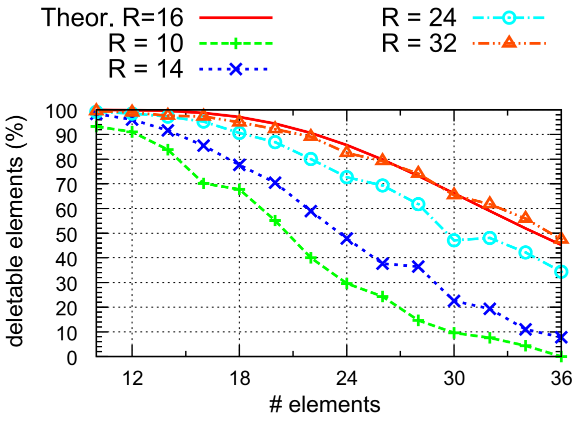
We explore two important aspects of the deletable regions extension. First, from a *qualitative* point of view we examine the actual capabilities to successfully delete elements for different m/n ratios, number of *regions* r and choices d. Second, we evaluate the *quantitative* gains in terms of false positive reduction after element bits are deleted. Obviously, both aspects are related and intertwined with the ability to choose among candidate iBF representations to favor the deletion capabilities. Now, the application can choose the iBF candidate with the most number of bits set in collision free-zones, increasing thus the *bit deletability*. Alternatively, one may want to favor the *element deletability*, recalling that removing a single element bit is traduced into a practical deletion of the element.

Using our basic coding scheme (Section 4.2), we consume one bit per region to code whether collision happened and deletion is prohibited or not. Thus, the bits available for iBF construction are reduced to $m' = m - \log_2 d - r$.

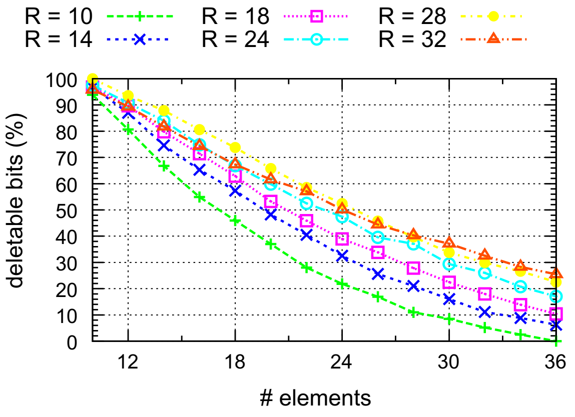
On each experiment round, we randomly select n elements from a pool of 1 million unique bit strings, and insert them updating the r bitmap accordingly. We then try to remove every inserted element and measure the quality and quantity of the deletion capabilities.

5.2.1. Quality: how many elements can be removed in practice?

Fig. 7(a) plots the average percentage of elements that could be deleted. As expected, partitioning the iBF into more regions results in a larger fraction of elements (and bits) being deletable. For instance, in the example of a 256-bit iBF with 32 regions (Fig. 7), when 24 elements are inserted, we are able to delete an average of more than 80% of the elements by safely removing around 50% of the bits (Fig. 7(b)). Playing with the candidate choices, we can enhance the bit (Fig. 8(a)) and element (Fig. 8(b)) deletability considerably. The actual deletability rates are lower than expected by theory (Fig. 4) but behave as predicted by the mathematical model of the element deletability probability (Eq. (7)). This divergence can be explained by the theoretical assumptions on random bit distributions

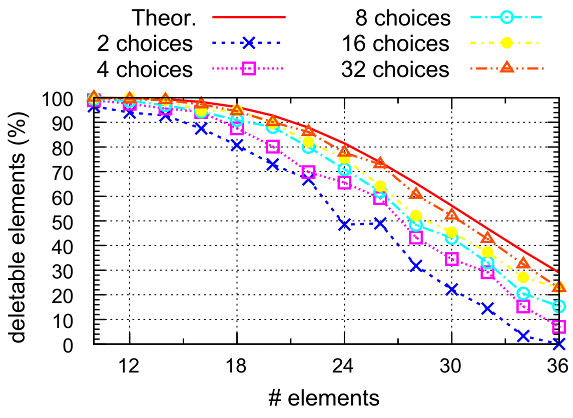


(a) Deletability - elements

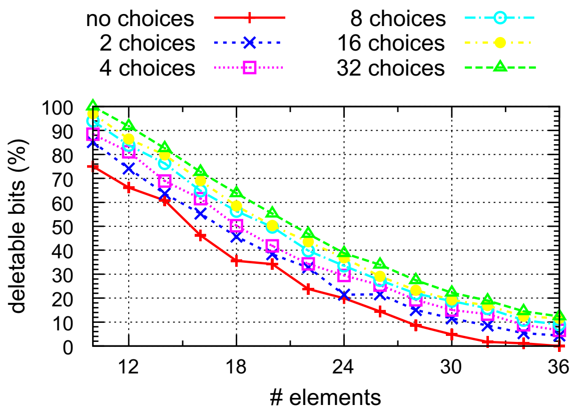


(b) Deletability - bits (d=16)

Fig. 7. Deletability as function of r ($m = 256$).



(a) Deletability - elements



(b) Deletability - bits

Fig. 8. Deletability as $f(d)$. $m = 256$, $r = 16$.

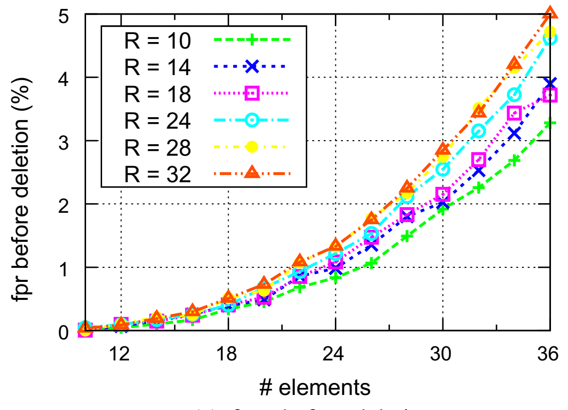
and the actual behaviour of hash functions, especially in small size bit vectors.

5.2.2. Quantity: what are the false positive rate gains due to bit deletability?

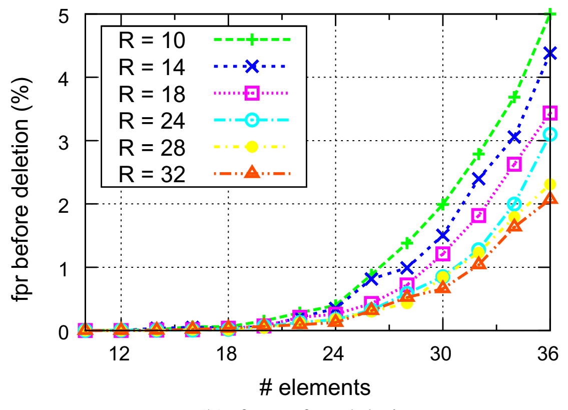
On the one hand, we have the potential gains of removing bits from collision-free zones. On the other, the cost of (1) coding the deletable regions (r bits), and (2) having more filled iBFs due to the rarefication of colliding bits. While Fig. 9(a) shows the price of having to code more regions (*fpr* before deleting elements), Fig. 9(b) illustrates the potential gains of removing every deletable bit. If we average the *fpr* before and after elements are deleted, the iBF performance appears equivalent to the *fpr* of a standard non-deletable m-bit iBF. In comparison, a counting BF with 2 bits per cell[3] would behave like an iBF of size $m/2$, which would have its element capacity prohibitively constrained.

Analyzing the impact of the *power of choices*, Fig. 10 shows that choosing the best deletable iBF candidate causes the colliding bits to "thin out" (greater ρ), yielding a higher *fpr* before deletion (Fig. 10(a)) and a smaller *fpr* after elements are removed (Fig. 10(b)).

[3] Using the power of choices, we could have with very high probability a candidate that does not exceed the counter value of 3, avoiding false negatives as long as no new additions are considered.

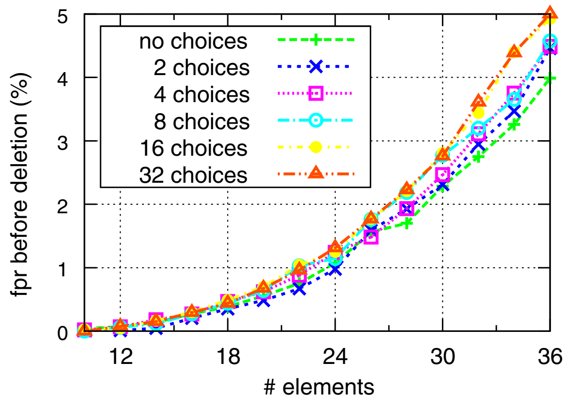


(a) fpr - before deletion

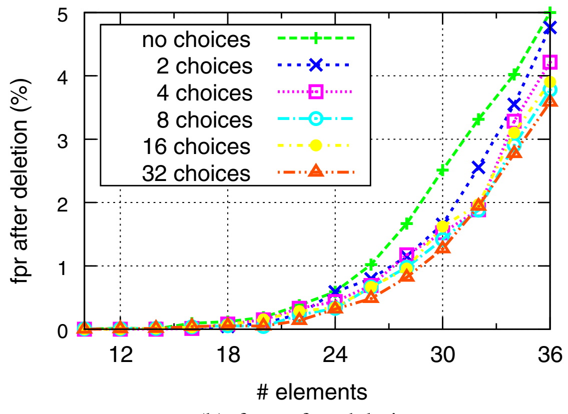


(b) fpr - after deletion

Fig. 9. False positives in function of r ($m = 256$, $d = 16$).



(a) fpr - before deletion



(b) fpr - after deletion

Fig. 10. False positives as $f(d)$. ($m = 256$, $r = 16$).

5.2.3. Discussion

There is a tussle between having a smaller fill factor ρ, with more collisions at construction time reducing the *fpa*, and the deletability extension that benefits from fewer collisions. Deletability may be a key property for some system designs, for instance, whenever an element in the iBF should be processed only once and then be removed, or when space is needed to add new elements on the fly. One key property is that just by inspecting the bitmap r, any node can safely (without introducing false negatives) remove an element from the iBF. A more detailed evaluation should consider the specific application dynamics into consideration, i.e., the nature and frequency of deletions/insertions at runtime.

From a *fpr* performance perspective, the cost of coding the deletable regions is only a slight increase in the *fpr* due to r being only a small fraction of m. However, reducing m seems to hinder the average *fpr* gains due to bit deletions upfront. Nonetheless, especially for space-restricted iBFs, the proposed extension is a far more attractive approach to enable (probabilistic) deletions than alternative solutions based on counting BFs. An open question is whether there is a better coding scheme for the deletable regions, for instance, using error correcting codes. Finally, the power of choices again proved to be a very handy technique to deal with the probabilistic nature of hash-based data structures, enabling candidate selection for different criteria like better *fpr* or certain element/bit deletability.

5.3. Security

Besides fast computation, the main requirements for the security extension are that (i) the random distribution of the iBF bits is conserved, and (ii) given a collection of packets I and the securely constructed iBFs, one cannot easily reveal information about the inserted elements (K). More generally, given a set of (I, *iBF*) pairs, it must be at best very expensive to retrieve information about the identities of K.

We first measured the randomness of the secure iBF construction outputs from Algorithm 1 by fixing a set of 20 elements and changing the per-packet 256-bit randomly generated I value on each experiment run. Table 2 gathers the average results of 100 experiments with 1000 runs per experiment. The observed distribution of outputs within an experiment, measured as the Hamming distance between output bit vectors (BV), was very close to the mean value of $m/2$ bits (128) with a small standard deviation.[4] The observed average number of bits set and their distribution were comparable to standard iBF constructs. Additionally, we analyzed whether the 20 most frequent bit positions set in secure iBFs corresponded to bits set in

[4] In future work we will extend these results and the hashing techniques evaluation of Section 5.4 with standard randomness tests such as those included in the Diehard suite (http://www.stat.fsu.edu/pub/diehard).

Table 2
Evaluation of the secure iBF algorithm ($m = 256$, $k = 4$, $n = 20$). Avg. (Stdev) after 1000 runs.

	Sec. iBF	Plain iBF	Random BV
Hamming dist.	127.94 (8.06)	0	127.95 (8.03)
# Bits set	96.27 (3.20)	96.29 (–)	127.97 (7.97)
Correlation	0.371		–

plain iBFs. We defined the *correlation factor* as the fraction of matches and obtained a value of 0,371, which is close to the probability of randomly guessing bits in a 256-bit iBF with $k = 4$ and $n = 20$ elements ($Pr \approx 96/256 \approx 0{,}37$).

The results indicate that, assuming a random packet identifier I, first, no actual patterns can be inferred from the securely inserted elements, and second, the random bit distribution of an iBF is conserved when using the proposed algorithm. However, we recognize the limitations of Algorithm 1. For instance, if provable protection against more elaborated attacks is required, then, a more secure and computationally expensive bit mixing procedure (Step 1 in Algorithm 1) should be considered, in addition to a time-based shared secret as suggested in Section 4.3.

5.4. Hashing technique

Finally, we investigate the impacts of the hash function implementation choice and the nature of the input elements in small size iBFs. There are two factors that determine the “quality” of the bit distribution and consequently may impact the observed *fpr*: (1) the *input* bit string, and (2) the *implementation* of the hash function.

5.4.1. Input data sets

Instead of considering elements as simple random bit strings, we now explore three types of elements that cover typical inputs of iBF applications:

- **32-bit IP addresses:** Nearly 9M IP addresses were generated by expanding the subnet values of IP prefixes advertised in the CAIDA database.[5] In addition, private IP addresses (10.0.0.0/16,192.168.0.0/16) were also used in the experiments.
- **256-bit random labels:** A set of 3M random labels was generated constructing each 256-bit label by picking randomly 64 hex characters and checking for uniqueness.
- **Variable-bit dictionary words:** A set formed by 98.568 entries of the American dictionary.[6]

5.4.2. Hash function choice

We chose 3 commonly used cryptographic hash functions (MD5, SHA1 and SHA256) and 2 general purpose hash functions (CRC32 and BOB).[7]

[5] ftp.ripe.net/ripe/stats/delegated-ripencc-20090308.

[6] /usr/share/dict/american-english.

[7] Related work has investigated the properties of 25 popular hash functions, pointing to BOB as a fast alternative that yields excellent randomized outputs for network applications [22]. Although MD5 and SHA1 are considered broken due to the recent discovery of collisions, they are perfectly valid for our randomness purposes.

Table 3
Observed *fpr* in 256-bit iBF using double hashing with SHA1 & MD5 and with 8-bit segments of CRC32. Avg. (StdDev); 1000 tests.

n	DoubleHash	IP	Random	Dict.
16	SHA1& MD5	0.340 (0.035)	0.338 (0.032)	0.328 (0.034)
	CRC32 segm.	0.345 (0.037)	0.349 (0.034)	0.338 (0.034)
32	SHA1& MD5	2.568 (0.436)	2.576 (0.449)	2.519 (0.385)
	CRC32 segm.	2.541 (0.418)	2.532 (0.403)	2.570 (0.444)

The observed *fpr* (Table 3) imply that, on average, the input type does not affect the iBF performance. Fig. 11 plots the observed normalized *sample variance* for different bit vector sizes (m). For lower m values the variances show a larger difference and start converging for $m > 512$. CRC presents the best output distribution when dealing with IP addresses as inputs. This may be explained by the 32-bit match of inputs and outputs. In general, the functions exhibit similar behavior, leading to the conclusion that all 5 hash functions can be used independently from the nature of the elements. This result experimentally confirms, also in the case of small m values, the observation by Mitzenmacher and Vadhan [23] that given a certain degree of randomness in the input, simple hash functions work well in practice.

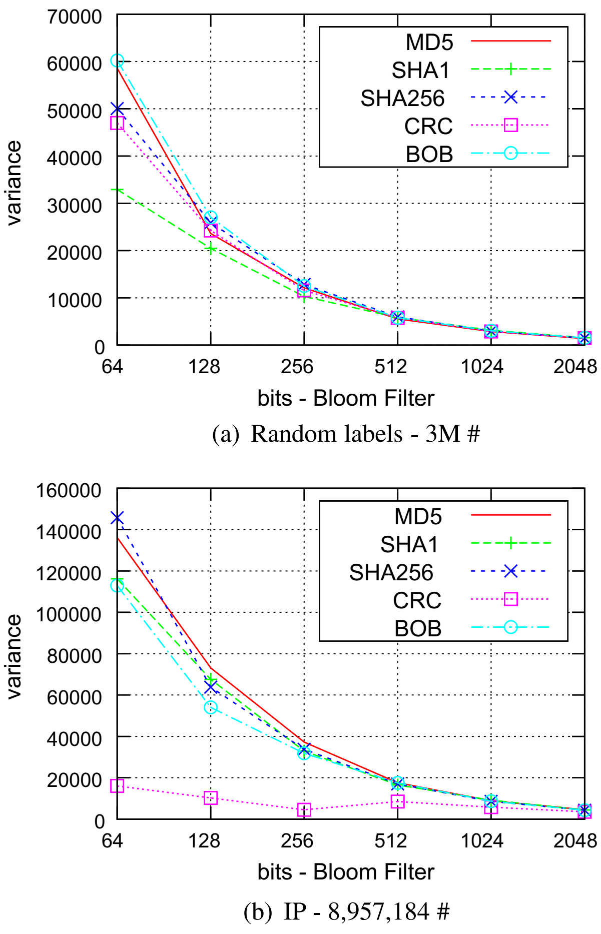


Fig. 11. Normalized bit distribution variance of hash outputs.

5.4.3. Hash segmentation technique

For the purposes of iBF construction, there is a waste of hash output bits due to the *mod m* residual restrictions. Hence, we want to know whether we can divide the output of a hash function into $\log_2 m$ segments and use each segment as an independent hash value. We compare the bit distribution and *fpr* performance of iBFs constructed using the double hashing technique with MD5 and SHA1 against iBFs generated with CRC32 segments as $h_i(x)$. The differences of the observed *fpr* (Table 3) are negligible, which suggests that this hashing technique may be practical. Hence, we can reduce the two independent hash function requirement of the double hashing technique to a single hash computation based on e.g., CRC32 or BOB. This result can be applied to iBF networking applications with on-line element hashing instead of pre-computed element names. Moreover, this efficient hash segmentation technique may be useful in other multiple-hashing-based data structures (e.g., d-left hash tables) that require hashing on a packet basis.

6. Related work

Although multiple variants of Bloom filter designs and applications have been proposed in the last years (e.g., *Bloomier, dynamic, spectral, adaptive, retouched*, etc.), to the best of our knowledge, none of the previous work focuses on the particular requirements of distributed networking applications using small Bloom filters in packet headers.

Prior work on improved Bloom filters include the Power of Two Choices filter [14] and the Partitioned Hashing [24], which rely on the power of choices at hashing time to improve the performance of BFs. False positives are reduced in [24] by a careful choice of the group of hash functions that are well-matched to the input elements. However, this scheme is not practical in distributed, highly dynamic environments. The main idea of [14] is to reduce the number of 1s by choosing the "best" set of hash functions. Besides our in-packet-header scope, our approach differs in that we include the information of which group of hash functions was used (d value) in the packet itself, avoiding thereby the caveat of checking multiple sets. On the other hand, we need to stick to one set of hash functions for all elements in the BF, whereas in [14] the optimal group of hash functions can be chosen on an element basis. To our benefit, due to the reduced bit vector scenario, we are able to select an optimal BF after evaluating all d candidates, which leads to improved performance even in very dense BF settings (small m/n ratios).

Regarding the extension to choose the best candidate filter, the Best-of-N method [15] only considers a standalone application where the best BF selection is based on the least dense filter constructed using the optimal number of hash functions. In contrast, our distributed iBF applications include candidates with different amount of bits set, as the maximum set cardinality may be unknown a priori. An optimized candidate iBF selection is possible whenever the iBF application is able to test for presence of elements that are known to be queried upfront. Moreover, selection criteria may be beyond reducing *fpr*, for instance benefiting the deletion of elements or avoiding specific false positives.

The closest BF design innovations to support deletions, other than counting BFs or d-left fingerprint hash tables [17], are the Variable-length Signatures (VBF) [25]. Similarly based on resetting at least one bit from element signatures, the main caveat is being prone to false negatives. In contrast, our deletable regions do not introduce false negatives at the cost of providing only probabilistic element deletions.

Security and privacy preserving extensions for standalone BFs have been previously proposed in different contexts (e.g., [18,26]). The novelty of our application resides in taking distributed systems and data packets specifics into consideration (e.g., flow-identifier, time-based loose synchronization of distributed secrets).

7. Relevance and extensions in practice

Our work on iBFs is mostly an outcome from our research on compact packet forwarding mechanisms. The idea of element Tags has its roots in the work on a link identifier based forwarding fabric [8], rendering the system more useful (network policy compliance, loop avoidance, security) and efficient (*fpr* control, larger multicast groups). Recently, we applied the notion of power of choices in the design of scalable data center forwarding services [27,28] based on 96-bit iBF encoding of valiant load balanced fat tree network paths between virtual machines hosted in rack servers.

In general, the element tag extensions can be applied to similar use cases of compact source routing. For instance, in the IP multicast proposal [7], having multiple choices would reduce false positives and enable compliance to inter-domain AS policies in the case of false positives. When applied to the credentials-based architecture proposed in [5], multiple candidates may allow iBFs to transverse larger paths before reaching the maximum density. Additionally, the security extension may provide extra protection from an en-route attacker spoofing the source IP address and re-using the flow credentials for unauthorized traffic.

In the field of secure packet forwarding mechanisms, we contributed to the development of self-routing capabilities for DDoS protection in Bloom filter based forwarding services [20]. In addition, recent work has validated the effectiveness of loop prevention with a new extension based on performing per-hop bit permutations [29]. A joint application of these iBF algorithmic techniques aims at solving forwarding anomalies of naive approaches to iBF based networking. Related work has explored the application of secure iBF forwarding methods to provide fast host mobility [30], scalable multicast VPN services in GMPLS-enabled networks [31], and an edge-controlled approach to stateless inter-domain 'bloomcasting' [21].

Last but not least, we are looking for new use cases for our deletable Bloom filter design [32]. Due to its space efficiency and the tunable probability of safe bit removals, the deletable regions extension may have interesting applications beyond the scope of iBFs. Similarly, the hash segmentation technique appears useful to lower the burden on any system requiring multiple expensive hash computations.

8. Conclusions

This paper explores an exciting front in the Bloom filter research space, namely the special category of small Bloom filters carried in packet headers. Using iBFs is an appealing approach for networking application designers choosing to move application state to the packets themselves. At the expense of some false positives, fixed-size iBFs are amenable to hardware and present a way for new networking applications.

We studied the design space of iBFs in depth and evaluated new ways to enrich iBF-based networking applications without sacrificing the Bloom filter simplicity. First, the power of choices extension shows to be a very powerful and handy technique to deal with the probabilistic nature of hash-based data structures, providing finer control over false positives and enabling compliance to system policies and design optimization goals. Second, the space-efficient element deletion technique provides an important (probabilistic) capability without the overhead of existing solutions like counting Bloom filters and avoiding the limitations of false-negative-prone alternatives. Third, security extensions were considered to couple iBFs to time and packet contents, providing a method to secure iBFs against tampering and replay attacks. Finally, we validated the extensions in a rich simulation set-up, including useful recommendations for efficient hashing implementations. We hope that this paper motivates the design of more iBF extensions and new networking applications.

Acknowledgments

We thank the reviewers of earlier versions of this paper. We are also thankful for the fruitful discussions with Mats Naslund, Pekka Nikander and Andras Zahemszky. The work presented in this paper is supported by Ericsson Research, CNPq and FAPESP.

Appendix A. Mathematical model for *d*-candidate *fpa* optimization (adapted from [15])

Given the iBF parameters m, n, k, and letting d be the number of different iBF candidates for the same element set, the probability of setting arbitrary but fix s bits in an iBF candidate can be formulated as an independent random variable experiment:

$$E^2[s] = m\left(1-\left(1-\frac{1}{m}\right)^{kn}\right) + m(m-1)\left(1-2\left(1-\frac{1}{m}\right)^{kn}+\left(1-\frac{2}{m}\right)^{kn}\right), \quad (A.1)$$

$$\sigma^2[s] = m\left(\frac{m-1}{m}\right)^{kn} + m^2\left(\frac{m-2}{m}\right)^{kn} + m\left(\frac{m-2}{m}\right)^{kn} + m^2\left(\frac{m-1}{m}\right)^{2kn}. \quad (A.2)$$

Defining $\mu = E[s]$ and $\sigma = \sigma[s]$, the minimum continuous probability density function is:

$$f_{min}(s) = \left(\frac{1}{2^{d-1}}\right) d\left(erfc\left(\frac{s-\mu}{\sigma\sqrt{2}}\right)\right)^{d-1}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(s-\mu)}{2\sigma^2}}\right). \quad (A.3)$$

Consequently, the expectation of the least number bits (s_{min}) set by any of d candidates:

$$E(S_{min}) = \int_{-\infty}^{\infty} s f_{min}(s) ds. \quad (A.4)$$

Finally, the probability of a false positive once the smallest fill ratio has been estimated:

$$pr[false\ positive] = \left(\frac{E[s_{min}]^k}{m}\right). \quad (A.5)$$

Appendix B. Element deletability probability

Consider a bit array of size $m' = m - r$ with $\lceil m'/r \rceil$ bit cells per region. The probability that a given cell has at least one collision is $p_c = 1 - p_0 - p_1$, where p_0 denotes the probability that a given cell is set to 0 and p_1 is the probability that a given cell is set to 1 only once after inserting n elements:

$$p_0 = (1 - 1/m')^{kn} \quad (B.1)$$

and

$$p_1 = (kn)(1/m')(1 - 1/m')^{kn-1}. \quad (B.2)$$

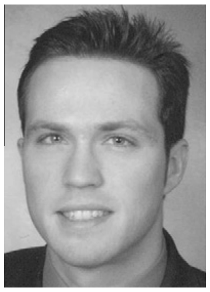
Then, the probability that a m'/r bit region is collision-free is given by $(1 - p_c)^{m'/r}$. Finally, for $r \geqslant k$ and $m \gg k$, the probability of an element being deletable (i.e., with one of its k bits in a collision-free region) can be approximated to:

$$p_d = (1 - (1 - p_c)^{m'/r})^k.$$

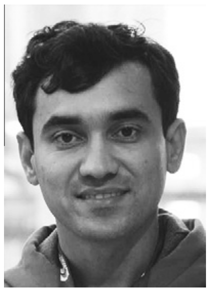
References

[1] A.Z. Broder, M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Math. 1 (4) (2003).
[2] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[3] H. Cai, P. Ge, J. Wang, Applications of Bloom filters in peer-to-peer systems: Issues and questions, in: NAS'08: Proceedings of the 2008 International Conference on Networking, Architecture, and Storage, Washington, DC, USA, 2008, pp. 97–103.
[4] P. Hebden, A. Pearce, Data-centric routing using Bloom filters in wireless sensor networks, in: M. Palaniswami (Ed.), Fourth International Conference on Intelligent Sensing and Information Processing (ICISIP-06), IEEE Press, Bangalore, India, 2006, pp. 72–78.
[5] T. Wolf, Data path credentials for high-performance capabilities-based networks, in: ANCS'08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM, New York, NY, USA, 2008, pp. 129–130.
[6] F. Ye, H. Luo, S. Lu, L. Zhang, S. Member, Statistical en-route filtering of injected false data in sensor networks, in: INFOCOM, 2004, pp. 839–850.
[7] S. Ratnasamy, A. Ermolinskiy, S. Shenker, Revisiting IP multicast, in: Proceedings of ACM SIGCOMM'06, Pisa, Italy, 2006.
[8] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, P. Nikander, LIPSIN: line speed publish/subscribe inter-networking, in: Proceedings of ACM SIGCOMM'09, Barcelona, Spain, 2009.
[9] R.P. Laufer, P.B. Velloso, D. d. O. Cunha, I.M. Moraes, M.D.D. Bicudo, M.D.D. Moreira, O.C.M.B. Duarte, Towards stateless single-packet IP

traceback, in: The 32nd IEEE Conference on Local Computer Networks (LCN'07), Washington, DC, USA, 2007, pp. 548–555.
[10] A. Whitaker, D. Wetherall, Forwarding without loops in icarus, in: Proceedings of OPENARCH 2002, 2002.
[11] T. Aura, P. Nikander, Stateless connections, in: ICICS'97, Springer-Verlag, London, UK, 1997, pp. 87–97.
[12] C.E. Rothenberg, F. Verdi, M. Magalhães, Towards a new generation of information-oriented internetworking architectures, in: First Workshop on Re-Architecting the Internet, Madrid, Spain, 2008.
[13] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false-positive rate of Bloom filters, Inf. Process. Lett. 108 (4) (2008) 210–213.
[14] S. Lumetta, M. Mitzenmacher, Using the power of two choices to improve Bloom filters, Internet Math. 4 (1) (2007) 17–33.
[15] M. Jimeno, K. Christensen, A. Roginsk, A power management proxy with a new best-of-n bloom filter design to reduce false positives, in: 21st IEEE International Performance, Computing, and Communications Conference, 2002, pp. 125–133.
[16] A. Kirsch, M. Mitzenmacher, Less hashing, same performance: building a better Bloom filter, in: ESA'06, Springer-Verlag, London, UK, 2006, pp. 456–467.
[17] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, An improved construction for counting Bloom filters, in: ESA'06: Proceedings of the 14th conference on Annual European Symposium, Springer-Verlag, London, UK, 2006, pp. 684–695.
[18] E.-J. Goh, Secure indexes, Cryptology ePrint archive, Report 2003/216, 2003. <http://eprint.iacr.org/2003/216/>.
[19] D. Hwang, M. Chaney, S. Karanam, N. Ton, K. Gaj, Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates, in: SASC'08, February 2008. <http://mason.gmu.edu/dhwang/pdf/2008eSTREAM.pdf>.
[20] C. Rothenberg, P. Jokela, P. Nikander, M. Sarela, J. Ylitalo, Self-routing denial-of-service resistant capabilities using in-packet Bloom filters, in: 5th European Conference on Computer Network Defense (EC2ND), 2009, pp. 46–51.
[21] M. Särelä, C.E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, J. Ott, BloomCasting: security in Bloom filter based multicast, in: Springer, NordSec, Lecture Notes in Computer Science, 2010.
[22] C. Henke, C. Schmoll, T. Zseby, Empirical evaluation of hash functions for multipoint measurements, SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 39–50.
[23] M. Mitzenmacher, S. Vadhan, Why simple hash functions work: exploiting the entropy in a data stream, in: SODA'08: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, pp. 746–755.
[24] F. Hao, M. Kodialam, T.V. Lakshman, Building high accuracy Bloom filters using partitioned hashing, in: SIGMETRICS'07, ACM, New York, NY, USA, 2007, pp. 277–288.
[25] Y. Lu, B. Prabhakar, F. Bonomi, Bloom filters: design innovations and novel applications, in: 43rd Annual Allerton Conference, 2005.
[26] R. Nojima, Y. Kadobayashi, Cryptographically secure bloom-filters, Trans. Data Privacy 2 (2) (2009) 131–139.
[27] C.E. Rothenberg, C.A.B. Macapuna, F. Verdi, M. Magalhães, A. Zahemszky, Data center networking with in-packet bloom filters, in: SBRC 2010, 2010.
[28] C.A.B. Macapuna, C.E. Rothenberg, M. Magalhães, In-packet Bloom filter based data center networking with distributed OpenFlow controllers, in: IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010), Miami, Florida, USA, 2010.
[29] M. Särelä, C.E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, J. Ott, Forwarding anomalies in Bloom filter based multicast, INFOCOM, Shanghai, China, 2011, accepted for publication.
[30] M. Särelä, J. Ott, J. Ylitalo, Fast inter-domain mobility with in-packet Bloom filters, in: MobiArch'10: Proceedings of the Fifth ACM International Workshop on Mobility in the Evolving Internet Architecture, ACM, New York, NY, USA, 2010, pp. 9–14.
[31] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, P. Nikander, MPSS: Multiprotocol Stateless Switching, in: Global Internet Symposium 2010, 2010, pp. 1–6.
[32] C.E. Rothenberg, C.A.B. Macapuna, F. Verdi, M. Magalhães, The deletable Bloom filter: a new member of the Bloom family, IEEE Commun. Lett. 14 (6) (2010) 557–559.



Christian Esteve Rothenberg is a Research Scientist in the areas of IP systems and networking at Fundação Centro de Pesquisa e Desenvolvimento (CPqD), Campinas, Brazil. His current research interests span Internet routing and packet forwarding, data center networks, NGN/IMS, OpenFlow, and named data networking. He received his Ph.D. in Electrical and Computer Engineering from University of Campinas UNICAMP) in 2010. In his doctoral studies he worked on probabilistic data structures applied to packet forwarding in content-oriented networks. He was a visiting researcher at Ericsson Research Nomadiclab, Helsinki, Finland, and contributed to the EU FP7 Publish/Subscribe Internet Routing Paradigm (PSIRP) project. He holds the Telecommunication Engineering degree from the Technical University of Madrid (UPM), Spain, and the M.Sc. (Dipl. Ing.) degree in Electrical Engineering and Information Technology from the Darmstadt University of Technology (TUD), Germany, 2006. During his master thesis at Deutsche Telekom he worked on IMS-based fixed mobile convergence and mobility management, and was engaged in R&D activities on converged access networks (ScaleNet) and self-optimizing radio access networks.



Carlos Macapuna obtained the Computer Engineering degree from the Federal University of Pará (UFPA) in 2008. Currently, he is finishing the Master degree at University of Campinas (UNICAMP), where he will start his PhD in 2011. During the graduate studies he worked with ADSL technologies, and currently, he is interested in OpenFlow technologies, data center architectures and new packet forwarding paradigms.

Maurício F. Magalhães received the B.S. degree in Electrical Engineering from the University of Brasilia, Brazil, the M.S. degree in Electrical Engineering/automation from the University of Campinas (UNICAMP), Brazil, and the Dr. Engineer degree from the Laboratoire d'Automatique de Grenoble (LAG)/INPG, France, in 1975, 1979, and 1983, respectively. Currently, he is a Professor in the Faculty of Electrical and Computer Engineering, UNICAMP. His research interests include next-generation Internet and service architectures.

Fábio Luciano Verdi is a Full Professor of Computer Science at Federal University of São Carlos (UFSCar) Campus Sorocaba. He received his Master degree in Computer Science and Ph.D. degree in Electrical and Computer Engineering both from University of Campinas (UNICAMP). His research interests include computer network management, data centers, cloud computing and smart grid.

Dr. Alexander Wiesmaier studied computer science with the focus on cryptography, communication networks, and software engineering. He did his final year thesis on IT security and software engineering, especially in the field of public key infrastructures (PKI). In his doctoral studies he deepened and broadened his knowledge in many areas of IT security. Besides PKI and amongst other topics he also dealt with embedded systems, information systems, electronic voting, and electronic identities. During and after his studies he gained practical work experience in IT security companies such as FlexSecure GmbH (Eberstadt, Germany) and Safelayer Secure Communications (Barcelona, Spain). Currently, he holds a position as Senior Researcher at Technische Universität Darmstadt (Department of Cryptography and Computer Algebra – CDC/Center for Advanced Security Research Darmstadt – CASED). He is in charge of research projects in the field of applied cryptography.

Appendix H

Publication H

M. Särelä, C. Esteve Rothenberg, A. Zahemszky, P. Nikander and J. Ott. “BloomCasting: Security in Bloom filter based multicast.” In proceedings of the 15th Nordic Conference in Secure IT Systems (Nordsec) 2010, Aalto University, Espoo, Finland, 27-30 October 2010.

BloomCasting: Security in Bloom filter based multicast

Mikko Särelä, Christian Esteve Rothenberg, András Zahemszky, Pekka Nikander, and Jörg Ott

{mikko.sarela,andras.zahemszky,pekka.nikander}@ericsson.com, chesteve@dca.fee.unicamp.br,jo@netlab.tkk.fi,

Abstract. Traditional multicasting techniques give senders and receivers little control for who can receive or send to the group and enable end hosts to attack the multicast infrastructure by creating large amounts of group specific state. Bloom filter based multicast has been proposed as a solution to scaling multicast to large number of groups.
In this paper, we study the security of multicast built on Bloom filter based forwarding and propose technique called BloomCasting, which enables controlled multicast packet forwarding. In Bloomcasting group management is handled at the source, which gives control over the receivers to the source. Cryptographically computed edge-pair labels give receivers control over from whom to receive. We evaluate a series of data plane attack vectors based on exploiting the false positives in Bloom filters and show that the security issues can be averted by locally varying the Bloom filter parameters, the use of cryptographic hash functions, and per hop bit permutations on the Bloom filter carried in the packet header.

1 Introduction

Recently, a number of routing and forwarding proposals [25, 16, 32] are re-thinking one of the most studied problems in computer networking – scalable multicast [12, 23]. The unifying theme of these proposals is to use Bloom filters in packet headers for compact multicast source routing. This makes it possible for the multicast architecture to scale to the billions, or even trillions, of groups required, should the system need to support all one-to-many and many-to-many communications, such as tele and video conferencing, chats, multiplayer online games, and content distribution, etc.

While the Bloom filter is a space efficient data structure and amenable to hardware implementations, it is also prone to false positives. With in-packet Bloom filter based packet forwarding, a false positive results in a packet being erroneously multicasted to neighbors not part of the original delivery tree. Consequently, false positives lead to reduced transport network efficiency due to unnecessary packet duplications – a fair tradeoff given the potential benefits. However, false positives have also security implications, especially for network availability.

Earlier work [26] has identified three forwarding anomalies (packet storms, forwarding loops, and flow duplication) and two solutions that provide fault tolerance for such anomalies, namely, varying the Bloom filter parameters and performing hop-specific bit permutations. Our contribution is to analyze the anomaly related problems and solutions from security perspective. It has also been shown [13] that Bloom filters can act simultaneously as capabilities, if the hash values used for the Bloom filter matching are cryptographically secure and depend on the packet flow.

In this paper, we concentrate on the security issues of Bloom filter based multicast forwarding plane. We analyze service and network infrastructure availability. The contributions of this paper are a characterization and evaluation of the security problems and solutions related to Bloom filter based forwarding. Other security issues for multicast, such as key management, policy, long term secrecy, ephemeral secrecy, forward secrecy, and non-repudiation are out of scope for this paper.

Additionally, we propose BloomCasting, a source specific multicasting technique that integrates the provided security solutions together. In BloomCasting, group membership protocol is carried from the receiver to the source. This pushes both the costs and the control of the multicast group management to the source. The Bloom filter used to forward the traffic is gathered hop-by-hop along the unicast path to the group source.

The rest of the paper is organized as follows. In Section 2, we review the principal aspects of Bloom filter based forwarding and scope the problem of secure multicast for the purposes of this paper. We present BloomCasting, a secure source-specific multicasting technique in Section 3 and in Section 4, we describe the security solutions in more detail. We evaluate our approach In Section 5, review the related work in Section 6, and conclude the paper in Section 7.

2 Security issues in Bloom filter based multicast

As with unicast, securing multicast communications requires considerations in two orthogonal planes: the data plane (protecting multicast data forwarding) and the control plane (securing multicast routing protocol messages), although the problems are more difficult because of the large number of entities involved. While secure multicast data handling involves the security-related packet treatments (e.g., encryption, group/source authentication and data integrity) along the network paths between the sender and the receivers, control plane security aspects involve multicast security policies and group key management i.e., secure distribution and refreshment of keying material (see e.g. [22, 11, 23, 18, 24]). Ultimately, control plane security must be handled individually by each multicast routing protocol to provide authentication mechanisms that allow only trusted routers and users to join multicast trees (e.g., PIM-SM [3]).

Our focus in this paper, however, is elsewhere – on the *availability of the multicast infrastructure* in an open and general source specific multicast model [9]. A source specific multicast group is defined by the source and group address taken together. We assume that multicast groups can contain receivers anywhere in the network. This means that hierarchical addressing [19] cannot be used to scale up the system with sub-linear growth in routing table size in relation to the number of groups. The number of potential source specific groups grows exponentially with the number of nodes in the network – compared to quadratic growth in the number of potential unicast connections and logarithmic growth in the size of routing table based on hierarchical addressing. State requirements create a potential for denial-of-service (DoS) attacks as described in 'stateless connections' [4].

Bloom filter based source routing has been proposed as a solution to scaling multicast into large networks and number of groups [25, 16, 32, 13]. Such an approach places the state requirement at the source, instead of the routers alleviating the potential for DoS attacks against the network infrastructure.

2.1 Forwarding with in-packet Bloom filters

The Bloom filter [10] is a hash-based probabilistic data structure capable of representing a set of elements S and answering set-membership questions of the type "is $x \in S$?". The insert operations consist of, given a bit array of size m, for each element x in a set S of size n, $k \ll m$ independent hash values are computed $H_1(x), ..., H_k(x)$, where $1 \leq H_i(x) \leq m, \forall x$ and the corresponding bit array locations are set to 1. Conversely, asking for the presence of an element y in the approximate set represented by the Bloom filter involves applying the same k hash functions and checking whether all bit positions are set to 1. In that case, the Bloom filter returns a 'true', claiming that y is an element of S. The Bloom filter always returns the right answer for each inserted elements, i.e., there are no false negatives. However, due to hash collisions, there is some probability $p(m, n, k)$ for the Bloom filter returning a false positive response, claiming an element being part of S even when it was not actually inserted.

In-packet Bloom filter based multicast [25, 16, 32, 13] is based on the idea of turning the forwarding operations into a set-membership problem. The basic idea consists of encoding a multicast tree by inserting the appropriate link identifiers into a Bloom filter carried in the packet header. Forwarding nodes along the path process the packet and check whether neighboring link identifiers are present in the Bloom filter. Then, a copy of the packet is forwarded along the matching interface(s).

Inherited from Bloom filters, false positives cause packets to be unnecessarily duplicated over some extra links. When a router receives a falsely forwarded packet for which it does not find a matching forwarding directive, the packet is simply discarded. Hence, Bloom filter forwarding guarantees packet delivery to all intended destinations but introduces a degree of wasted resources due to unnecessary packet duplications – a tradeoff worth to consider given the benefits in terms of space efficiency (i.e., reduced state) and performance (i.e., fast forwarding decisions).

2.2 Threat model and existing attacks

We restrict the scope of this paper to security issues of the Bloom filter based forwarding plane of one-to-many multicast, also referred to as source-specific multicast (SSM) architectures. We assume an attacker who may control large number of hosts (e.g. botnet) that wishes either to disrupt the network infrastructure, or deny service to target host or network links. We also evaluate available possibilities for *controlled multicast*, i.e. ensuring that only authorized senders and receivers are capable of sending to and receiving from a particular multicast group.

Our adversary model assumes malicious end hosts and benign routers. Consequently, *packet drop attack* or *blackhole attack* fall out of the scope. This assumption is coherent with the wired networking scenario under consideration where trust among routers and the management plane is provided by e.g. pair-wise shared secret techniques. Moreover, we assume an end-to-end security mechanism to provide payload confidentiality, authentication, and integrity (e.g., as discussed in [15]). Attacks related to these security mechanisms are not discussed further in this paper.

While false positives represent a well-known limitation of Bloom filters, the security implications of (random) false positives in packet forwarding are far reaching and less understood. Our main security goal is to guarantee *forwarding service availability* of Bloom filter based data planes under malicious attacks. Hence, we seek for data plane mechanisms that ensure that only packets from authorized users are forwarded, i.e., providing resistance to (potentially distributed) DoS attacks .

DoS can be divided into attacks on infrastructure availability and (end) service availability. These can be disrupted by bandwidth, state, or computation consumption attacks (cf.[7]). Any unauthorized sending of multicast data can be construed as a DoS attack. For instance, *flooding attacks* would cause an escalating of packets filling the network links to an extend that legitimate packets end up discarded due to massive link congestion. Such denial of service may affect a greater proportion of the network due to the "multiplier effect" of false-positive-prone multicast packet distribution.

Chain reaction attacks False positives can cause forwarding anomalies that greatly increase the amount of network traffic. These include packet storms, forwarding loops, and flow duplication [26]. We review these anomalies that an attacker could do here. We highlight the fact that if Bloom filters are assigned per multicast tree or per flow, the anomalies will affect every packet in a given multicast tree or flow.

Packets storms are caused when, for sizable part of the network, the average number of false positives per router exceeds one. Should this be the case, then on average each false positive causes more than one additional false positive, creating an explosive chain reaction. The average number of false positives is $\rho^k \cdot (d - b - 1)$, where ρ is the fill factor of the Bloom filter, k is the number of hash functions used, d is the number of neighbors, and b is the number of actual

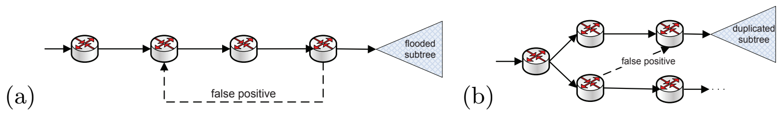


Fig. 1. (a) Forwarding loop and (b) flow duplication

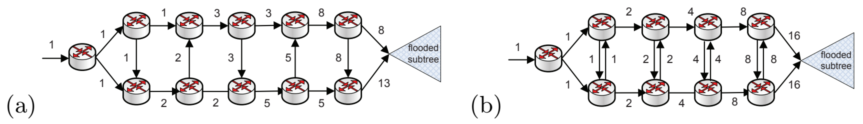


Fig. 2. (a) Flow duplication with Fibonacci number growth in the number of packet copies and (b) exponential growth in the number of packet copies

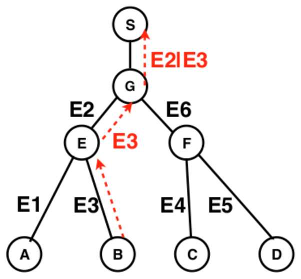
branches in the multicast tree at that node. After the first false positive $b = 0$. As an example, considering $k = 5$ and $\rho = 0.5$, a nodes with degree $d > 32$ would produce more than one false positive per node.

Forwarding loop happens, if a set of false positives cause the packet to return to a router it has already visited. The router will then forward the packet again to all the nodes downstream of it, including the false positive links that caused the packet to loop. As a result, not only will the packet loop, but every loop causes a copy of the packet to be sent to the full sub-tree below the router. A forwarding loop is shown in Figure 1.

Flow duplication is another possible anomaly as shown in Figure 2. Figures 2(b)-(c) show that even flow duplication can cause the number of packet to grow – according to Fibonacci sequence and as the powers of two.

The above attacks can also be combined. If link identifiers are common knowledge, the attacker can form a Bloom filter that corresponds to the Figure 2(c) which also includes one or more links back to the first router, causing the packet load to explode both in network and in all receiver hosts.

Target path attack An attacker controlling a large number of hosts can try to coordinate as many packet flows as possible to a single link or a particular path. If link identifiers are common knowledge (1), then this is simple. Each host just computes a forwarding tree that goes through chosen link. If however, the link identifiers are secret and static (2), then the attacker has a few potential attacks available: *injection attack* – where she tries Bloom filters that get traffic forwarded along a certain delivery tree, *correlation attack* – where she attempts to infer workable link identifiers from a collection of legitimate Bloom filters, and *replay attack* – where a valid Bloom filter is misused to send unauthorized traffic (i.e., with different content or flow identifiers). [13]



Source S

Receiver	BF
A	E1|E2
B	E2|E3
C	E4|E6
D	E5|E6
ABCD	E1|E2|E3| E4|E5|E6

Fig. 3. The left side shows multicast Join message using iBFs. The right side shows a simplified Membership Table MT(S) that contains the Bloom filters for A, B, C, and D. The separated bottom row shows how to combine the Bloom filters in to an iBF.

3 BloomCasting

BloomCasting is a secure source specific multicast technique, which transfers the membership control and per group forwarding state from the multicast routers to the source. Similar to [25, 16, 32], it uses in-packet Bloom filter (iBF) to encode the forwarding tree. BloomCasting separates *multicast group management* and *multicast forwarding.*

To join, a host sends a join request (`BC_JOIN`) towards the source S. Intermediate routers record forwarding information into the packet, thus when the packet reaches S, it will contain a *collecting* iBF for the source-receiver path. By combining together the iBFs for all the receivers, the source will have an iBF encoding for the whole multicast tree. When a host does not wish to receive packets for the group anymore, it sends an authenticated leave message to S. Upon processing this packet, the source will reconstruct the Bloom filter for the group leaving out the recently pruned path. The operation is illustrated on Figure 3.

Data packets are routed using the *forwarding* iBF placed in the `BC_FW` header. Each intermediate router takes its forwarding decision by querying it with the question: which of my outgoing links are present in the iBF? It then forwards the packet to the corresponding peers. Eventually, the packet reaches all the receivers, following the sequence of routers the `BC_JOIN` packets traversed, in reverse order.

3.1 Group Membership Management

Group membership management includes the joining, leaving, and maintenance of multicast groups, and this is the main task of the control plane. Along this discussion, we show how multicast trees are encoded into an in-packet Bloom filter (iBF).

Joining a group: When a host joins a multicast group, it sends a (`BC_JOIN`) message towards the source. The packet contains the following information: (S,G)

Algorithm 1: Adding edge-pair labels (E) and permuting collect and forward iBFs at transit routers.

```
Collect_iBF (C):
  E ← $Z_K$(S,G,$R_p$, $R_c$, $R_n$);
  C ← C ∨ E;
  C ← $\text{Permute}_c$(C);

Forward_iBF (F):
  foreach outgoing link i do
      F ← $\text{Permute}_c^{-1}$(F);
      E ← $Z_K$(S,G,$R_n$, $R_c$, $R_p$);
      if $E \wedge F = E$ then
          Send F → i;
      end
  end
```

specifying the multicast group and a *collecting* iBF. The latter is used for collecting the forwarding information between the source and the receiver. Finally, it also contains a hash chain anchor for future signaling security.

In each router, the next hop for the BC_JOIN message towards S is found from routing information base.[1] As the message travels upstream towards the source, each router records forwarding information into the packet by inserting the edge pair label E into the *collector* iBF. After this, for loop prevention and increased security, it performs a bit permutation on the *collector* iBF. Finally, it selects the next hop usptream towards S. The operation is shown on Algorithm 1. Unlike traditional IP multicast approaches, where the forwarding information is installed in routers on the delivery tree, transit routers do not keep any group-specific state.

Once the BC_JOIN message reaches the source, it contains sufficient information so that the source can send source-routing style packets to the recently joined host. The source stores this information in the Membership Table (MT), as shown in Figure 3. The source can now send packets to the multicast tree by combining iBFs for the group, by bitwise ORing them together.

Leaving a group: When a receiver wishes to leave the group, it sends a BC_LEAVE towards S, including the next element from the hash chain it used when joining the group. On-path routers forward the packet to S. As no further processing is needed in intermediate routers, unlike pruning packets in IP multicast, BC_LEAVE packets always routed to the source.

S verifies the hash and removes (or de-activates) the entry in the Membership Table. Single message hash authentication, vulnerable to man-in-the-middle attacks, is sufficient, since the hash is only used to verify that the host wishes to

[1] Just like in standardized IP multicast protocols, this forwarding decision can be taken according to the RIB created by BGP or according to the Multicast RIB created by MBGP [8].

leave the group. As a final step, it recomputes the forwarding iBF of the delivery tree. An example of a forwarding iBF is shown in Figure 3 at the separated bottom row of the table.

Refreshing membership state: The iBFs in the MT may become stale, either because of changing the key to compute the edge-pair labels or due to route failures. Keys are expected to change periodically (e.g., every few hours) to increase security by excluding brute force attacks [13]. This means that the iBF needs to be recomputed with a new `BC_JOIN` packet. When making the forwarding decision, during a transition period routers need to compute edge-pair labels for both the old and the new key. If they find that an edge-pair label computed with the old key is present in the iBF, they set a flag in the `BC_FW` header indicating that the receiver should send a `BC_JOIN` again, as the iBF will soon become invalid. When a packet is to be forwarded on a failed link, the router sends an error message back to the source.

3.2 Multicast Forwarding

So far, we have discussed how hosts join and leave multicast groups. We now show how data packets are forwarded between the source and the receiver.

As we saw previously, iBFs for each receiver border router are stored separately in the Membership Table. We also saw the basic concept of deriving the forwarding iBF from the MT information; now we extend that with new details.

For each group, the source stores one or more iBF for each next hop router in its BloomCasting Forwarding Table (BFT).[2] In practice, the capacity of a packet-size iBF is limited in order to guarantee a certain false positive performance (practical values suggest around 25 destinations in 800-bit iBFs [25]). In case of large multicast groups, several iBFs are created, one for each partial multicast tree, and duplicate packets are sent to each next hop.

The source creates one copy of the packet for each next hop for (S,G) in the BFT. It creates a `BC_FW` header, fills it with the corresponding iBF, and sends it to the next hop router.

Each router makes a forwarding decision based on the iBF, as shown in Algorithm 1. First, it applies the reverse permutation function to the iBF, replacing the iBF with the result. Then, it checks for the presence of peer routers by computing one edge-pair label for each potential next hop router R_n, based on the previous and the current router on path R_p and R_c respectively,[3] and on group identity (S,G) found in the IP header as shown in Algorithm 1. In the final step, the router checks whether the iBF contains the edge-pair label, by simple bitwise AND and comparison operations.

The remaining problem is how to compute the dynamic edge-pair labels at core routers at line speed. This can be done by taking the values (S, G, K, Rp, R_c, R_n)

[2] This improves forwarding performance, as the false positive probability increases with the number of iBF inserted elements.

[3] The router uses the same inputs as in the `BC_JOIN`. hence the R_p and R_n switch places due to direction change

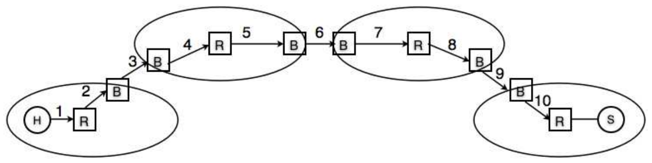


Fig. 4. Protocol messages when joining group (S,G): 1 - IGMP Membership Report or MLD Multicast Listener Report; 2,10 - PIM-SSM JOIN(S,G); 3-9 - BC_JOIN(S,G)

and running them through a fast, spreading hash function (cf. e.g. [20, 31]). The spreading hash function yields the bit locations for the edge-pair labels. The method can be applied locally at each router, having no impact on the protocol.

3.3 Connecting intra-domain multicast with BloomCasting

BloomCasting can be used to specify the operations between source and receiver ASes.[4] This section discusses how multicast forwarding state is set up inside the domains containing the sender and/or receivers using IP multicast (PIM-SSM deployments) Figure 4 illustrates the protocol messages when a multicast receiver joins a multicast group (S,G).

When a receiver joins (S,G), it signals (1) its interest in its LAN with IGMPv3 or MLDv2 protocols. The Designated Router then sends a PIM-SSM `JOIN(S,G)` message upstream (2), by looking up the reverse path to S. The message is propagated upstream until a router is found that holds forwarding state for the group or until a border router of the domain is reached (standard PIM-SSM operations). The border routers implement both PIM-SSM and BloomCasting. PIM signaling now terminates. If the border router was not yet a receiver for the group, it creates a `BC_JOIN` packet and sends it towards S (3-9).

The iBF collection process is otherwise as described in Section 3.1 except each AS is considered to be a singe logical router.

At the other end, the source AS border router receives a `BC_JOIN` for a group that resides in the local domain and processes it as specified in Section 3.1. If it is not yet a receiver for the group locally, it sends a join packet using PIM-SSM standardized operations (10). The `JOIN(S,G)` is propagated further upstream towards the source, with standard PIM operations. Eventually, as far as PIM concerned, a domain-local multicast tree will be built with routers representing local receivers and border routers representing subscribers in remote domains.

The data packets are *forwarded* using the domain-local multicast protocol to the border routers in the source AS. The border router creates a single copy for each entry in the BFT, adds the BloomCasting header, and forwards the packets. When an ingress border router receives packet with the `BC_FW` header, it checks whether it has domain-local receivers and forwards a decapsulated copy using

[4] An AS is an autonomous system, a network participating in the inter-domain routing protocol (BGP). The source and receiver could also be an area consisting multiple ASes that deploys a shared multicast architecture.

the domain-local multicast protocol. The router also checks whether neighboring domains are included in the iBF and forwards the packet to those domains (using e.g. IP-in-IP, GRE encapsulation, or MPLS paths or trees).

4 Security techniques in Bloom Filter based forwarding

In Section 2, we introduced a threat model for in-packet Bloom filter based forwarding by showing several attacks taking advantage of some forwarding anomalies inherent to Bloom filter based forwarding. Now, we present techniques for solving these forwarding anomalies; then, in Section 5, we evaluate them from security perspective.

Basically, the solutions presented here include pre-processing verification of Bloom filters, and some rules to be followed in the packet forwarding process and during the Bloom filter creation phase.

Limiting the fill factor (ρ_{max}) ensures that the attacker cannot set, e.g., all bits in the Bloom filter to 1, which would equal to every link in the network being included. Before any packet handling operation, routers need to verify the Bloom filter [30], i.e. they need to check for ρ_{max} compliance before attempting to forward the packet. Typically, ρ_{max} is set to ≈ 0.5, which corresponds to the most efficient usage in terms of bits per element per false positive rate.

Cryptographic Bloom filters: Bloom filters for forwarding can be differentiated based on the nature of the link identifiers: (1) link identifiers are common knowledge [25], (2) link identifiers are secret, but static [16], and (3) link identifiers are secret per flow and change periodically with key are computed per incoming/outgoing edge pair instead of per link [13].

Bloom filters gain capabilities [2], when the edge pair label is computed using cryptographically secure hash functions, secret key, and flow information from the packet (e.g., IP 5-tuple, (S, G)). Each link identifier of size m and with k bits set to one (i.e., a one element Bloom filter) can be computed as the output of a function $zF(In, Out, K(t_i), I, p)$. The resulting identifiers become *dynamic* and bound to the In and Out interfaces of each hop i, the time period of a secret key $K(t_i)$, and additionally dependent of packet information like the Flow ID I (e.g., source and group addresses) and an optimization parameter p (cf.Z-formation [13]).

Varying the number of hash bits (k_{var}): This technique deals with the number of ones (k) in the link identifiers set by different routers, and aims to decrease the false positive rate. Assuming that there is a fixed maximum fill factor ρ for iBF, e.g. $\rho = 0.5 + \epsilon$, the average number of false positive in a given router depends on its degree d, the number of hash functions k it uses, and the number of outgoing branches b such that the average number of false positives is $\rho^k \cdot (d - b - 1)$. Hence, we proposed [26] that each router sets k locally such that $\rho^k \cdot d < \alpha$, where $\alpha < 1$ sets the routers preference for the average false positive rate. As routers compute the hash functions for the *collecting* iBF themselves, the number k is purely a local matter. In other words, the number of bits k set to 1 in the link identifiers is not a global parameter, but can be defined per node.

Permutations $P_i(iBF)$: We use per hop bit permutations to prevent loops and flow duplications. A bit permutation is a (pseudo) random rearrangement of the bit array. Each router can use the same bit permutation for all iBFs passing through it making it easy to implement with programmable logic.

First, after passing through the intended path to a router R, the $f$orwarding iBF has to match the k hash values that the router added when the *collecting* iBF was forwarded through it. When the iBF is collected, the routers between R and source S change some bits from 0 to 1 and permute the packet. S then combines a set of *collecting* iBFs in to a *forwarding* iBF and the routers between S and R (including R) perform reverse permutations on the iBF. Hence, once the packet arrives in R, the bits that R set to 1 will be in exactly the same positions as they were when the iBF was collected. Since no operation changes a value of a bit from 1 to 0, the matching process works correctly.

Second, if the path taken is different from the one intended for the packet, the iBF should not match the k hash values. Per hop bit permutations enable the iBF itself to carry a “memory” of the path it has traversed [26]. As each router modifies the iBF when forwarding the packet, after passing through two different edges and entering back the initial node, i.e. after a loop, the iBF is changed with a random bit permutation. Hence, it will likely not match the same edge-pair labels again. Each router permutes $P_i(iBF)$ when the iBF is initially collected and then reverse permutes $P_i(iBF)^{-1}$ the iBF when a packet is sent using the iBF.

5 Security evaluation

We now analyze how BloomCasting mitigates the security threats (described in Section 2) against Bloom filter based multicast forwarding. As mentioned in Section 2.2, we focus on malicious host-initiated attacks. Further architectural considerations w.r.t scalability, bandwidth efficiency, state requirements, control overhead, etc. are out of scope of this evaluation and left for future work.

Table 1 presents an overview of the mapping between the available techniques (Section 4) and the attacks addressed. As can be seen, BloomCasting combines four techniques to prevent the six security threats described in Section 2.2.

Packet storms are prevented with the combination of limiting the maximum fill factor ρ_{max} and the *varying k_{var} technique.* Globally enforced ρ_{max} values enable each router to compute k_{var} locally so that every Bloom filter with a valid fill factor produces, on average, less than 1 false positives. Since the Bloom filters are collected on path with the `BC_JOIN` packet, it is easy to set k_{var} locally. Additionally, this optimization of k reduces the actual false positive rate [26].

Loops are a serious threat to any network infrastructure. The combination of maximum fill factor ρ and z-Formation makes it difficult for an attacker to construct looping Bloom filters. The first removes the easy attack of just adding bits into the Bloom until every link matches and the z-Formation ensures that guessing the right Bloom filter is difficult (see [13] for details).

Table 1. Mapping of solutions to attacks

Attack - Technique	ρ_{max}	k_{var}	z-F	$P(iBF)$
Packet storms	+	+		
Loops	+		+	+
Flow duplication	+		+	+
Injection	+		+	
Correlation			+	+
Replay			+	

To prevent accidental loops, each router performs a bit permutation on the Bloom filter before performing the outport matching – when using the Bloom filter for forwarding (and after matching – when collecting a Bloom filter). If a packet does go through a loop, either because of a false positive or a malicious source, the Bloom filter has been modified with a random permutation (a product of the permutations performed by the set of routers participating in the loop).

Using permutations ensures a high probability that the packet will not continue looping and that it will not be forwarded to the downstream tree for a second, or nth time. As an example, the chances of an infinite loop in a three node loop configuration with $\rho = 0.5$, $k = 6$, and $m = 256$ are in the order of $O(10^{-12})$. The chances that a packet will be forwarded through the subtree once are ρ^{κ}, where $\kappa = \sum k_i$ is the sum of all hash functions used in the subtree.

Finally, while the security is not dependent on the secrecy of the permutations performed in each router, it is dependent on the secrecy of the edge-pair labels. Consider a *known permutation* attack, in which an attacker knows the network topology and the permutations used by a set of routers. It can now compute the cycle sets of the combined permutation and choose a combination that has approximately the size of the maximum fill factor. However, it does not know a combination of a Bloom filter and source and group address that will ensure that the routers on path and in the loop will have edge-pair labels that match the Bloom filter. The best it can do is vary the group address. In this case, the probability of success is ρ^{κ}, where κ is the total number of bits that need to be set on path to the point of loop and in the loop.

Flow duplication: Similarly to loops, flow duplication can be effectively prevented with the combination of restricting fill factor ρ, edge-pair labels, and per hop bit permutations. The result gives an attacker ρ^{κ} probability of creating a specific subtree by accident.

Packet injection attacks, correlation attacks, and replay attacks can be efficiently prevented using the z-Formation technique [13]. It uses cryptographically secure edge-pair labels that are computed based on the flow, and time, and path. This makes it impossible to share iBFs from different points of network, at different time instants, or to different destinations.

Consequently, the best strategy for a successful *packet injection attack* is reduced to a brute force attack consisting of generating random labels and hoping that at least one of them reaches the target(s). An attacker needs malformed iBFs to cause h consecutive false positives to get packets forwarded through a valid iBF path of length h. The chances of success in one attempt can be approximated to $p = {\rho_{max}}^{h \cdot k}$, which is very low for typical configurations (e.g., $p = 2^{-36}$ for $h = 4, k = 8, \rho = 0.5$, i.e., over 10^{10} attempts are required for a 1/2 probability successful attack). Such brute force attacks can be easily detected, rate limited and pushed back, for instance after the false positive rate from a given source exceeds some threshold. Additionally, a forged iBF would work through the target path attack as long as the distributed secret K(t) is not renewed.

Source and receiver control: As the group management in BloomCasting is end-to-end, it gives source control over the receivers it accepts. If it wishes to, it can require receiver authentication before adding a receiver into the group. Similarly, multicasting to a receiver requires knowing the iBF that forms the path between source and destination. Since the iBF is cryptographically bound to (S,G), each router's secret key, and the path (via permutations and edge-pair labels), guessing an iBF for a path is difficult, as shown above.

Resource consumption attacks against the memory and processing capacity of routers do not become easier than they are in unicast forwarding. The routers do not need to maintain multicast state and the iBF collection and forwarding processing can be done in line speed in hardware and in parallel for each peer. The multicast source needs to maintain state for receivers. This is a needed feature, since this makes it possible source control over who can and who cannot join the multicast group. Simultaneously, it leaves the source vulnerable to attacker who creates a storm of multicast join packets. A source can use a receiver authentication protocol, which pushes the authentication protocol state to the initiator (e.g., the base exchange of Host Identity Protocol [21] could be used for that purpose) to limit the state requirements to authenticated receivers.

False positive forwarded packets may compromise the *ephemeral secrecy* of the multicast data to non group-members, i.e., some packets may reach unintended destinations. The time- and bit-varying iBFs contribute to spreading false multicasted packets across different links over time, preventing thus a complete reception of a multicast packet flow.[5]

Anonymity of source is not an option in source specific, since the group is identified with combination (S,G) where S is the sender address and G the group address. However, even though the protocol uses source routing, the actual paths, or nodes on path are not revealed to the source and the source can only use the related iBFs in combination with traffic destined to (S,G).

Receivers do not need to reveal their identifies or addresses to the network, or the source – the receiver (IP) address is not necessary in the protocol. The authentication, should the source require it, can be done end-to-end without revealing the identities to the intermediate routers. As the keys used to compute

[5] As assumed in Section 2, *data authenticity* is kept out of scope of the iBF forwarding service and can be provided by orthogonal security policies and group key management techniques (e.g., following the guidelines of [15]).

iBFs are changed periodically, correlation attacks between two or more Bloom filters used at different times become impossible. Similarly, since the edge-pair labels are tied to group identifier (S,G), an attacker cannot use a set of iBFs with different group addresses to determine whether the set contains one or more common receivers. These techniques effectively prevent traffic analysis and related vulnerabilities such as clogging attacks (cf. [5]).

6 Related work

Compared with unicast, multicast communication is at a substantially increased risk from specific security threats due to the lack of effective group access control and larger opportunities for link attacks. Over the last decade, much effort has been put in the field of multicast security, see e.g. [6, 18, 11, 15, 27].

At the IETF, earlier work has provided a taxonomy of multicast security issues [11] and a framework for secure IP multicast solutions [14] to address the three broad core problem areas identified: (i) fast and efficient source authentication (e.g. [6, 17]), (ii) secure and scalable group key management techniques, and (iii) methods to express and implement multicast-specific security policies. Our focus, however, has been on DoS attacks against the network infrastructure.

Service availability attacks due to routing loops and blackholes were discussed in [28]. The proposed solution was the keyed HIP (KHIP) protocol to allow only trusted routers joining the multicast tree. Our aim is a general and open SSM architecture that does not require group access restrictions provided by the infrastructure.

Free Riding Multicast [25] proposes an open any source multicast service in which each link is encoded as a set of hashes from the AS number pair. This leaves the forwarding plane open to a variety of attacks. Odar [29] showed that Bloom filters can be used for anonymous routing in adhoc networks. Limiting fill factor as a security feature in Bloom filter based (unicast) capabilities was first proposed in [30]. LIPSIN [16] uses Bloom filter forwarding plane for publish/subscribe architecture with a separate topology management system that helps to keep the link identifiers secret. However, an attacker can still use target path attacks. Z-formation [13] prevents target path attacks by using edge-pair labels that depend on flow identifier, but is still open to e.g. chain reaction attacks.

Si^3 [1] proposed a secure overlay to solve problems related to secure multicast. While distributed hash tables spread load efficiently across the system, they lack e.g. policy compliant paths and control over who is responsible for particular connection.

7 Conclusions

In this paper, we evaluated the security of Bloom filter based forwarding. False positives inherent to Bloom filters enable a host of attacks on target service and network infrastructure availability. These attacks include chain reaction attacks, which use the Bloom filter properties (e.g. false positives) to ensure that the network forwarding infrastructure multiplies every packet sent using the Bloom filter and targeted attacks in which the attacker enables many nodes to target the same path in the network.

We show that these problems can be solved by the combination of limiting Bloom filter fill factor, both minimum and maximum, using cryptographically computed edge-pair labels in the Bloom filters, varying the number of hash functions locally based on the router degree, and using per hop bit permutations on the Bloom filter.

We also proposed BloomCasting, a secure source-specific multicasting technique based on in-packet Bloom filters. The technique is based on end-to-end signaling of group membership and hop-by-hop collection of the needed Bloom filters. As future work, we intend to study the possibility of collecting multiple paths in advance as a technique for increasing fault tolerance to route failures.

References

1. Adkins, D., Lakshminarayanan, K., Perrig, A., Stoica, I.: Towards a more functional and secure network infrastructure (2003)
2. Anderson, T., Roscoe, T., Wetherall, D.: Preventing Internet denial-of-service with capabilities. ACM SIGCOMM Computer Communication Review 34(1), 44 (2004)
3. Atwood, W., Islam, S., Siami, M.: Authentication and Confidentiality in Protocol Independent Multicast Sparse Mode (PIM-SM) Link-Local Messages. RFC 5796 (Proposed Standard) (Mar 2010), `http://www.ietf.org/rfc/rfc5796.txt`
4. Aura, T., Nikander, P.: Stateless connections. In: ICICS '97: Proceedings of the First International Conference on Information and Communication Security. pp. 87–97. Springer-Verlag, London, UK (1997)
5. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In: IHW '01: Proceedings of the 4th International Workshop on Information Hiding. pp. 245–257. Springer-Verlag, London, UK (2001)
6. Ballardie, T., Crowcroft, J.: Multicast-specific security threats and countermeasures. In: SNDSS '95: Proceedings of the 1995 Symposium on Network and Distributed System Security (SNDSS'95). p. 2. IEEE Computer Society, Washington, DC, USA (1995)
7. Barbir, A., Murphy, S., Yang, Y.: Generic Threats to Routing Protocols. RFC 4593 (Informational) (Oct 2006), `http://www.ietf.org/rfc/rfc4593.txt`
8. Bates, T., Chandra, R., Katz, D., Rekhter, Y.: Multiprotocol Extensions for BGP-4. RFC 4760 (Draft Standard) (Jan 2007), `http://www.ietf.org/rfc/rfc4760.txt`
9. Bhattacharyya, S.: An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational) (Jul 2003), `http://www.ietf.org/rfc/rfc3569.txt`
10. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
11. Canetti, R., Pinkas, B.: A taxonomy of multicast security issues. IRTF Internet-Draft (draft-irtf-smug-taxonomy-01) (August 2000)
12. Diot, C., Dabbous, W., Crowcroft, J.: Multipoint communication: A survey of protocols, functions, and mechanisms. IEEE Journal on Selected Areas in Communications 15(3), 277–290 (1997)

13. Esteve, C., Jokela, P., Nikander, P., Särelä, M., Ylitalo, J.: Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters. Proceedings of European Conference on Computer Network Defence (EC2ND) (2009)
14. Hardjono, T., Canetti, R., Baugher, M., Dinsmore, P.: Secure ip multicast: Problem areas, framework, and building blocks. IRTF Internet-Draft (draft-irtf-smug-framework-01) (September 2000)
15. Hardjono, T., Weis, B.: The Multicast Group Security Architecture. RFC 3740 (Informational) (Mar 2004), `http://www.ietf.org/rfc/rfc3740.txt`
16. Jokela, P., Zahemszky, A., Esteve, C., Arianfar, S., Nikander, P.: LIPSIN: Line speed publish/subscribe inter-networking. In: SIGCOMM (2009)
17. Judge, P., Ammar, M.: Gothic: a group access control architecture for secure multicast and anycast. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. vol. 3, pp. 1547 – 1556 vol.3 (2002)
18. Judge, P., Ammar, M.: Security issues and solutions in multicast content distribution: A survey. IEEE Network 17, 30–36 (2003)
19. Kleinrock, L., Kamoun, F.: Hierarchical routing for large networks Performance evaluation and optimization. Computer Networks (1976) 1(3), 155 (1977)
20. Krawczyk, H.: LFSR-based hashing and authentication. In: Advances in Cryptology–CRYPTO'94. pp. 129–139. Springer (1994)
21. Moskowitz, R., Nikander, P.: Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational) (May 2006), `http://www.ietf.org/rfc/rfc4423.txt`
22. Moyer, M., Rao, J., Rohatgi, P.: A survey of security issues in multicast communications. IEEE network 13(6), 12–23 (1999)
23. Paul, P., Raghavan, S.V.: Survey of multicast routing algorithms and protocols. In: ICCC '02: Proceedings of the 15th international conference on Computer communication. pp. 902–926. International Council for Computer Communication, Washington, DC, USA (2002)
24. Rafaeli, S., Hutchison, D.: A survey of key management for secure group communication. ACM Computing Surveys (CSUR) 35(3), 329 (2003)
25. Ratnasamy, S., Ermolinskiy, A., Shenker, S.: Revisiting IP multicast. ACM SIGCOMM Computer Communication Review 36(4), 26 (2006)
26. Särelä, M., Rothenberg, C.E., Aura, T., Zahemszky, A., Nikander, P., Ott, J.: Forwarding Anomalies in Bloom Filter Based Multicast. Tech. rep., Aalto University (October 2010), submitted for publication
27. Savola, P., Lehtonen, R., Meyer, D.: Protocol Independent Multicast - Sparse Mode (PIM-SM) Multicast Routing Security Issues and Enhancements. RFC 4609 (Informational) (Oct 2006), `http://www.ietf.org/rfc/rfc4609.txt`
28. Shields, C., Garcia-Luna-Aceves, J.J.: Khip—a scalable protocol for secure multicast routing. In: SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication. pp. 53–64. ACM, New York, NY, USA (1999)
29. Sy, D., Chen, R., Bao, L.: Odar: On-demand anonymous routing in ad hoc networks. In: Proc. of IEEE Mobile Adhoc and Sensor Systems (MASS). pp. 267–276 (2006)
30. Wolf, T.: A credential-based data path architecture for assurable global networking. In: Proc. of IEEE MILCOM. Orlando, FL (October 2007)
31. Yuksel, K.: Universal hashing for ultra-low-power cryptographic hardware applications. Ph.D. thesis, Citeseer (2004)
32. Zahemszky, A., Jokela, P., Särelä, M., Ruponen, S., Kempf, J., Nikander, P.: MPSS: Multiprotocol Stateless Switching. In: Global Internet Symposium 2010 (2010)