Data Center Fault-Tolerant Routing and Forwarding: An Approach based on Encoded Paths

Ramon Marques Ramos
Federal University of Espírito Santo (UFES)
Vitória - ES - Brazil
rramos@inf.ufes.br

Magnos Martinello
Federal University of Espírito Santo (UFES)
Vitória - ES - Brazil
magnos@inf.ufes.br

Christian Esteve Rothenberg
Telecom Research and Development Center (CPqD)
Campinas - SP - Brazil
esteve@cpqd.com.br

Abstract

Traditional layer 2 and layer 3 network designs face some limitations on data center networks such as lack of scalability, difficult management and inflexible communication. We observe that data center networks are often managed as a single logical network fabric with a well-controlled baseline topology and growth model. This paper describes a data center networking approach based on encoded paths carried in the packet header. We present an OpenFlow-based testbed implementation of a data center architecture governed by a logically centralized Network Manager, which is responsible to transparently provide the networking and support functions to operate the data center network. We evaluate the proposal in terms of fail recovery time, state requirements, and load balancing capabilities. The results of the experiments show that our proposal improves the fail recovery time, while preserving scalability requirements.

1. Introduction

Data Center Networks (DCN) are an essential component of today's cloud-oriented Internet infrastructure. Proper planning of the data center infrastructure design is critical, where performance, resiliency, and scalability under certain cost constraints require careful considerations [1]. Data centers are growing in size and importance as many enterprises are migrating their application and data to the cloud. To achieve the benefits of resource sharing, these data centers must scale to very large sizes at marginal cost increases.

Continued growth of DCN requires a reliable infrastructure because interruptions in services can have significant consequences to enterprises [3]. An increasingly portion of Internet traffic is based on the data communication and processing that takes place within data center networks. However, the routing, forwarding, and management protocols that run in today's data centers were designed for the general LAN setting and are proving inadequate along a number of dimensions [10, 20].

The Ethernet protocol relies on network-wide flooding to locate end hosts and broadcast based protocols, like ARP, resulting in large state requirements and control message overhead that grows with the size of the network. Another weakness of current Ethernet architectures is relying on the Spanning Tree Protocol (STP). While STP performs well for small networks, it introduces substantial inefficiencies on larger networks that have more demanding requirements for low latency, high availability, and traffic engineering. IP routing ensures efficient and flexible use of networking resources via shortest path routing, but it leads to a huge configuration overhead and hampers the arbitrary allocation of Virtual Machines (VMs) to any physical server.

In order to conceive this vision, we intend to build a data center network that meets the following objectives:

- **Layer 2 semantics**: Just as if the servers were on a LAN, where any IP address can be connected to any port of an Ethernet switch due to flat addressing, data center management software should be able to easily assign any server to any service.

- **Multipathing**: Network traffic should be capable of using all paths between the communicating parties and avoid link congestion. This in turn means to remove

STP which wastes precious bandwidth by blocking many links to prevent forwarding loops.

- **Scalability**: To support the rising use of server, network, and storage virtualization, the DCN design should ensure the ability to scale with the growing communication and addressing requirements.

- **Fault tolerance**: At large scale, failures are the norm, so failure recovery should be rapid and efficient. Existing sessions should be capable of recovering from failures of network nodes and the routing/forwarding paths should converge as fast as possible.

- **Automatic configuration**: The astronomical size of virtualized data centers makes manual network administration tasks a prohibitive and error-prone configuration burden. To avoid service outages and provide agility, the network architecture should be able to rely on plug-and-play mechanisms.

Traditional fixed network approaches are simply not designed to meet the current needs of a virtualized environment and are not flexible enough to support the changing demands. Recently, a promisingly disruptive networking technology referred to as Software Defined Network (SDN) is emerging. SDN is an architectural proposition based on a number of key attributes, including: separation of data and control planes; a uniform vendor-agnostic interface to the forwarding engine (i.e. OpenFlow [17]); a logically centralized control plane; and the ability to slice and virtualize the underlying physical network [8].

This paper contributes with a set of Ethernet-compatible routing and forwarding, and fault tolerance mechanism with the goal of meeting the requirements discussed above. We propose a scheme that employs centralized controller defining rules at the source switch to embed the path information within the packet header in form of **Encoded Path (EP)**. The EP specifies a set of nodes within the data center through which the intermediate switches should forward the packets.

The advantage of this source routing approach is that the forwarding task of each switch becomes trivial, as it requires only local knowledge of its neighbors, rather than a forwarding entry for every potential destination. Besides, since the first hop switch specifies the EP, the multiple available paths can be explored. As the paths are precomputed, when a fail occurs in any topology link, another link-disjoint path can be selected quickly by only embedding a different encoded path at the originating switch without requiring network-wide convergence algorithms in the control plane. The proposed architecture also provides support for ARP handling, and load balancing. The experimental evaluation of our prototype implementation suggests that the design space of new DCNs has been unleashed by OpenFlow/SDN.

The rest of the paper is organized as follows. Section 2 introduces background information about DCN architectures and Software Defined Networks and discusses related works. Section 3 presents our design architecture and implementation, describing the key functional blocks . Section 4 evaluates our proposal in terms of fail recovery time, network state requirements, and load balancing capabilities. Finally, Section 5 concludes the paper and outlines the future works.

2. Background and Related Works

2.1. Data Center Networks

There are a number of data forwarding alternatives in data center networks. The high-level dichotomy is between creating a layer 2 network or a layer 3 network, each with associated tradeoffs. A layer 3 approach assigns IP addresses to hosts hierarchically based on their directly connected switch. In the topology of Figure 1, hosts connected to the same ToR (Top of Rack) could be assigned the same /26 prefix and hosts in the same row may have a /22 prefix. Using segmentation via subnetting, such careful IP assignment enables relatively small forwarding tables across the data center switching fabric.

Standard intra-domain routing protocols such as OSPF [18] may be employed among switches to find shortest paths among hosts. OSPF is able to detect failures and broadcast the information to all switches to avoid failed links or switches. Transient loops with layer 3 forwarding is less of an issue as the IP-layer TTL limits the per-packet resource consumption during convergence.

The biggest problem with the IP-centric approach is its massive configuration overhead [9]. Improperly synchronized state between system components, such as a DHCP server and a configured switch subnet identifier can lead to unreachable hosts and difficult to diagnose errors.

For these reasons, certain data centers deploy a layer 2 network where forwarding is performed based on flat MAC addresses. A layer 2 fabric imposes less administrative overhead, but commonly requires a loop protection mechanism such as STP. Spanning tree automatically breaks loops, preventing broadcast packets from continuously circulating and melting down the network, but it leads to suboptimal paths and uneven link loads. Load is especially high on links near the root bridge. Thus, choosing the right root bridge is extremely important, which imposes an additional administrative burden. Moreover, flooding and source-learning introduce two problems in a large broadcast domain. The forwarding tables at a bridges can grow very large because flat addressing increases the table size proportionally to the total number of hosts in the network. And the control overhead required to disseminate each hosts information via flooding can be very large, wasting link bandwidth and processing resources [10].

A middle ground between a layer 2 and layer 3 fabric consists of employing VLANs to allow a single logical layer 2 fabric to cross multiple switch boundaries. While feasible for smaller-scale topologies, VLANs introduces several problems. Bridges provisioned with multiple VLANs must maintain forwarding table entries and process broadcast traffic for every active host in every VLAN visible to themselves, limiting scalability. Furthermore, they require bandwidth resources to be explicitly assigned to each VLAN at each participating switch, limiting flexibility for dynamically changing communication patterns. Finally, VLANs also use a single spanning tree to forward packets, which prevents certain links from being used.

2.2. Fat-tree Network Designs

The fat-tree topology has many properties that are attractive for large scale interconnection and system area networks [15, 14]. This topology is inherently highly resilient with a large number of redundant paths between two processing nodes, and most importantly, the bisection bandwidth of the fat-tree topology scales linearly with the network size. As a result, the switching capabilities of the network form an integral part of its overall performance. For the most effective use of resources, the switching network must not present any bandwidth bottlenecks in the flow of data between nodes [2, 25].

A three-tier arrangement built from 20 4-port switches is shown in Fig 1, which presents a lower layer of ToR switches, an intermediate layer of Aggregation (AGGR) switches, and an upper layer of CORE switches. Based on this architecture, data may be transmitted between any two hosts, undergoing a maximum of five switch hops.

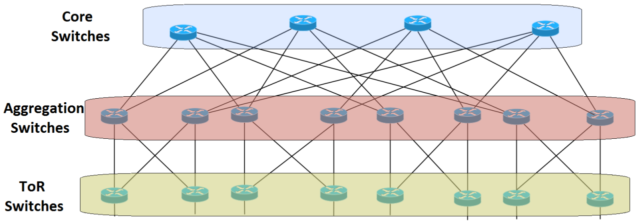


Figure 1. Fat-tree Topology

2.3. Software Defined Networking

While many new data center innovations are limited, there is a promising alternative under the umbrella term of Software-Defined Networking (SDN) [21]. The control plane in SDN is commonly called the network operating system and is separated from the forwarding plane. Typically, the network OS (e.g NOX [8]) observes and controls the entire network state from a central vantage point, hosting features such as routing protocols, access control, resource virtualization, energy management, and new prototype features.

An uniform vendor-agnostic interface named *OpenFlow*, has been the trigger that enables SDN designs by providing a standardized protocol between the network OS and the forwarding elements. OpenFlow allows to fully control the forwarding state in either a proactive or reactive way. In the latter, when an OpenFlow switch receives a packet for which it has no matching flow entry, it sends the 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.

The centralized view of SDNs simplify the conception of control applications. For example, to implement shortest-path routing, the controller can calculate the forwarding rules for each switch by running Dijkstras algorithm on the graph of the network topology rather than running a more complicated distributed protocol. To enforce a fine-grained access control policy, the controller can involve an external authentication server and install a custom forwarding path for each user. To balance the load between back-end servers in a data center, the controller can migrate ows in response to server load and network congestion.

2.4. Related Works

Our attention in this section is devoted to research works specifically targeting scalable layer 2 networks for data center architectures.

VL2 [7] provides the illusion of a large L2 network on top of an IP network, using a logically centralized directory service. VM hosts add a shim layer to the networking stack called the VL2 agent which encapsulates packets into a IP-in-IP tunnel.

PortLand [20] proposes a scalable routing and forwarding protocol for data centers with three-tiered hierarchical topologies. 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. PortLand employs a centralized fabric manager to resolve ARP queries, and to simplify multicast and fault tolerance. When a link failure occurs in PortLand, fabric manager informs all affected switches of the failure, which then individually recalculate their forwarding tables based on the new version of the topology.

SiBF [22] represents a source route compactly into an in-packet Bloom Filter. The bloom filter bits are carried by the MAC fields and the MAC rewriting occurs at the source and destination ToR switches.

Unlike VL2, our proposal avoids end point modifications and shim-header overheads. This works also differs from Portlad by employing precomputed paths which leads to decrease the failure recovery overhead. Compared to SiBF, our approach does not use bloom filter avoiding false positives, which causes a packet to have more than one next-hop candidate.

3. Design and Implementation

The goal of this work is to provide a scalable, fault-tolerant layer 2 routing and forwarding approach to fat-tree data center networks. The proposed architecture is based on separating route computation (on the deployed topology) from failure handling (when link status changes) [4]. Thus, the core idea is to let routing compute every potential path based solely on the deployed topology. That is, routing should ignore all status changes and only recompute paths when the topology changes. Since these topology changes are rare and often known in advance [4], the computation of routes can be done in a centralized fashion. This flexibility, in both the length and location of the computation, allows a wide variety of paths computation according to various criteria.

The novelty in our approach is to explore the structure of fat-tree topologies, enabling fast forwarding and fault-tolerant routing by allowing the source switch to embed the path information within the packet header in form of the **Encoded Path (EP)**. The EP specifies a set of nodes within the data center through which the intermediate switches should forward the packets. In case of a failure event (link or switch), the recovery mechanism consists on modifying the Encoded Path embedded in the packets allowing them to use other available network paths. All the canditate paths are previously computed.

The proposed architecture employs a logically centralized Network Manager (NM) that makes all decisions on the data center network. The NM has been implemented as an application on top of the NOX controller. Basically, the NM receives a map of the network (see topology discovery details in Sec. 3.1) to i) calculate all routes between each pair of hosts, and ii) install the forwarding state of switch the intermediate switches solely based on the neighbouring switch information.

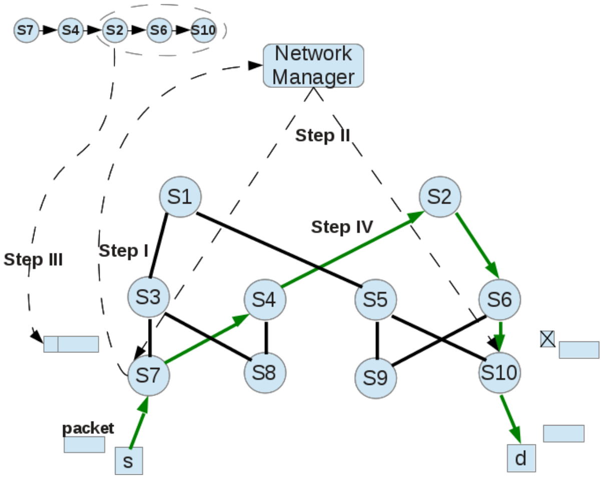


Figure 2. Overview of the design.

Figure 2 shows an example to illustrate the proposed design. Suppose the source s wishes to send a packet to a destination d. The packet misses a rule in the flow table of source ToR S7 and is forwarded to the NM (Step I). The NM selects the path S7-S4-S2-S6-S10 and installs one rule (OpenFlow entry) at the source and destination ToRs (Step II). The rule at source ToR S7 instructs the switch to (i) embed the selected path into an encoded version (EP) written in the packet header fields (Step III), and (ii) forward the packet via the outport to S4. At S4, S2 and S6, all the forwarding decisions are based on the contents of the EP carried in the packet header (Step IV). Finally, the rule in the destination ToR S10 removes the EP and delivers the packet to the destination d (Step V). Note that the EP does not specify the entire path (end-to-end), but only the necessary fabric nodes to get the packet forwarded between the ToRs.

In the following, we describe the details of the underlying assumptions and mechanisms.

3.1. Topology Discovery and Route Computation

In order to construct source routes, two pre-requisites are required: (1) topology information, and (2) server location. At bootstrap time, the Network Manager needs to know the network topology and correctly infer the tree topology and the role of each switch (i.e., ToR, AGGR, CORE) without manual intervention.

To discover the topology of the network we use the Discovery application available in NOX. Discovery is a LLDP application for topology inference. It sends LLDP packets out of every switch interface and then uses received LLDP packets on the neighbouring switch to infer working switch-to-switch links (e.g. switch A port 1 connected with switch

B port 4). This information is enough to run a simple algorithm that infers the level of each switch in the topology. The server location information is gathered from the NOX Authenticator module that keeps record of all authenticated hosts and users in the network.

The route computation module can be configured to calculate all routes between all pairs of hosts, or a predetermined number k of paths. This parameter can be defined by the network administrator based on the network size. For better performance, routes can be selected based on the disjoint criteria [16].

3.2. Encoding the Path

After choosing a path between two communicating entities, the controller must install a rule in the source ToR that encodes the selected path into a sequence of packet header bits. The concept itself is agnostic to the particular location in the packet header used to carry the bits. It may reside in a shim header between the IP and MAC layers, in an IP option, or in a novel header format in a next-generation Internet protocol. Our key goal in designing the encoding format is to ensure simple data plane forwarding that can be deployed over existing hardware (e.g., commercial OpenFlow switches).

In the proposed approach, the EP is represented as a sequence of segments, one for each switch that forwards packets based on the EP. For instance, in Figure 2 a path between s and d consists of 5 hops, but only the 3 intermediate hops perform actual forwarding operations based on the EP. The source and destination ToRs send packets to a port based on the rule installed by the Network Manager, so, in this case the EP consists of 3 segments. The segment corresponding to a switch Si encodes the node's next hop.

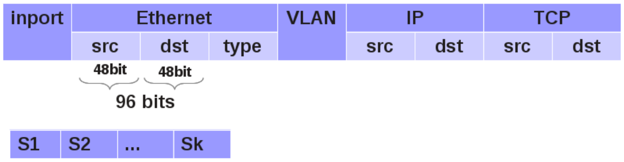


Figure 3. Encoding format layout.

We chose to encode the path into the 96 bits of the Ethernet MAC fields (Figure 3), with the segments having the size of $96/k$ bits, where k is the number intermediate switches on the longest path. [1]

Therefore, in a 3 tier fat-tree topology, each segment receives a 32 bit space. Since OpenFlow switches are identified by a 64-bit value called Datapath ID (dpid), the NM has to map the dpid to a number of bits that can fit within the segments. These mappings only need to be locally unique to each switch and its neighbours and hence 32 bits or less is more than sufficient.

3.3. Forwarding based on Encoded Paths

We discuss next how routers use the EP information to forward packets. The key idea is to split the 96 bits of the MAC Ethernet fields in blocks, each block to be used by a switch along the path to forward the packet. Thus, the core and aggregation switches need to match against a block of bits (i.e. partially wildcarded) in the Ethernet fields. Unfortunately, the OpenFlow 1.0 switches used in our testbed implementation do not support arbitrary wildcard masks for the L2 fields. However, with a simple workaround we were able to emulate the same EP-based forwarding using plain 1.0 OpenFlow features.

In the 3-tier fat-tree topology under consideration, an EP requires 3 blocks that can be carried in 3 independent Ethernet field, namely, MAC_{SRC}, MAC_{DST}, and $VLAN$. The MAC_{SRC} field plays the role of the first EP block, the MAC_{DST} field serves as the intermediate EP block, and, finally, the $VLAN$ field acts as the last EP block. So, the switches can match against the full Ethernet fields without requiring bitmask support in the Ethernet MAC fields.[2]

3.4. Forwarding Mechanism

We now describe the forwarding mechanism at intermediate switches. The input to this operation is the header described in §3.2, and the output is the interface out to which the packet is forwarded.

In EP forwarding, the IP address is used only to identify the hosts (or VMs) within the data center. This way, unlike the traditional hierarchical assignment of IP addresses, there are no restrictions on how addresses are allocated. This option makes the IP address not significant for routing packets inside the data center forwarding fabric. In essence, we separate host location from host identifier in a manner that is transparent to end hosts and compatible with existing commodity switch hardware. Moreover, this approach does not introduce additional protocol headers such as MAC-in-MAC or IP-in-IP.

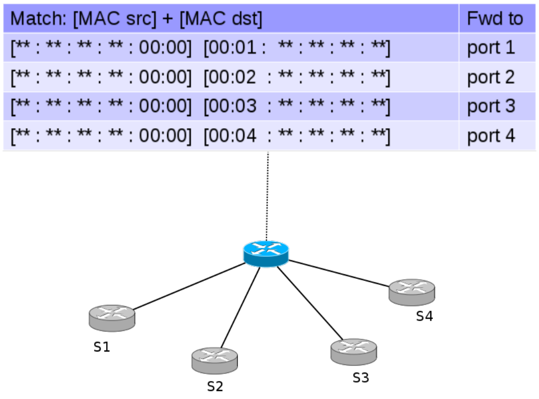
The forwarding mechanism in the ToRs is different from the intermediate switches. When a packet arrives at the ToR and lacks of a matching entry, the packet is sent to the NM which will install the necessary rules to deliver the packet to the destination. In the source ToR, the rule matches the destination MAC address and instructs the ToR to rewrite

[1] If the segments can not be divided equally (e.g., in a fat-tree of 4 levels requiring 5 segments), there is no problem if one of the segments has different size.

[2] Note that OpenFlow protocol version 1.1 [5] onwards supports arbitrary bitmasks on the Ethernet fields, so the EP can be implemented as described in section 3.2.

the MAC field for the EP and forward the packet to the next hop. In the destination ToR, the rule matches the host identifier (IP address), the EP is removed by restoring the correct Ethernet fields, and the packet is delivered to the destination host. To improve the end-to-end communication setup time, the NM simultaneously installs the rules needed at both ToRs, so that the reply can flow directly through the datapath without involving the NM.

Forwarding in the intermediate switches is based on the fixed rules installed by the NM during booting up. These rules, based on the EP, indicate the next hop to which the packet should be sent. Instead of traditional exact matching of MAC fields, these rules match an EP segment, i.e. a wildcarded subset of the MAC bits. Figure 4 shows an example of a switch's forwarding table when connected to four neighbors. The switch makes the forwarding decision based on the 32-bit segment (bits 32-63).



Match: [MAC src] + [MAC dst]	Fwd to
[** : ** : ** : ** : 00:00] [00:01 : ** : ** : ** : **]	port 1
[** : ** : ** : ** : 00:00] [00:02 : ** : ** : ** : **]	port 2
[** : ** : ** : ** : 00:00] [00:03 : ** : ** : ** : **]	port 3
[** : ** : ** : ** : 00:00] [00:04 : ** : ** : ** : **]	port 4

Figure 4. Flow table of a intermediate switch.

Which segment to be used for matching depends on the position of the switch in the packet's path. In general terms, the first intermediate switch forwards the packet based on the first segment, and so forth. However, when a packet goes from a upper layer to a lower layer the packet may divert from its path because of "double matches", as shown in Fig. 5. In order to prevent double matches, before a switch sends the packet via the output port the matched segment field is erased by an additional action that rewrites the field with all zeros.

3.5. Fault-Tolerant Routing

Given the baseline topology and the ability to forward based on EPs, we are now concerned with fault tolerance. In this work, we focus is on the network failure recovery after the failure is detected, independently of the failure detection method to be used. Candidate approaches for failure detection include path-level mechanisms [6, 23] or controller-based approaches the LLDP-based topology discovery. Different detection mechanisms will yield different link recovery times. For the purposes of this evaluation, we assume a $port-down$ event received by the controller which will react with a fault restoration mechanism as described below.

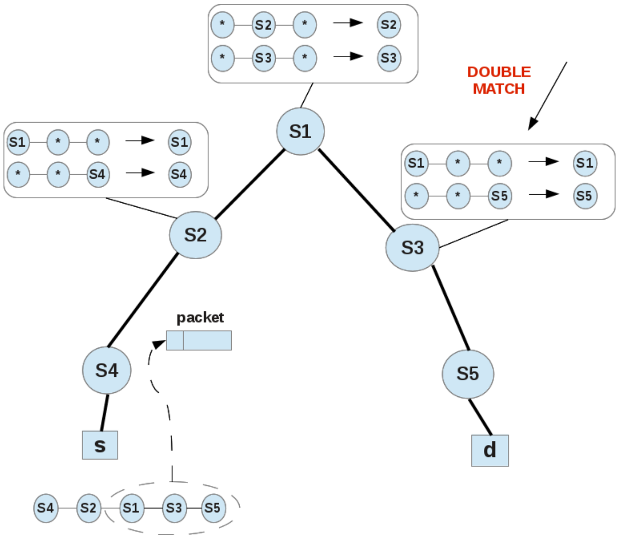


Figure 5. EP-based forwarding

The fail recovery process can be described using the example illustrated in Fig. 6. The switch detects a link failure (Step I) and informs the NM Manager about the failure (Step II). The NM updates the connectivity matrix that it maintains for the entire topology (Step III) and identifies the installed flows that pass through the affected link. The NM then chooses new available routes for the affected flows and sends $flow-mod$ messages (Step IV) to the corresponding ToR to modify the EP to route via the alternative routes.

Traditional routing protocols, like OSPF, require all-to-all communication among n switches with O(n^2) network messages and associated processing overhead. Our proposal requires one message from the switch that detects the fault to the controller, and one message to each ToR that forwards packets to the discontinued routes, resulting in a O(n) communication.

Alternatively, the NM could install two rules with different priorities at a ToR switch, each one specifying a link-disjoint EP. If the high priority path is interrupted, the NM (or the switch itself) can simply remove (or decrease the priority of) this entry and the current flow packets will start matching with the backup EP entry. Exploiting features introduced in the new versions of OpenFlow such as fast failover groups, the switch itself could change the forwarding behavior to the backup EPs without requiring a round trip to the controller, which would be later informed about the link down events and the local recovery state changes.

Failures on the controller or on the connection between switches and controller may occur. This kind of problem has been addressed considering a logically centralized, but physically distributed controller [11, 24]. The control plane resilience is beyond the scope of this work and will be addressed in future prototype implementations.

3.6. Address Resolution Protocol - ARP

ARP resolution is performed in a centralized fashion without the need of broadcast messages. ARP requests are also used to associate each hosts IP address with its attachment point to the ToR switch.

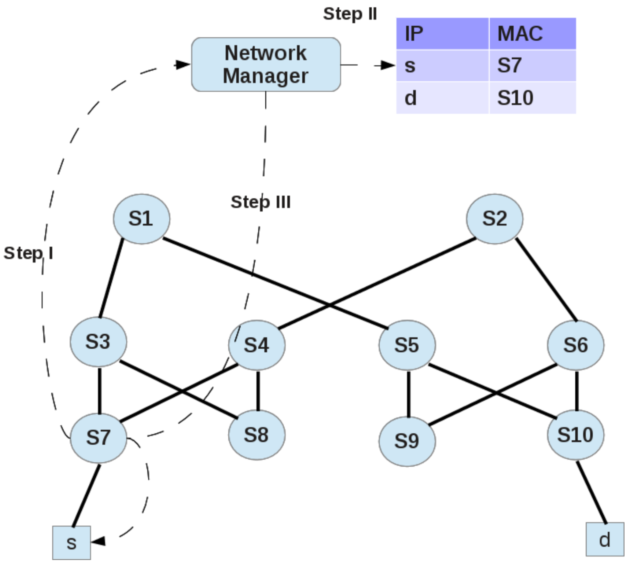
Ethernet by default broadcasts ARPs to all hosts in the same layer 2 domain. We leverage the Network Manager to reduce broadcast overhead, as depicted in Fig. 7. In step I, a ToR switch receives an ARP request for a destination IP and forwards the request to the NM. In step II the NM consults its MAC table to see if an entry is available for the target IP address. If so, it creates an ARP reply in step III and returns it to the original host.

End hosts receive the destination ToR switch MAC address in response to an ARP request. In the source ToR, the installed rule can match either just the destination ToR MAC address, or any combination of fields up to the full 10-tuple. The design choice offers a trade-off between zero delay for upcoming flows towards the same destination ToR (as there will be already a matching entry for MAC_{DST}) and the load-balancing granularity and effectiveness (as the subsequent flows will follow the same EP). How optimize this trade-off depending on the application needs (e.g., short low latency flows vs. longer BW intense flows) is subject of future work. the destination host, the Network Manager will install the missing rules.

3.7. Load Balancing

Designing data center networks using rich multipath topologies such as fat-trees solves the high-bandwidth requirement. However, these networks require load balancing mechanisms to best utilize the multiple end-to-end paths to provide high bisectional bandwidth. Load balancing can be performed in a stateless fashion with no overhead using oblivious routing [25], where the path between the communicating nodes s and d is randomly selected from a probability distribution over all s to d paths. We follow this approach and let the NM select, at random, one among the precomputed paths.

At runtime, the NM may decide to employ additional flow table entries to override the initial forwarding behavior of individual flows. For example, knowing that certain hosts, or certain TCP ports are used to exchange large volumes of data, such as large backups or big data processing, specific rules for these flows could be installed, avoiding the default path, potentially using additional knowledge about the application requirements and the network state.



IP	MAC
s	S7
d	S10

Figure 7. Address Resolution Protocol

4. Evaluation

In this section, we evaluate the EP approach in terms of failure recovery time, load balance capabilities, and state requirements. Our testbed platform is based on Mininet [13], a platform for rapidly prototyping with OpenFlow topologies and controller applications. The topology used in the tests is the same as shown in Fig. 1, a multi-rooted fat-tree with 4 core switches, 8 aggregation switches, 8 Tor Switches and two hosts connected to each ToR. The switches inside Mininet connect to an OpenFlow controller in the same machine. The OpenFlow protocol version used in our implementation is 1.0.

4.1. Failure Recovery Time

We evaluate the failure recovery mechanism for both UDP and TCP flows. The goal is to measure the time to reestablish the flows after a failure is detected. Note that the failure detection time is not considered in these experiments.

UDP traffic is started between two hosts, such hosts are chosen so that the distance between them was 5 hops, and is introduced a varying number of random link failures. In the case where at least one of the failures falls on the selected

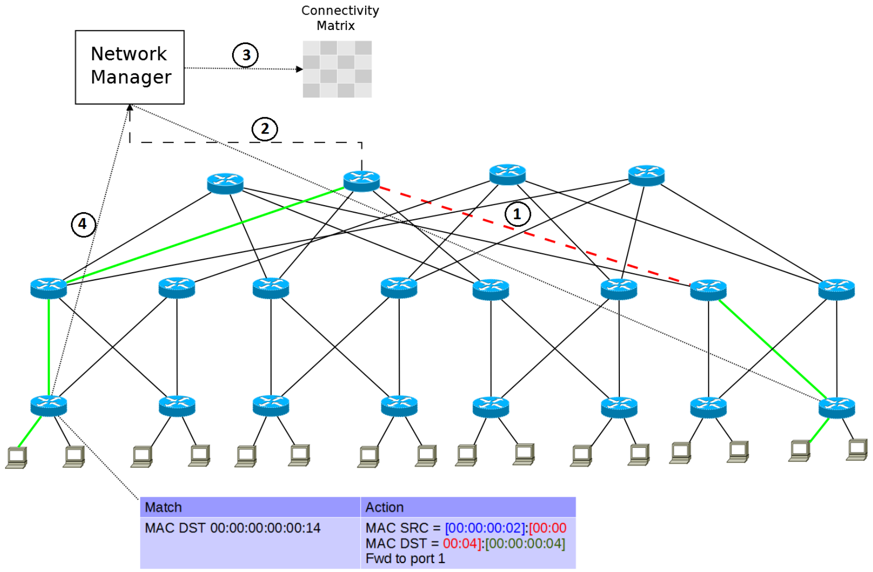

Connectivity Matrix
Network Manager
3
2
1
4
Match
Action
MAC DST 00:00:00:00:00:14
MAC SRC = [00:00:00:00:02]:[00:00
MAC DST = 00:04]:[00:00:00:04]
Fwd to port 1

Figure 6. Fail Recovery

path between sender and receiver, we measured the time required to reestablish communication.

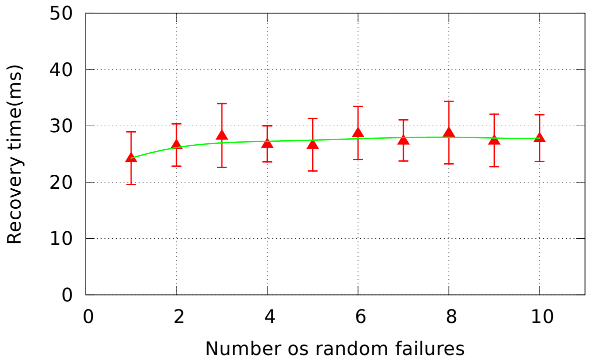


Figure 8. UDP Recovery Time

Figure 8 shows the average recovery time as a function of the number of randomly failures, after running 10 experiments. Recovery time was about 23 ms for a single failure and increases slowly with the number of failures. In fact, the controller may need to repeat the procedure of changing the Encoded Path at ToRs which explains this small variation as the number of failures increases. It is worth to mention that this result has been obtained using the same topology and methodology as PortLand. Although the metric used in PortLand was the time to detect and recovery from failures, it had an average time of 60 ms.

Recovery time was presented without considering the time of the failure detection, while Portland considers this time, because we wanted to isolate the contribution of the failure detection mechanism, either on a controller-based or dataplane-based (e.g. BFD, Ethernet OAM, OSPF Hello). Further on the PortLand comparison, it uses the proposed location discovery protocol(LDP) for failure detection but the paper does not specify the frequency of LDP packets, which would be required to try a fair and accurate comparison.

The same experiment was repeated for TCP communication. We monitored network activity at the sender while injecting a link failure along the path between sender and receiver. As illustrated in Figure 9, the recovery time for TCP flows takes longer than the baseline for UDP. This discrepancy results is because TCP loses an entire window worth of data. The TCP retransmission timeout is approximately 200 ms in our system. By the time the first retransmission takes place, connectivity has already been reestablished in the network.

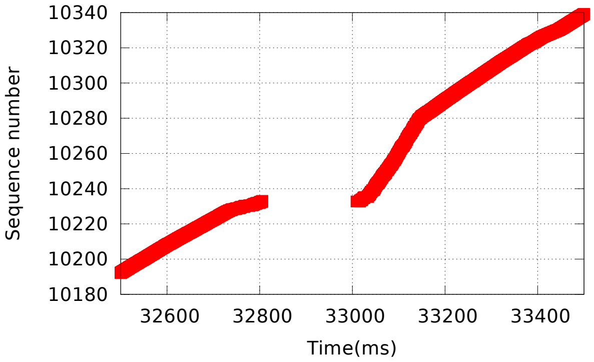


Figure 9. TCP Recovery Time

4.2. Load Balance capabilities

Now, we investigate the load balancing capabilities of the oblivious routing with EP forwarding in our testbed environment. Given an all to all communication, the goal is to evaluate how well the traffic is spread among the available links. We compare the link utilization of our implementation with a standard Spanning Tree (SPT) implementation over the same topology. We used textitiperf to generate the tcp flows.

Figure 10 shows the normalized link utilization after 10 (ten) experiment runs. As expected, SPT over-utilizes some links (40%), while most of links (approximately 60%) are not utilized. On the other hand, the oblivious routing approach spreads traffic more uniformly through the network, varying around 25% from the ideal value , i.e., 1, as it can be seen in the figure.

4.3. State Analysis

Finally, we compare the state requirements among PortLand [20], VL2 [7], SiBF [22] and EP. The network topology is a three-tier network topology, with ToRs connected to 20 servers and 2 AGGRs. The AGGR switch ports (p_1) are used to connect to p_1/2 ToRs and p_1/2 COREs with p_2 links. Depending on the exact values of p_1 and p_2 , the interconnection matrix can be scaled from e.g., 3,000 (p_1 = 24) to 100,000 (p_1 = 144) physical nodes. Due to the strict source routing approach, flow table requirements are minimal and constant in the COREs and AGGRs switches, i.e., only one entry per interface as shown in Table 1 for PortLand, SiBF and EP. On the other hand, VL2 requires a number of forwarding entries proportional to the number of switches in order to route packets along the two-levels of IP encapsulation.

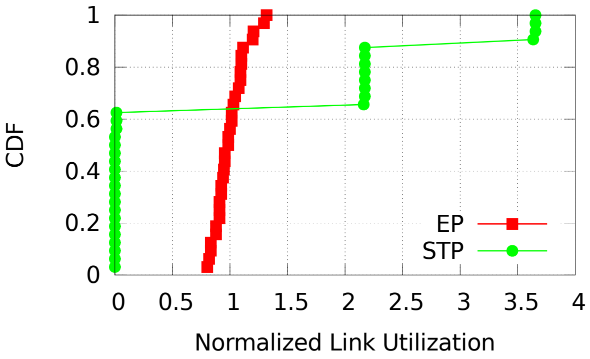


Figure 10. Load balance

For EP, further scaling the network does not affect the number of entries flow in the switches that is constant and equal to the number of neighboring switches. At ToRs, the amount of flow entries grows with the number of simultaneous outbound flows (assumed 5 per VM) plus a constant amount of entries (1 per VM) in order to perform the MAC re-writing and correct delivery of inbound packets.

5. Conclusion and Future Work

In this paper, we have presented an early implementation of an OpenFlow-based source routing mechanism to attain scalability and fault-tolerant routing solution for data center networks. The proposal works by encoding in the switching edges the end-to-end path into data packet headers to forward the flows throughout the fabric. Our implementation on a virtualized testbed shows that the approach can achieve efficient failure recovery, good multi-path utilization while using minimal flow table state in the switches.

Future work will extend our current implementation in a number of ways. We will consider the hypervisor vSwitch as the first networking hop where initial flow matching and header re-writing takes place. We will investigate support to VM migration and multicast services. We also plan to add extensions to an OpenFlow 1.2 software switch to provide local failure recovery capabilities and exploit the group table fast failover functionalities. One idea along this avenue is to let the packets carry alternative paths in the packet header, as proposed in [19]. This way, switches can react to failures without depending on the NM. In addition, upon failure events, packets sent out via the alternative paths may carry enough information about the failed links encountered between source and destination so that convergence can take place much faster across the data plane, an idea suggested in [12] for inter-domain routing.

References

[1] *Cisco Data Center Infrastructure 2.5 Design Guide*. Cisco Systems, Inc, San Jose, CA, 2007.

[2] Focalpoint in large-scale clos switche. Technical report, Fulcrum Microsystems, 10 2007.

[3] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[4] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker. Dynamic route recomputation considered harmful. *SIGCOMM Comput. Commun. Rev.*, 40(2):66–71, Apr. 2010.

[5] O. Consortium. OpenFlow 1.1 switch specification. *Can be accessed at http://openflowswitch. org/*, pages 1–56, 2011.

[6] M. Fu, Z. Le, and Z. Zhu. Bfd-based failure detection and localization in ip over obs/wdm multilayer network. *Int. J. Commun. Syst.*, 25(3):277–293, Mar. 2012.

Physical hosts	2.880	23.040	103.608
Racks	144	1152	5148
Aggregation	24 (p1=24)	96 (p1=48)	144 (p1=144)
Core	12 (p2=24)	24 (p2=96)	72 (p2=144)

	VL2	PortLand	SiBF	EP	VL2	PortLand	SiBF	EP	VL2	PortLand	SiBF	EP
Entries at ToR	200	120	120	120	1292	120	120	120	5420	120	120	120
Entries at AGGR	180	24	24	24	1272	48	48	48	5400	144	144	144
Entries at Core	180	24	24	24	1272	96	96	96	5400	144	144	144

Table 1. State Analysis

[7] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[9] Z. Kerravala. Configuration management delivers business resiliency. The Yankee Group, November 2002.

[10] C. Kim, M. Caesar, and J. Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. *SIGCOMM Comput. Commun. Rev.*, 38(4):3–14, Aug. 2008.

[11] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[12] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. *SIGCOMM Comput. Commun. Rev.*, 37(4):241–252, Aug. 2007.

[13] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[14] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, Oct. 1985.

[15] X. Lin, Y. Chung, and T. Huang. A multiple lid routing scheme for fat-tree-based infiniband networks. *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.

[16] S. Mahapatra, X. Yuan, and W. Nienaber. Limited multi-path routing on extended generalized fat-trees. *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 938–945, May 2012.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[18] J. Moy and G. J. Moy. The OSPF Specification, 1989.

[19] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker. Slick packets. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 245–256, New York, NY, USA, 2011. ACM.

[20] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[21] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks, 2012.

[22] C. E. Rothenberg, C. A. B. Macapuna, F. L. Verdi, M. F. Magalhes, and A. Zahemszky. Data center networking with in-packet bloom filters. *XXVIII Simpsio Brasileiro de Redes de Computadores e Sistemas Distribudos*, 2010.

[23] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford. Network architecture for joint failure recovery and traffic engineering. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 97–108, New York, NY, USA, 2011. ACM.

[24] A. Tootoonchian and Y. Ganjali. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[25] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem. Oblivious routing for fat-tree based system area networks with uncertain traffic demands. *SIGMETRICS Perform. Eval. Rev.*, 35(1):337–348, June 2007.