Hybrid Networking Towards a Software Defined Era

Christian Esteve Rothenberg

June 5, 2013

Contents

Chapter 1

Hybrid Networking Towards a Software Defined Era

While Software-Defined Networking (SDN) has been an increasing buzz word with diverse interpretations, the first SDN products [4, 32, 33] based on the OpenFlow protocol – the original trigger of the SDN term [20] – have seen the light in 2012 targeting virtual data center networks for the enterprise and cloud service providers. In addition, the wide area network (WAN) has been targeted by Google's inter-datacenter network design based on OpenFlow to program (in-house built) 100G equipment with the forwarding information base generated from open-source routing stacks (Quagga IS-IS/BGP) augmented with a centralized traffic engineering application [19]. The result is a cost-effective, predictable, highly automated and tested network that reaches unprecedented levels of utilization.

The first SDN deployments have been mainly "greenfield" scenarios and/or tightly controlled single administrative domains. Initial roll-out strategies are mainly based on virtual switch overlay models or OpenFlow-only network-wide controls. However, broader adoption of SDN beyond data center silos – and between themselves – requires considering the interaction and integration with legacy control planes providing traditional switching, routing, and Operation, Administration and Management (OAM) functions. Certainly, rip-and-replace is not a viable strategy for broad adoption of new networking technologies.

This chapter introduces the motivation and problem statement of hybrid networking models in OpenFlow/SDN and discusses different approaches to combine traditional control plane functions with OpenFlow control (Section 1.1). Hybrid networking in a SDN scenario should allow deploying OpenFlow for a subset of all flows only, enable OpenFlow on a subset of devices and/or ports only, and overall provide options to interact with existing OAM protocols, legacy devices, and neighbouring domains. As in any technology transition period where fork-lift upgrades may not be a choice for many, allowing for migration paths is critical for adoption. We introduce the design and implementation of two SDN control platforms that aim at addressing requirements of hybrid networking deployments.

While the LegacyFlow project (Section 1.2) aims at turning legacy devices into OpenFlow-capable nodes in the topology available to the controller, the RouteFlow project (Section 1.3) bridges the gap with traditional IP routing stacks. These proposals allow for transitioning opportunities from existing networks to OpenFlow/SDN. We will discuss how OpenFlow direct forwarding information base (FIB) manipulation can help IP routing control applications and enable cost-effective routing architectures by reducing the number of L3 control units and removing them from the devices themselves to centralized clusters. Details on the current implemented prototypes will be provided in addition to results from experimental evaluation work (Section 1.4). We will conclude with final remarks and a a glimpse on our future work (Section 1.5).

1.1 Hybrid Networking in OpenFlow/SDN

The concept carried by the word *hybrid* spans several levels. The Hybrid Working Group of the Open Networking Foundation (ONF) [34] has a special focus on hybrid switch architectures. The goal is to have an equipment that can be configured to behave as a legacy switch or as an OpenFlow switch – in some cases, as both at the same time. This duality can be achieved, for example, by partitioning the set of ports of a switch, where one subset is devoted to OpenFlow controlled networks and the another subset to legacy networks. For these subsets be active at the same time, each one having its own data-plane, multi-table support at the forwarding engine (e.g. via TCAM partitioning) should be a requisite.

Besides port-based partitioning, it is also possible to have VLAN-based (prior to entering the OpenFlow pipeline), or flow-based partitioning using OpenFlow matching and the `LOCAL` and/or `NORMAL` actions to redirect packets to the legacy pipeline or the switch's local networking stack and its management stack. Flow-based partitioning is the most flexible option and allows each packet entering a switch to be classified by an OpenFlow flow description and treated by the appropriate data-plane (OpenFlow-controlled or legacy-controlled).

The ONF Hybrid Working Group is currently discussing the different models, requirements and use cases for a hybrid programmable forwarding plane (HPFP). One architectural approach for hybrid switches is called *Ships in the Night* (SIN). As its name suggests and similarly how the term has been used in the traditional IP networking domain [21], in the SIN architecture there is no interaction of OpenFlow and legacy control-planes, and the respective data-planes are also isolated. Thus, there is no need to synchronize states between control or management planes. It can be considered a conservative and practical strategy for a migration path of network switch architectures, where new OpenFlow functionalities are incorporated to existing legacy equipments. The clear decoupling of OpenFlow and legacy control-planes simplifies operation and makes each control-plane be unaware of the other, like "old ships in the night". The approach is not free of issues and some use cases deserve special attention, for example, when using tunnel interfaces, logical interface of the tunnel and the physical one used as ingress/egress port may not be managed by the same control-plane.

The ONF Hybrid WG is also discussing about the integration of OpenFlow and legacy control-planes of a hybrid switch. Several aspects must be considered to enable this kind of integration, one of them is how to deal with shared resources between control planes, like ports, tables or meters. Safe operations of this integrated hybrid model require local resource arbitration, or an external orchestration, to manage the conflicts, integrate control states and resources. Notifications from legacy data-plane to OpenFlow controller is another important and useful aspect to be considered, and also communication between both control planes.

The best-known public implementation example of a software-based OpenFlow switch is Open vSwitch (OVS). Generally deployed as a software switch running on top of a Linux kernel, it can be configured to act like an OpenFlow-only or a hybrid switch, supporting the SIN architecture with per-VLAN and per-port segregation. OVS has been also ported to hardware-based switches. HP Procurve has SIN implementation based on OVS with per-port, per-VLAN and per-flow segregation. Pica8 hardware switches also use OVS, with their latest product versions supporting OpenFlow version 1.2. NEC Programmable Flow Switch (PFS) has a cascade model of integration, where packets are treated by the OpenFlow control-plane then directed to the legacy control-plane, without any other integration of both control-planes. The NEC PFS has several shared resources, like TCAM, buffers and queues.

Among use cases involving hybrid switches, one is related with the need to complement OpenFlow switch capabilities to make this (still evolving) technology fully compatible with current networks in operation. One example is Quality of Service (QoS) policing, which is not completely defined on the earliest OpenFlow protocol versions and implementations. The opposite rationale for hybrid use cases also apply: it may be useful to have an OpenFlow switch complementing the capabilities of a legacy switch, for example, adding flexibility for rules matching. Another example related to hybrid switches with shared resources is the use of legacy mechanisms to set up tunnels and the OpenFlow control-plane to decide which packets are directed to which tunnel.

When talking about *hybrid networks*, as already discussed, the focus is on how to coexist OpenFlow-enabled network devices with legacy ones. In such concept, a hybrid switch can be very useful if, for instance, it can hold both data-planes active at the same time and provide ways to intercommunicate both control-planes. So, there are some different approaches that can be used in order to interconnect OpenFlow and legacy “amount” of a network. On a hybrid network, a flow can traverse distinct unsynchronized data-planes, each of them governed by OpenFlow or legacy distinct control-planes. This transition between data-planes can occur within a single switch or through different switches of a network.

From this perspective, there are number of ways to define the meaning of *hybrid network*, where hybrid is the co-existence of traditional environments of closed vendor’s switches (and routers) with new OpenFlow-enabled devices. However, for very clear decoupling as well as for avoiding any upgrade of existing devices, the hybrid approach here focused refers to the interconnection of both control and data planes of legacy and new OpenFlow network elements.

1.1.1 The Emergence of Hybrid Networks

In a general sense, in any network of any kind there is a need to establish a communication between different nodes in order to achieve a path (hopeful the best one) between an origin-destination pair. Recall that, in current networks, there are basically two ways to establish a path: by the use of a distributed protocol running at each node or by direct configuring the data-plane at each node along a chosen path. With respect to distributed protocols, for example, we have routing protocols, like OSPF, ISIS, BGP etc, or MPLS signalling. With respect to direct configuration of the data-plane, it is possible to manually “stitch” VLANs through the network or use a GMPLS-based approach [27].

From a SDN perspective, hybrid networks need to be discussed based on the major characteristic of OpenFlow-enabled networks, which is the inherent external, typically centralized, control of network nodes. This perspective induces at least three major approaches to be used for hybrid networking:

1. Make each OpenFlow node behave as a router, so it does not matter if such node is connected to another OpenFlow node or to a legacy router, supposing the legacy nodes

understand the respective L3 protocol;

2. Make each OpenFlow node understand circuit technology signalling to establish LSPs, supposing other legacy nodes along the path also understand the same signalling protocol;

3. Use OpenFlow-based control-plane for the whole hybrid network and adopt an intermediate layer to translate respective OpenFlow forwarding rules to the configuration syntax of each legacy non-OpenFlow switch.

With respect to OpenFlow controllers, the following considerations arise:

- For each approach above, OpenFlow domains may be controlled by one or more OpenFlow controllers, so subsets of the whole OpenFlow domain are permitted.
- For the approach # 3, consider OpenFlow the uniform way to control the whole hybrid network and allow multiple OpenFlow controllers.
- It is possible to plug solutions # 1 or # 2 on top of solution # 3, thus the complete hybrid network will act like a pure OpenFlow network doing L3 routing or MPLS signalling.

The advantage of the latter compound approach (# 1 or # 2 on top of # 3), relies at the specificities of the possible implementations of solutions # 1 and # 2. As both are OpenFlow-based, decisions on how to establish IP routes or LSPs can be centralized, which is interesting for a network operation/management point of view. Another interesting issue is related to services that can be offered on top of these solutions, like the Routing as a Service (RaaS) concept [7, 24]. For instance, RaaS can be engineered based on an OpenFlow architecture and be applied to a hybrid network in which legacy devices are supported by solution # 3. Therefore, this approach offers a high-level interface for the description of routing services. Related to this scenario, the IETF has recently started a discussion list on an Interface to the Internet Routing System (I2RS) [25], that aims to improve interactions between the underlying routing system and network control or management applications. Another IETF activity worth to mention that allows for control split architectures is Forwarding and

Control Element Separation (ForCES) [45], a framework that started long before OpenFlow but focused on the specifics of L3 devices, where as OpenFlow devoted its initial pragmatic efforts to re-use TCAM capabilities in existing devices to offer a flat (L1-L4) flow abstraction programmable via a small instruction set.

Approaches # 3 and # 1 above described are respectively implemented by two OpenFlow-based architectures that will be further detailed in this chapter: LegacyFlow and RouteFlow. LegacyFlow extends the OpenFlow-based controlled network to embrace non-OpenFlow nodes. RouteFlow implements an IP level control-plane on top of an OpenFlow network, making the underlying nodes act like routers, with different arrangement possibilities.

The common ground of these pieces of work is (*i*) considering *hybrid* as the co-existence of traditional environments of closed vendor's routers and switches with new OpenFlow-enabled devices, (*ii*) targeting the interconnection of both control and data plane of legacy and new network elements, and (*iii*) taking a controller-centric approach, drawing the hybrid line outside of any device itself, but into the controller application space. Note that this approach is fundamentally different than a "ships in the night" strategies where legacy and new control plane interconnect and co-existence are device-centric and not controller-centric.

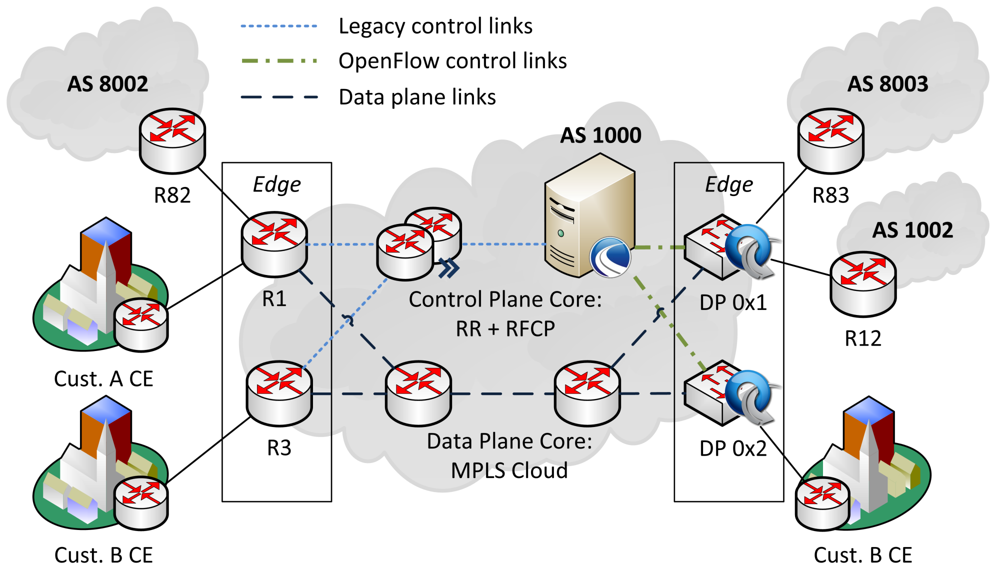


Figure 1.1: Routing architectures: Current (left) vs. OpenFlow-based controller-centric hybrid SDN approach (right).

As an example of this deployment scenario for the case of RouteFlow, Figure 1.1 illustrates the new control plane added in the form of a BGP controller that acts as a gateway between existing route reflectors (RRs) and OpenFlow controllers programming the datapaths devices. Legacy neighboring routers close their (eBGP) control plane sessions with the routing engines running in centralized servers but effectively acting as if they were in the edge devices themselves, which are reduced to plain packet forwarding as instructed via OpenFlow, the protocol used to download the FIB and to serve as the indirection point and encapsulation vehicle for the control plane traffic.

1.2 LegacyFlow: Leveraging the legacy switching infrastructure

The OpenFlow protocol allows production networking, such as campus networks, metropolitan networks or Research&Development (R&D) networks, to experiment with so-called *Future Internet* software, protocols, and architectures in parallel to production traffic. However, during roll-out, there are practical problems that arise when dealing with legacy elements or complete networks that do not support OpenFlow. Equipment replacement or upgrades involve high costs of re-engineering and are often not feasible at all. In practice, the successful deployment of such a cutting-edge networking is far from seamless [10].

The overarching problem we aim at addressing in the LegactFlow project revolves around the question of *how to integrate OpenFlow with legacy networks?*

1.2.1 Challenges Between OpenFlow and Legacy Networks

We start by conceptualizing *Legacy Networks* as networks that either are composed of non-OpenFlow equipments (e.g., the current Internet) or technologies that are currently not supported by the OpenFlow protocol (e.g., L1-L0 circuit technologies).

Legacy networks will always be a levee in the integration or connection among these new infrastructures (R&D experimental facilities or OpenFlow networks). Hence, it is important to understand some of the underlying problems between this two worlds, legacy and OpenFlow networks, including: (i) integration of OpenFlow and legacy networks, (ii) convergence of transport technologies not supported by OpenFlow into a unified control and management of legacy equipment by OpenFlow. The former challenge is related to the *compatibility* requirements with legacy network protocols. The latter issue is OpenFlow being unable to handle or manage legacy equipment by using the same OpenFlow protocol in a unifying approach. As a result of these two main challenges, the connection of multiple OpenFlow environments through legacy networks is very difficult.

Another important issue to the success of OpenFlow is its extensibility to support relevant layer 1 and layer 0 circuit technologies of legacy networks, such as MPLS, GMPLS, SONET/SDH or WDM. On the other hand, new OpenFlow version as 1.2, it allows the inclusion of new match conditions and actions, but it is still unclear how the control capabilities negotiation between transport elements and controller should happen in addition to a multi-version OpenFlow operation.

Integrating OpenFlow and Legacy Networks

In the current state-of-the-art, the integration of different OpenFlow islands requires different engineering and handcrafted strategies on the legacy network devices and their protocols. A transparent and automatic manner is still an unresolved issue, because most of the time there is no legacy protocol emulation application, on OpenFlow controller-based, (e.g., ARP replier, LLDP) truly compatible with the distributed protocols running in the legacy equipment. Figure 1.2 illustrates an example of such a mismatch in the case of link layer topology discovery. The OpenFlow Discovery Protocol (OFDP) [12] is an artificial term[1] of how the Link Layer Discovery Protocol (LLDP) is being leveraged with subtle modifications to perform topology discovery in an OpenFlow network. OFDP provides a controller-driven means to finding out neighbouring OpenFlow switches.

Scenario A shows two OpenFlow domains using the OpenFlow LLDP implementation to find out links between domains. In this case, the discovery works well because all OpenFlow equipment implement the same protocol code (e.g., Ethertype set to $0x88cc$). However, in Scenario B, there is a legacy domain between the OpenFlow domains, and the OpenFlow discovery process fails due to the legacy switches do not correctly support the OpenFlow LLDP code. Thus, the LLDP packets are dropped and the link between the OpenFlow domains is not detected.

Support of Legacy Circuit Technologies

Another common problem when dealing with legacy networks is the support of popular circuit technologies. A practical example of the legacy network is the Internet backbone, which

[1] http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

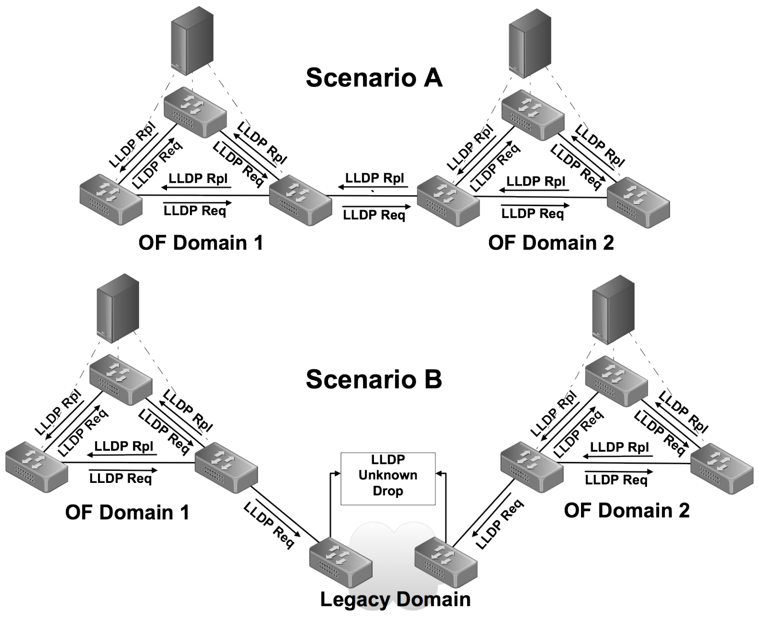


Figure 1.2: A OpenFlow and legacy network scenario

comprises a wide range of transport technologies. Current OpenFlow specifications do not offer consistent support of circuit-based technologies, such as MPLS, GMPLS, SONET/SDH or WDM. Since these technologies are essential elements on operational networks, they should be somehow embraced, since an equipment rip-off is may not be wise or even possible. Therefore, The question is *how to absorb these technologies into an unifying architecture?*

The problem is that the majority of these technologies do not offer essential features for architectural evolution or virtualization based on slicing and programmability. For example, GMPLS is considered a conservative protocol suite, which leaves a little room for innovation because of the complexity of integrating new features in the distributed control plane. On the other hand, OpenFlow can be easily virtualized along different dimensions (switches, flow spaces, controllers, etc.), while allowing the network control layer to be sliced and the system performance improved [9].

Management of Legacy Vendor Equipment

We now turn our attention to examining legacy equipment, such as switches, routers, and optical devices. Issues about whether or not (and where) to use OpenFlow are common concerns among network and system administrators these days.

Despite being a costly proposition, the ecosystem is not ready to replace every networking gear to turn the network 100% OpenFlow-enabled. Some devices are paramount in the production network and currently there is no equivalent OpenFlow-enabled equipment for every network device, for instance 100G core packet switches, expensive ROADM optical switches, or very low latency legacy switches. Hence, it becomes necessary to find means of using the OpenFlow without losing compatibility with the legacy equipment.

1.2.2 Design and Architecture

LegacyFlow proposes a hybrid model that focuses on the three main issues presented earlier: 1) integration of OpenFlow with legacy networks, 2) (re)use of legacy technologies, and 3) management of legacy equipment. Some advantages of our approach include the reuse of existing equipment in operation, reducing the cost of OpenFlow deployment, offering ways for gradual deployment, while contributing to the capabilities of an unifying OpenFlow control plane.

The first step in the project was to look for a “merge point” between the two networks (legacy and OpenFlow) without weakening the key capabilities of programmability and network virtualization. One requirement is being able to manage the legacy equipment via OpenFlow but without the chance of developing an OpenFlow agent running inside the control plane of the switch.

The adopted model defined in the LegacyFlow framework introduces a new virtual datapath component in the form of a software switch that acts as the control *glue* between OpenFlow and the Legacy network. Acting as a proxy to OpenFlow actions, it translates the OpenFlow rules to the equivalent operations available in the underlying legacy equip-

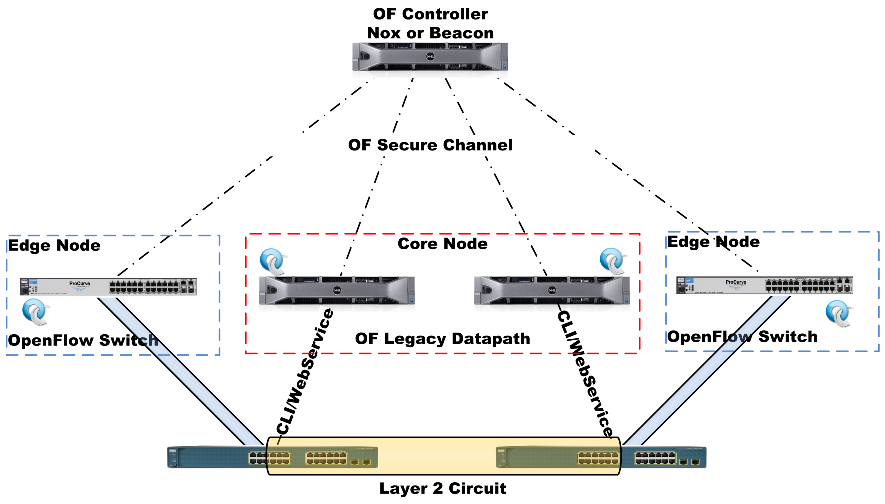


Figure 1.3: A view of the LegacyFlow solution

ment using interfaces, such as CLI, WebService or SNMP. Of course, this operation mode imposes some limitations to the type of actions supported by the devices. On the other hand, it allows us to do exposing to OpenFlow of some switch features not supported in the standard protocol (e.g. Lambda switching).

One fundamental restriction of this approach is sacrifying the reactive mode of operation of OpenFlow, which packets without a matching rule are forwarded to the controller via `packet-in` events. At this current stage, LegacyFlow does not support this type of send-to-controller actions in non-OpenFlow devices. Hence, these legacy devices will be generally used in network segments that inter-connect OpenFlow switches located at the ingress or egress point of a flow. Figure 1.3 illustrates edge switches that are fully OpenFlow-enabled and support thus capable of forwarding packet to the controller, either as an explicit flow action or as an implicit no-match switch configuration.

For instance, when a new flow enters a LegacyFlow network, the first packet can be recognized by the edge OpenFlow device and sent to the controller via a `packet-in`. In turn, the controller installs the required flow rules on the datapaths along the path, for instance creating a tunnel or circuit based on VLAN ID throughout the legacy and OpenFlow equipments. The circuit technology could not be limited to VLAN, but MPLS/GMPLS LSP (Label Switching Path) or WDM (Wavelength Division Multiplexing) light paths.

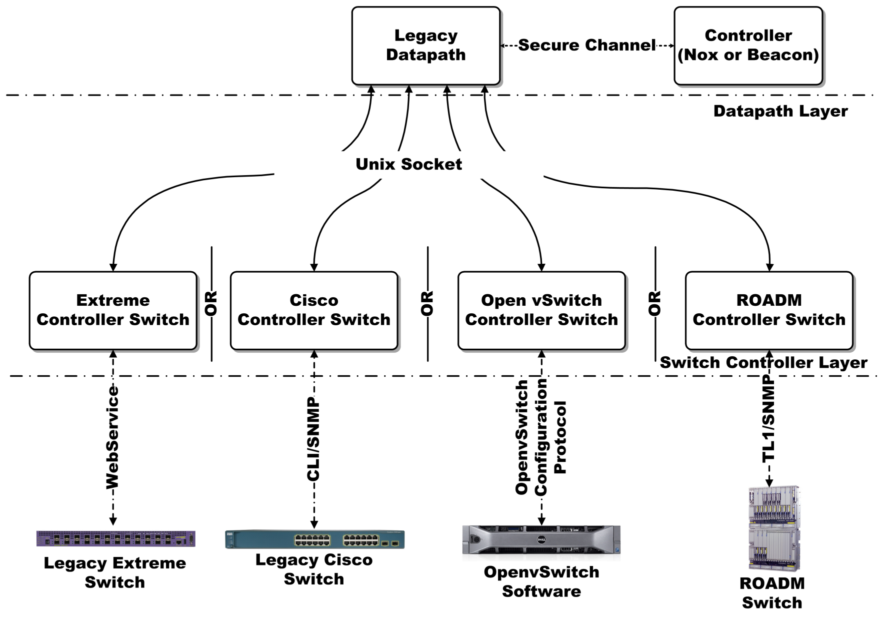


Figure 1.4: The LegacyFlow Architecture

Architecture

The LegacyFlow architecture shows in Figure 1.4 is divided in two layers: 1) Datapath and 2) Switch Controller.

In the **Datapath Layer** (DL) there is a new software switch (LegacyFlow datapath) that (i) communicates with the controller via the OpenFlow protocol, and (2) provides a bridge of access of different features from the legacy equipment to controller.

The **Switch Controller Layer** (SCL) provides the configuration module for each legacy equipment (e.g., switch, router or optical switch) along the information related to the available access method depending on each vendor, commonly WebService, Telnet/CLI, and SNMP. The communication between DL and SCL is based on a client/server IPC. This modular approach keeps independent the development on the Legacy Datapath and the Controller Switch, allowing a simpler integration of new modules to the LegacyFlow architecture.

The link between the Legacy Datapath and Controller Switch is point-to-point – each Legacy Datapath is responsible for a kind of Controller Switch. Currently, the LegacyFlow

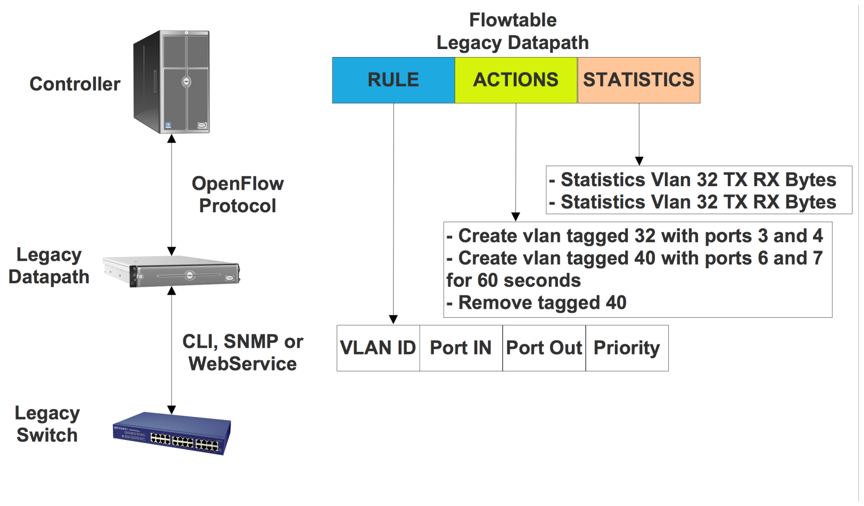


Figure 1.5: LegacyFlow Datapath

supports the legacy equipment of the following vendors: Extreme and Cisco. At this point of writing there is ongoing development for Juniper routers and ROADM optical equipment.

LegacyFlow Datapath

The LegacyFlow Datapath (LD) acts as a proxy receiving OpenFlow commands from the controller and applying the corresponding actions in the real switch. Each legacy switch (part of the hybrid OpenFlow network) is assigned to a LD that runs in a guest machine (real or virtual) with GNU/Linux OS, as shown in Figure 1.5.

From an outside point, the LD represents some information about the features of the legacy switches, such as port numbers, link speed, sent-and-received packets rate, among other capabilities specified in OpenFlow. At the same time, it translates the proactive installation of OpenFlow entries into the corresponding switch commands whenever possible, returning an error if the match field or action is not controllable via the vendor-specific APIs.

To initialize an OpenFlow datapath it is necessary to pass through the parameters of the network interface. Likewise, the LD must be started with information of the switch interfaces. As shown in Figure 1.6, the LegacyFlow Virtual Interface (LVI) is a Linux virtual interface module that represents a switch port of the LD. The LD is initialized by creating virtual network interfaces according to the number of switch ports of and their features, mirroring from the real switch characteristics such as link speed, maximum transmission unit (MTU) size, among other values. Depending on how the switch offers this information, translation from the switch to the virtual interfaces via SNMP or WebService.

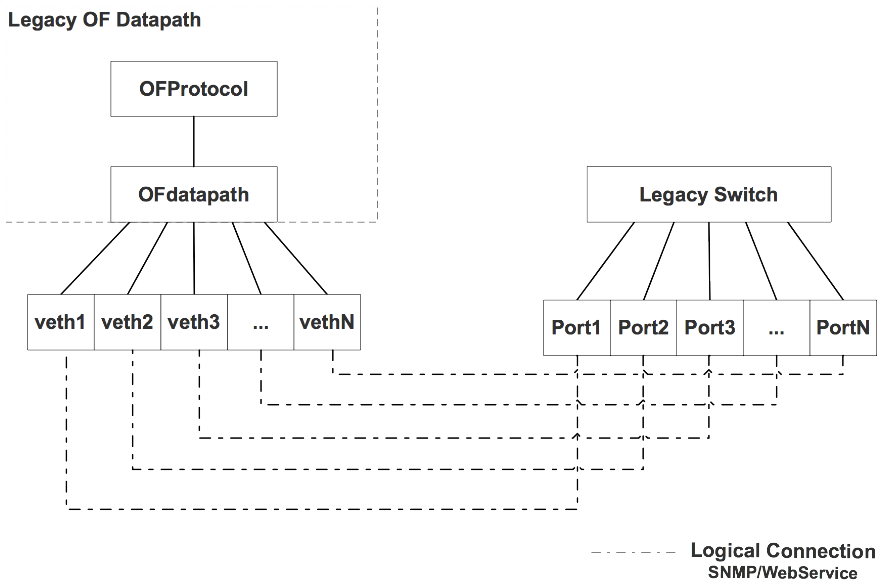


Figure 1.6: Connection between LVI and Legacy Switch

The LD module allows the controller to acquire a view of legacy equipment and features in a similar to way to OpenFlow equipment, managing the state and collecting statistics in a unified way. Currently, the LVI can represent Layer 2 interface such as: MAC address, Speed, Link Status, Name and Modes. In the future, it will be able to represent layer 1 port information (WDM) such as Name, Wavelength and Bandwidth.

LegacyFlow Controller Switch

The Legacy Controller Switch (LCS) is set of applications used to configure of the legacy equipment. The communication between the LD and LCS is based on an IPC where the LD is a client and LCS is a server. This approach facilitates the integration of new LCS independently from the programming language or the configuration interface (i.e. SNMP, NetConf, WebService or Telnet/CLI).

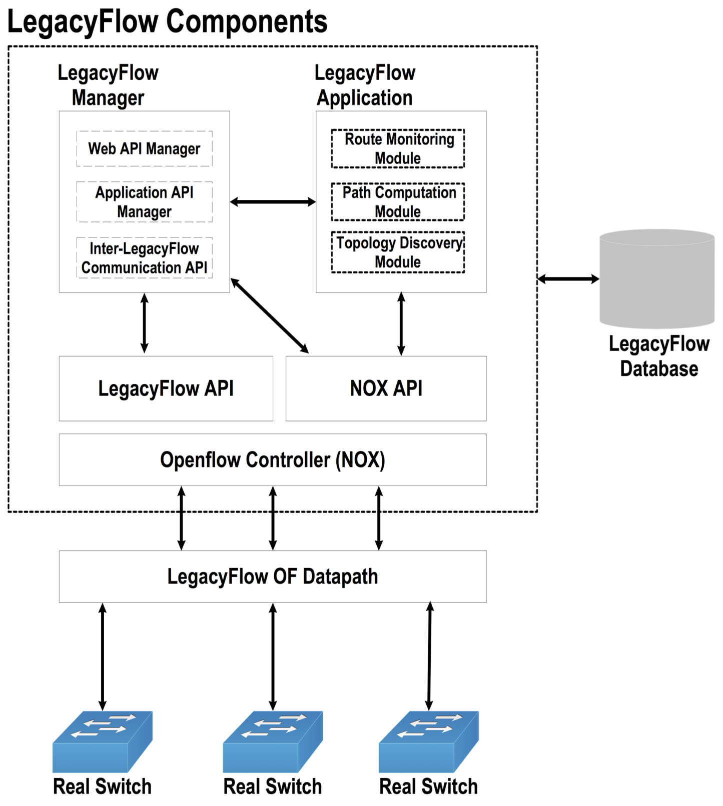


Figure 1.7: Connection between LVI and Legacy Switch

An out-of-band control channel between the LCS and the legacy equipment is used to send configuration commands. The port management equipment is generally used to establish this communication. Each LCS is limited to handling just one piece of legacy equipment. The information about the equipment being managed is obtained from the configuration file during the initialization process of the LD.

LegacyFlow Controller Components

The LegacyFlow Controller Components (LCC) refers to the set of applications that integrate to the OpenFlow controller. These components are necessary to keep a transparent combination of the OpenFlow and the legacy network. Currently, the components are developed using NOX API and can be divided into three parts: LegacyFlow API; LegacyFlow Application; LegacyFlow Manager; and LegacyFlow Database. Figure 1.7 illustrates a high-level view of the LLC.

The LegacyFlow API is responsible for offering access interfaces to the LegacyFlow actions. These new actions are only recognized and executed on the appropriate LD. The LegacyFlow Manager (LM) is a component developed to manage the communication with other elements such as as web GUI, helper applications, or other LegacyFlow controller instances. The three main sub-components are:

- Web manager API. It interacts with the WEB GUI based on HTTP+JSON.
- Application manager API. It provides interfaces to helper applications such as route calculation, topology management, monitoring, etc.
- Inter-LegacyFlow communication API. It manages the communication between LCCs located in other OpenFlow domains and offers resource allocation, topology exchange, and resource updates.

The LegacyFlow Application assists the LCC on decisions such as policies, routes, or topology, common applications developed by the network administrator. Currently, the following applications are available: Route Monitoring to track created circuits or paths; Path Calculation to compute routes between source and destination OpenFlow switches; and Topology Discovery to maintain the topology formed by legacy switches.

Finally, the LegacyFlow database (LDB) stores every information created by LCC, including circuits, interface statistics, or monitoring. Today we use a No-SQL database that provides JSON APIs.

1.2.3 Use Case Operations

We now illustrate the LegacyFlow in operation in the case of a circuit path among legacy switches based on VLAN ID.

The activity diagram is presented in Figure 1.8. Initially, the LD is started and receives as mandatory parameter the LD switch model describing the available communication protocols. The LVI is initiated by creating virtual interfaces on the operational system according

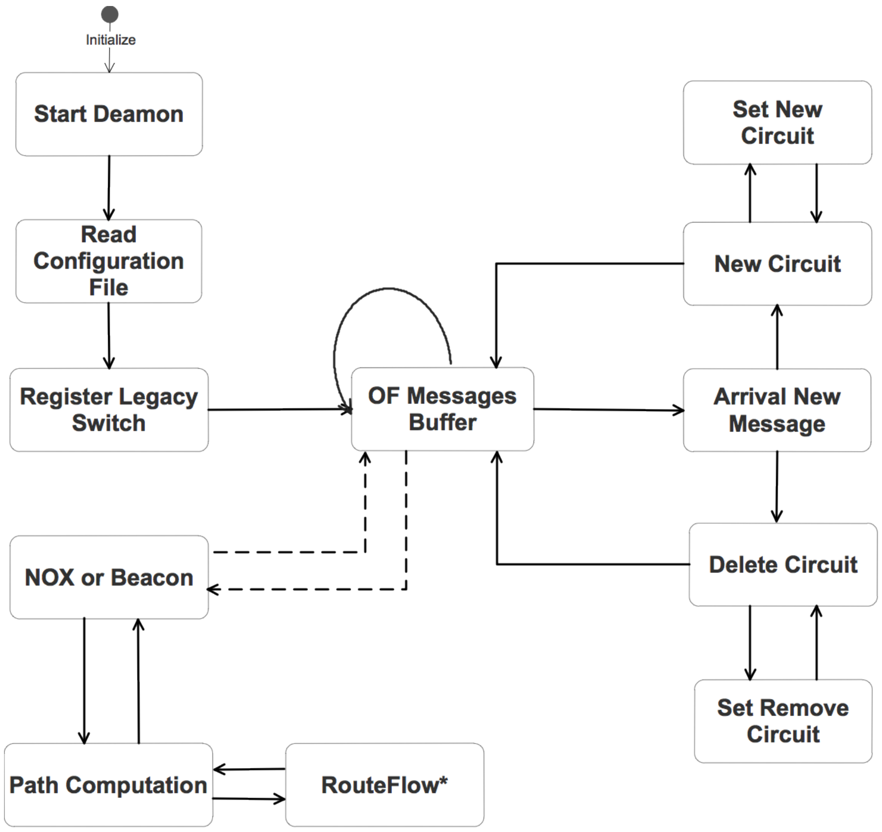


Figure 1.8: LegacyFlow Datapath

to the ones available in the switch. Using an outer and dedicated out-of-band channel between the switch and the LD, SNMP connections are created to collect information of the switch interfaces and apply them to the virtual interfaces. Currently, this state is updated every 3 seconds to keep information relatively accurate.

After that the switch is registered with the LD, one vendor-specific switch controller is initialized to manage the real switch using an inter-process pipeline. Following the setup phase, the interfaces are connected to the LD, which can start receiving flow commands from the controller. OpenFlow messages received in the LD are interpreted and checked for compatibility with actions supported by the legacy switches. The OF message buffer is responsible for receiving all OpenFlow protocol messages and handling them to the corresponding actions such as delete or create circuit. When the action is called a specific inter-process call is executed. In this example, these actions are the configuration counterparts to remove or create VLANs on the real switch, using parameters such as: ingress port, egress port and

VLAN ID.

When a flow packet gets into a qualified OpenFlow switch, the controller identifies the source and destination of the flow and uses the path computation module to calculate one path among the Legacy switches. When the circuit is established, the flow is forwarded to the destination.

1.2.4 Concluding Remarks and Work Ahead

The LegacyFlow efforts continue towards improving a gradual deployment options for OpenFlow in production environments. The architectural proposal prioritizes reusing the legacy equipment without changing the entire infrastructure. The approach introduces OpenFlow datapath abstractions per legacy device following a proxy model that translates OpenFlow commands into vendor-specific control and configuration equivalents. Dynamic circuits across legacy switches are used to bridge the connectivity among OpenFlow-enabled devices.

Similar work on dynamic circuits is being done in the GENI experimental network and the 'stitching' architecture based on VLANs. The ongoing activities towards a Slice Federation Architecture includes investigations into the interactions with the controller proxy FlowVisor that allows to share the programmable infrastructure among multiple controllers to realize isolated virtual networks.

As for the future, there are multiple lines of work that we would like to pursue. One front seeks to adapt LegacyFlow to inter-work with DCN (Dynamic Circuit Network) similar to IDC/OSCARS [35], DRAGON [11] or AUTOBAHN [3] – the shared goal being on-demand dynamic circuit management between OpenFlow domains and/or testbed facilities. We will also investigate the applicability of LegacyFlow will as a tool to create circuits utilizing multi layer technologies and handling expected issues of a unified control plane such as protection and restoration, standardization, and so on. Further integration work is likely to be required in addition to extensibility of the components to embrace more legacy technologies and additional features of the OpenFlow framework. To strengthen the architecture performance measurements need to be undertaken to stress test the different components. Finally, experimental evaluation work on the feasibility and effectiveness of the resulting artifact will be pursued within the facilities of the FIBRE federated testbed facilities [15].

1.3 RouteFlow: When IP routing meets OpenFlow

RouteFlow was born as a *Gedankenexperiment* ("thought experiment") on whether the Linux-based control plane embedded in a 1U merchant silicon Ethernet switch prototype could be run out of the box in a commodity server with OpenFlow being the solely communication channel between data and control planes. Firstly baptized as QuagFlow [31] (Quagga + OpenFlow), the experiment turned out to be viable in terms of convergence and performance when compared to a traditional lab setup [30]. With increasing interest from the community, the RouteFlow project emerged and went public to serve the goal of gluing together open-source routing stacks and OpenFlow infrastructures.

Fundamentally, RouteFlow is based on three main modules: the RouteFlow client (RFClient), the RouteFlow server (RFServer), and the RouteFlow proxy (RFProxy).[2] Figure 1.9 depicts a simplified view of a typical RouteFlow scenario: routing engines in a virtualized environment generate the forwarding information base according to the configured routing protocols (e.g., OSPF, BGP) and ARP processes. In turn, the routing and ARP tables are collected by the RFClient daemons and then translated into OpenFlow tuples that are sent to the RFServer, which adapts this FIB to the specified routing control logic and finally instructs the RFProxy, a controller application, to configure the switches using OpenFlow commands.

This approach leads to a flexible, high-performance and commercially competitive solution, at this time built on available resources such as: (a) programmable low cost switches and small-footprint embedded software (i.e. OpenFlow); (b) a stack of open-source routing protocols (e.g. Quagga); and (c) commodity x86 server technology.

The project counts with a growing user base worldwide (more than 1,000 downloads and more than 10,000 unique visitors since the project start in April 2010). External contributions range from bug reporting to actual code submissions via the community-oriented GitHub repository. To cite a few examples, Google has contributed with an SNMP plug-in

[2]As a historical note, the first QuagFlow prototype implemented RFServer and RFProxy as a single NOX application. After the separation (in the first RouteFlow versions) RFProxy was named RouteFlow controller. This caused some confusion, since it actually an application on top of an OpenFlow controller, so we renamed it. Its purpose and general design remain the same.

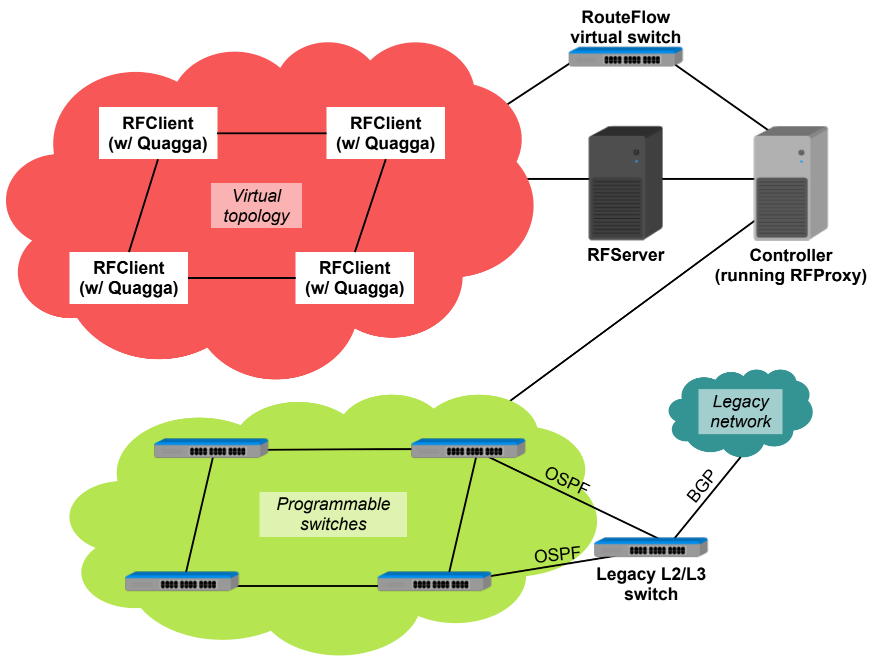


Figure 1.9: A typical, simplified RouteFlow scenario

and is currently working on MPLS support and new APIs of the Quagga routing engine maintained by the ISC Open Source Routing initiative. Indiana University has added an advanced GUI and run pilots with HW switches in the US-wide NDDI testbed. Unirio has prototyped a single node abstraction with a domain-wide eBGP controller. Unicamp has done a port to the Ryu OpenFlow 1.2 controller and is experimenting with new data center designs. While some users look at RouteFlow as “Quagga on steroids” to achieve a HW-accelerated open-source routing solution, others are looking at cost-effective BGP-free edge designs in hybrid IP-SDN networking scenarios where RouteFlow offers a migration path to OpenFlow/SDN [39]. These are ongoing examples of the power of innovation resulting from the blend of open interfaces to commercial HW and open-source community-driven SW development.

1.3.1 Architectural details

The current RouteFlow architecture is an evolution from previous prototype designs to a better layered, distributed system, flexible enough to accommodate different virtualiza-

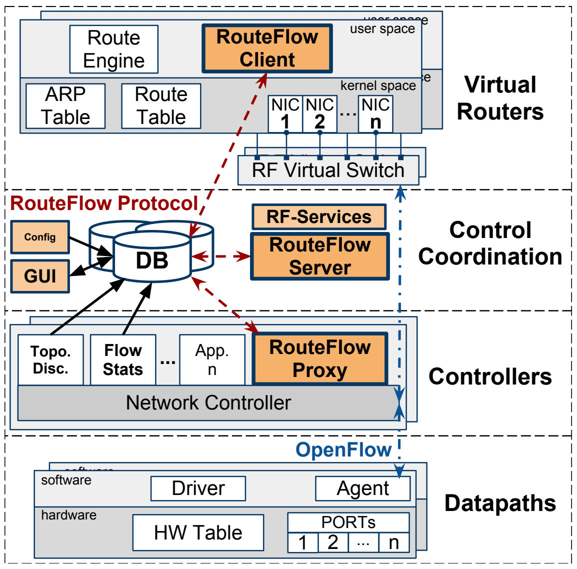


Figure 1.10: Architecture design of the RFCP.

tion use cases ($m : n$ mapping of routing engine virtual interfaces to physical OpenFlow-enabled ports) and ease the development of advanced routing-oriented applications. Past re-architecting efforts have attempted to solve problems revealed during the first year of the public release including feedback from third party users and lessons learned from demonstrations using commercial OpenFlow switches.[3] The main issues addressed include configurability, component flexibility, resilience, easy management interfaces, and collection of statistics.

Anticipating the need for updating (or even replacing) parts of the architecture while facilitating multi-controller support, the implementation is segregated into the following three components (cf. Fig. 1.10):

· **RFClient:** Collects routing and forwarding information generated by the routing engine (e.g. Quagga [37]) of the Linux system,[4] where it runs as a user-space daemon. Optionally, to extract additional routing information (e.g. all BGP paths in the RIB-in), it hooks into or peers (e.g. iBGP) with the routing engine(s).

[3]Open Networking Summit I (Oct/2011) and II (Apr/2012), Super Computing Research Sandbox (Nov/2011), OFELIA/CHANGE Summer School (Nov/2011), Internet2 NDDI (Jan/2012), 7th API on SDN (Jun/2012). See details on: https://sites.google.com/site/routeflow/updates

[4]Typically, a lightweight virtual container like LXC [26].

· **RFServer:** Standalone application responsible for the system's core logic (e.g., event processing, VM-to-DP mapping, etc.). RFCP Services are implemented as operator-tailored modules which use the knowledge information base to deliver arbitrary, high-level routing logics (e.g., load balancing, preferred exit points, etc.).

· **RFProxy:** Simple 'proxy' application on top of an OpenFlow controller (e.g., NOX, POX) which serves the RFCP with switch interaction and state collected from topology discovery and monitoring applications.

In line with the best design practices of cloud applications, we opted for a scalable, fault-tolerant datastore that centralizes (i) the RFCP core state (e.g., resource associations), (ii) the network view (logical, physical, and protocol-specific), and (iii) any information base (e.g., traffic histogram/forecasts, flow monitoring, administrative policies) used to develop routing applications. Hence, the datastore embodies so-called Network Information Base (NIB) [23] and Knowledge Information Base (KIB) [8].

In addition to acting as the back-end storage, the distributed NoSQL database of choice (e.g., MongoDB, Redis, Cassandra) is used as the pubsub-like message queuing IPC that loosely couples the modules via an extensible JSON-based implementation of the RouteFlow protocol. Without sacrificing performance, the datastore-based IPC easies fault-management, debugging, and monitoring by natively keeping a history of the RouteFlow workflow allowing for replay or catch-up operations. Noteworthy, the IPC implementation (e.g., message factory) is completely agnostic to the DB of choice, should we change this decision.[5]

Altogether, the design tries to follow principles that allow architectural evolvability [17]: layers of indirection, system modularity, and interface extensibility.

[5]While some may call Database-as-an-IPC an anti-pattern (cf. http://en.wikipedia.org/wiki/Database-as-IPC), we debate this belief when considering NoSQL solutions like MongoDB acting as a messaging and transport layer (e.g. http://shtylman.com/post/the-tail-of-mongodb/).

Messaging workflow

To allow for legacy networks integration, RouteFlow use flow entries to match known routing messages received by the physical infrastructure, handling them into the virtual topology. Conversely, pertinent routing messages originated in the virtual topology are sent through the physical infrastructure. For example, adjacent Open Shortest Path First (OSPF) routers instantiated in the VMs will have their `HELLO` messages send 'down' to the OpenFlow switches and forwarded out via the corresponding physical ports. In turn, the OSPF messages are received via the physical ports of the adjacent OpenFlow switch and will match a flow entry based on the OSPF protocol information. The messages will then be sent 'up' via the RouteFlow middleware into the corresponding virtual interface.

After the OpenFlow physical topology is discovered and registered as available resource, the RFServer maps available VMs (interfaces) to OpenFlow switch (ports). The routing service will decide on how this mapping shall be done and allows for different modes of operation (later detailed in Sec. 1.3.2). For instance, in case of an aggregated routing model, a single control plane VM may span multiple switches which are effectively programmed to act as a single router. Each VM runs a routing protocol stack, and is instantiated with as many network interfaces (NICs) as there are active ports in the corresponding OpenFlow domain. The NICs are bound to software switches, through which physical connectivity can be mapped. Routing protocols in the VMs execute as usual and compute their individual FIBs. For each FIB update, the RFClient captures the event and sends an update message to the RFServer. In turn, based on the implemented routing services, the RFServer transforms each FIB update (e.g. route our ARP entry add/remove) into a (number of) flow modification commands that the RFProxies receive and execute by issuing the corresponding OpenFlow protocol flow-mod commands.

In the datapath devices, matching packets on routing protocol and control traffic (e.g., ARP, BGP, RIP, OSPF) are directed by the RFProxy to the corresponding VM interfaces via a software switch [36]. The behavior of this virtual switch[6] is also controlled by the RFProxy and allows for a direct channel between the physical and virtual environments, eliminating

[6]We employ Open vSwitch for this task: http://openvswitch.org/

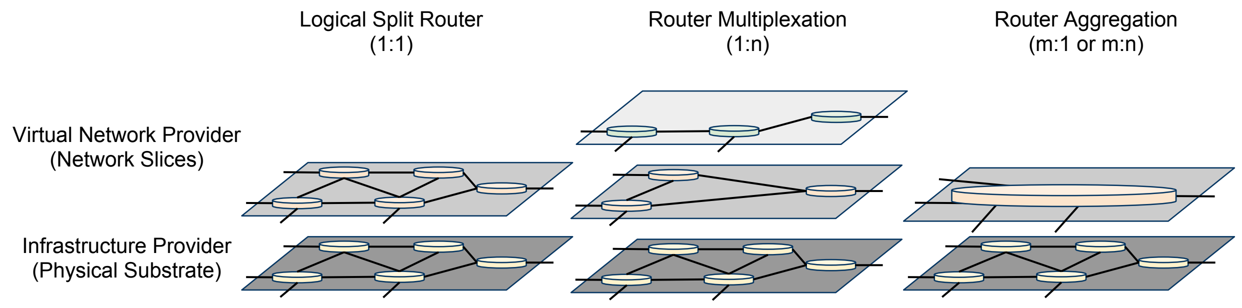


Figure 1.11: Different modes of operation and usage scenarios of router virtualization.

the need to pass through the RFServer and RFClient, reducing the delay in routing protocol messages and allowing for distributed virtual switches and additional programmability. Note that only control plane traffic is sent via OpenFlow to the RouteFlow domain. User traffic is solely forwarded in the data plane via the OpenFlow-compiled FIB state installed in the datapath devices.

1.3.2 Modes of operation

Separating the control plane from the forwarding substrate allows for a flexible mapping and operation between the virtual elements and their physical counterparts. Figure 1.11 shows three main modes of operation that RouteFlow aims at supporting.

Logical split: This 1 : 1 mapping between hardware switches and the virtualized routing engines basically mirrors the physical substrate (number of switch ports, connectivity) into the virtual control plane. Two submodes of operation can be defined depending on whether the routing protocol messages are sent through the physical infrastructure (i.e. traditional routing) or are kept in the virtual plane. The latter allows to separate the problems of physical topology discovery and maintenance from the routing state distribution. This enables an optimization of the routing protocols where fast connectivity maintenance can be run in the data plane (e.g., BFD-like) while route updates (e.g., LSA) can be flooded inside the virtual domain.

Multiplexing: This $1 : n$ mapping of physical to virtual substrate represents the common approach to router virtualization where multiple control planes run simultaneously and install their independent FIBs on the same hardware. Multi-tenant virtual networks can be defined by letting control protocol messages flow through the virtual plane and stitching the data plane connectivity accordingly.

Aggregation: This $m : 1$ mapping of hardware resources to virtual instances allows to simplify the network protocol engineering by bundling a group of switches, such that neighbouring devices or domains can treat the aggregated as if it were a single element. This way intra-domain routing can be independently defined per software while legacy inter-domain or inter-zone routing (e.g. BGP) can be converged into single control unit for signaling scalability and simplified, centralized management purposes.

1.3.3 Protocols and Abstractions

The RouteFlow protocol glues together the different modules with a simple command/response syntax northbound to the RFClients and a subset of the OpenFlow messages southbound to the RFProxy application. As one approach to OpenFlow version polyglotism, the RouteFlow protocol layer abstracts most of the details of the differences from controller implementing OpenFlow versions 1.x and the slightly higher-level RouteFlow APIs. There is no conceptional barrier to support flexible $m : n$ mapping as pursued within the IETF ForCES WG [45] and the definition of Control Elements (CE) and Forwarding Elements (CE), which compound form a Network Element (NE) that logically constitutes a router.

Besides being the API to the datapaths, OpenFlow is used as the vehicle to deliver control plane messages from/to the interfaces of the VMs running the routing engine and the RF-Client. Programming the virtual switches (OVS in Fig. 1.10), we can select a mode of operation where routing protocol messages are sent 'down' to the physical devices or are kept 'up' in the virtual network plane, which can be a reproduction of the discovered physical connectivity or a simplified/arbitrary version (e.g. single router abstractions [22,39]) of the HW resources. In the former, messages follow the physical path, so no additional fault-detection mechanism are required at the cost of extra propagation delays. In the latter, signaling is kept in the virtual domain, benefiting from low latencies and contributing to better scalability. This option provides a more natural support for virtual networks but requires extra programmability and fault-detection extensions to update VMs' logical associations due to physical link changes.

All in all, the resulting software artifact is capable of instructing the SDN data plane with regard to IP forwarding and addresses the following premises:

- Ability to define desired routing criteria (i.e. business policies) in an abstraction level which does not require individual configuration of multiple routing processes;
- An unified, programmer-friendly database that conveys knowledge and network information, allowing for provisioning and traffic-engineered route control services.
- Scalable virtual control plane elements, which provide strong automation and flexibility for in-memory representation of intended control scenarios.

1.3.4 Use Cases

Pioneering work on Routing Control Platforms (RCP) [5,14] has shown many of the potential benefits in more flexible routing [44], security [38], and connectivity management [43]. Replacing iBGP control with the direct FIB manipulation and extended action set of OpenFlow allows for a wider range of rich IP routing application. In the following, we will briefly present some use case scenarios that we want to pursue with the RouteFlow platform.

Engineered path selection. Leveraging the knowledge of all available paths in a domain, RouteFlow is able to offer advanced load-balancing and cost/performance path selection [13] [29] [44] per application or customer using OpenFlow flexible matching to push traffic into traffic-engineered paths or VRFs. Policy-based routing becomes untangled from flow classification restrictions or technology requirements. Moreover, OpenFlow allows for direct preparation for optical switching in the core as well edge to edge. Recent research [41] has shown that network-wide visibility and control enable joint traffic engineering and failure recovery by balancing load efficiently, while potentially simplifying router design and reducing operation costs for ISPs.

Path protection and prefix independent convergence. The complete view of route state allows for novel IP route protection schemes like per-edge computation of primary and secondary paths for each prefix, considering shared risk group inferences [8]. Group- and multi-tables introduced in version 1.1 allow for an implementation of hierarchical FIB organization [16] to quickly recover from edge failures. Network-wide, loop-free alternate next hops can be pre-calculated and maintained in the group tables, removing the burden of one flow-mod per prefix updates, and allowing a fast local repair –provided an OAM extension is available, for instance, triggered by a Bidirectional Forwarding Detection (BFD) state change. The indirection points itnroduced by OpenFlow group tables allow to flexibly point and update next hops, a feature known as prefix-independent convergence capabilities. The common design principle is moving away from dynamic route computation by decoupling failure recovery from path computation [6].

Data plane security. One generic OpenFlow/SDN application is to distribute firewall and IPS/IDS functionality along the path, potentially on a customer basis, instead of either forcing all traffic to pass through or be detoured to a specialized box. Once detected, a selective DDoS blackholing strategy (akin [43]) can be deployed by inserting higher priority flow entries with drop action only on the inbound ports under attack. OpenFlow fine granular rules ($inport$,src,dst,$tp - port$) allow to blackhole traffic without taking out of service the target IPs (as with today's common filtering and BGP-based mechanisms). Similar filtering capabilities can be employed to duplicate traffic for legal intercept purposes or advanced monitoring applications in spirit of SPAN/TAP ports but pin out by OpenFlow flexible matching.

Secure inter-domain routing. Secure origin BGP (soBGP) or secure BGP (SBGP) deployment gets easier due to the readily available CPU power when consolidating BGP into commodity server technology. Benefits of the centralization approach include: 1) elimination of duplicate processing, signing once per neighbor rather than once per neighbor peering location; 2) avoidance of code upgrades of all existing AS border routers; 3) maintenance of keys in one place, rather than on every router; duplicate processing plus the non-trivial SW and HW upgrades of AS border routers. Deployment gets easier due to plenty CPU and crypto assistance on the controller, which centralizes key management and reduces the signing overheads to once per neighbor rather than per peering location. Moreover, route selection can prioritize those trustable ASes participating in address registries, PKI, DNSSEC, or new security techniques beyond today's concepts. For instance, Morpheus [38] proposes algorithms (e.g. history-based anomaly detection) to achieve 'pretty good' BGP security without requiring massive S*BGP adoption [18].

Simplifying customer multi-homing and IPv6 migration. Enterprise networks lack of customer-friendly, effective multi-homing solutions, especially in the absence of PI addresses or dual-stack IPv4/v6. RouteFlow-based policy routing could be used as an upstream affinity mechanism, only sending packets sourced from ISP1 networks via ISP1 (and 2 via 2) by flow matching based on source IP subnet. Effectively, it delivers a VRF per WAN link with source address-based VRF selection for egress traffic. The OpenFlow API allows to update flows based on performance and/or cost [29]. Simultaneously, edge migration to IPv6 could be accelerated and cheaply rolled out with OpenFlow match plus encapsulate actions. IPv6-v4 migration/translation (e.g., NAT64, DS-Lite) services could be flexibly inserted into the datapath, potentially enabling to let operators offer a hosted service for the IPv6 control plane.

Internet eXchange Point. Internet exchange point (IX or IXP) is the physical infrastructure through which ISPs networks (autonomous systems) exchange Internet traffic. High-capacity L2/L3 switches serve as the main interconnect of these networks, where BGP is used at the protocol level to implement the business-driven traffic policies. The growing number and size of IXPs in addition to the increase and evolution of peering relationships (flatter Internet effect) make IXPs an attractive ecosystem with a fundamental role in overall Internet traffic, especially in these days of distributed cloud and content delivery networks [1]. We believe that RouteFlow-like architectures leveraging route servers and traffic monitoring may contribute to fasten the pace of IXP operations and enable new service models beneficial to all parties involved. Initial activities along this application space have been recently reported in New Zealand.[7]

[7]http://list.waikato.ac.nz/pipermail/nznog/2012-December/019635.html

1.3.5 Concluding Remarks

While the trigger for this piece of work was looking at a transition scenario from traditional IP to SDN via an hybrid OF controller, the work on centralized IP routing has shown a number of advantages of the proposed IP split (hybrid) architecture, some applicable in general and inherent from OpenFlow/SDN, and some routing protocol-specific like BGP routing applications.

The horizon is however not without challenges. May be the most pressing ones from an architectural point of view is reaching a solution with high availability in all component layers, from the OpenFlow switches connecting to master and slave controllers, through the RouteFlow middleware, and up to the virtualized routing stacks. Other open challenges which may be more transient due to the maturity of the OpenFlow switch products are (i) the processing limitations of the OpenFlow agent in the switch CPU (limiting the number of OpenFlow events e.g., flow-mod/sec., data-to-control capacity and delay), and (ii) the current available hardware-based flow table sizes in the range of a few thousands.

1.4 Prototype Implementations and Experimental Evaluation

Throughout the life of the project, many experiments and demonstrations have be conducted to proof the concept of the RouteFlow architecture. In this section, we will present the latest collected data to evaluate the current RouteFlow prototype and exemplify some of the recent use cases like multi-controller deployments. This section tries to answer commonly asked questions around the performance and scalability of the architecture.

1.4.1 How much latency is introduced between data and control plane?

To answer this question we carried two types of experiments to measure the different paths between the control and data planes. The first one measures the latency of events (e.g. ARP) that result in `flow-mods` carried through the RouteFlow IPC / messaging layer. The second one measures the latency of control traffic (e.g. PING) carried via the two OpenFlow channels that glue together VMs and the physical devices .

Route update events

Performance measurements were made using the *cbench* tool [42], which simulates a number of OpenFlow switches generating requests and listening for flow installations. We adapted *cbench* to fake ARP requests (inside 60 bytes `packet-in` OpenFlow messages). These requests are handled by a modified version of the RFClient so that it ignores the routing engine. This way, we could effectively eliminate the influences of both the HW and SW which are not under our control, measuring more closely the specific delay introduced by RouteFlow.

The code and tools used to run these tests are openly available.[8] Except otherwise specified, the benchmarks were made on a Dell Latitude e6520 with an Intel Core i7 2620M processor and 3 GB of RAM.

In latency mode, *cbench* sends an ARP request and waits for the `flow-mod` message before sending the next request. The results for RouteFlow running in latency mode on POX and

[8]https://github.com/CPqD/RouteFlow/tree/benchmark

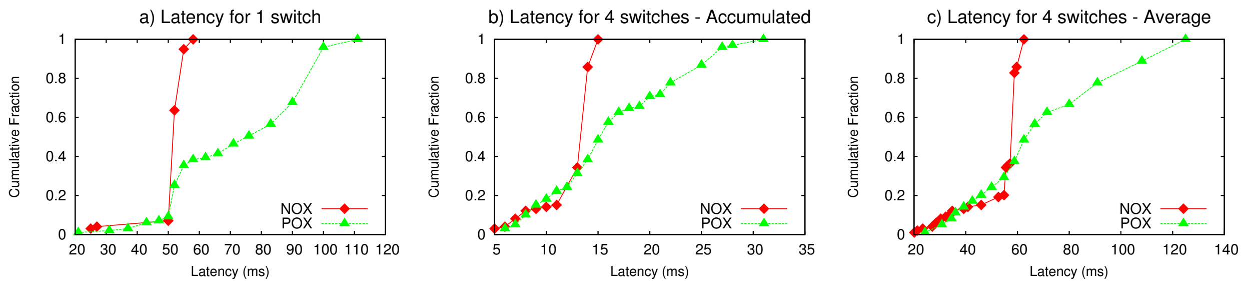


Figure 1.12: Latency CDF graphs for NOX and POX controlling a single network with 1 and 4 switches (taken from 100 rounds)

NOX are shown in Figure 1.12.

Each test is composed by several rounds of 1 second in duration, in which fake packets are sent to the controller and then handled by RFProxy that redirects them to the corresponding RFClient. For every test packet, the RFClient is configured to send a flow installation message. By doing this, we are testing a worst-case scenario in which every control packet results in a change in the network. These tests are intended to measure the performance and behavior of the new IPC mechanism.[9]

Figure 1.12 illustrates the cumulative distribution of latency values in three tests. Figure 1.12a shows the latency distribution for a network of only 1 switch. In this case, the IPC polling mechanism is not used to its full extent, since just one message will be queued every time. Therefore, the latency for the majority of the rounds is around the polling timeout. Figure 1.12b shows the accumulated latency, calculated considering all 4 switches as one. When compared to Figure 1.12c, which shows the average latency for all the 4 switches, the scales differ, but the behavior is similar. The accumulated latency shows that the IPC performs better in relation to the case in Figure 1.12a, mostly because the IPC will read all messages as they become available; when running with more than one switch, it is more likely that more than one message will be queued at any given time, keeping the IPC busy in a working cycle, not waiting for the next poll timeout.

Another comparison based on Figure 1.12 reveals that RouteFlow running on top of

[9]The IPC mechanism uses a 50 ms polling time to check for unread messages in MongoDB. This value was chosen because it optimizes the ratio of DB access to message rate when running in latency mode. Whenever a polling timeout occurs, the IPC will read all available messages.

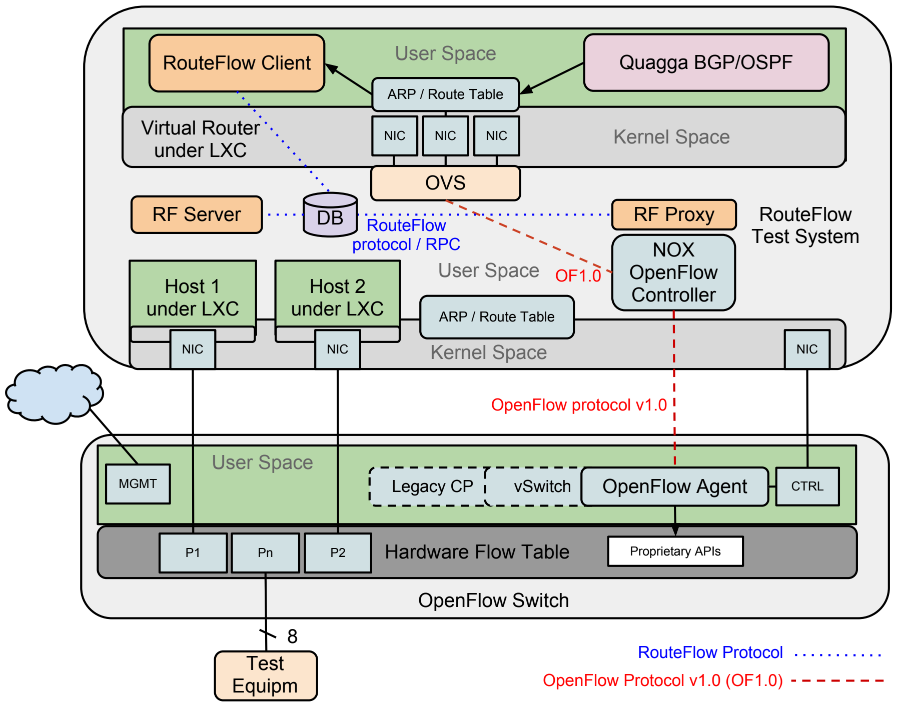


Figure 1.13: RouteFlow Test System built by Router Analysis [40]. The reference test controller design is based on a SuperMicro X9SAE-V with a single Intel I7-3770 CPU, 32GB ECC RAM, 240GB Intel 520 Series SSD and 4 Intel I340-T2 GbE NICs.

NOX (RFProxy implemented in C++) is more consistent in its performance, with most cycles lasting less than 60 ms. The results for POX (RFProxy implemented in Python) are less consistent, with more cycles lasting almost twice the worst case for NOX.

OpenFlow control channel performance

We turn now the attention to the latency of the data to control plane channel formed by two OpenFlow segments, one 'southbound' from the controller to the datapath switches, and the second 'northbound' from the controller to the OVS where the virtualized routing engines are attached. Recall that this control to data plane channel is used to exchange control plane traffic entering the switch and allowing neighbouring legacy routers to interact with the RouteFlow control plane.

These experiments were carried in the testing setup presented in Figure 1.13. The two OpenFlow control channels are depicted as red slashed lines. The tests consisted in PING

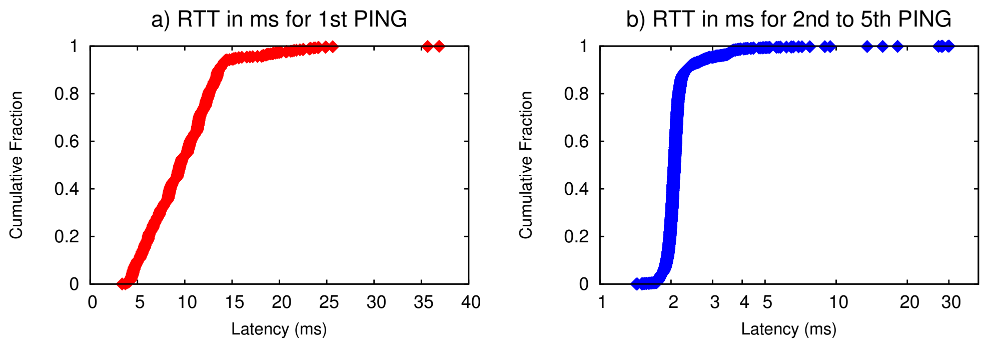


Figure 1.14: CDF of the RTT values for the first PING (left) and the second to fifth PING request/reply (right).

requests originating in the LXC virtualized control plane and with destination being the IP addresses of Host1, Host2, and each of the 8 interfaces of the test equipment plugged to the physical ports of the hardware-based OpenFlow switch under test (Pronto 3290). In a sequential fashion, 5 PINGs were sent to each IP, and the experiment was repeated 100 runs, obtaining altogether 5,000 RTT values (5 PINGs times 10 IPs times 100 repetitions).

Figure 1.14 presents the cumulative distributed function obtained for the RTT values. The RTT of each first PING request accounts also for the ARP resolution process that takes place prior to sending the ICMP packet. The large fraction of the first PING take less than 15 *ms*. The next PINGs travel back and forth through both OpenFlow control segments in less than 3 *ms*. PING (and ARP) packet exchanges match first the OVS in the control plane and are first carried in OpenFlow `packet-in` messages to the RFProxy instance and then sent out via OpenFlow `packet-out` messages to the datapath physical ports. The reverse path uses the same OpenFlow message sequence but in a different order. The physical switch originates `packet-in` messages carrying packets that match the control traffic match rules installed, and the RFProxy decapsulates the payload and sends it via `packet-out` to the corresponding interfaces of the control plane OVS.

Note that these values are obtained in positive load conditions, since neither the hosts nor the OpenFlow switch is busy with other tasks. CPU consumption of the OpenFlow agent due to additional (serial) messaging processing load are expected to increase the latencies to the control plane. This effect has been observed and reported by others [28, 40] and should be further investigated in future work on stress testing under realistic workload conditions.

1.4.2 How many control plane events can be handled?

In throughput mode, *cbench* keeps sending as much ARP requests as possible in order to measure how much flow installations are made by the application. The throughput test stresses RouteFlow and the controller, showing how many flows can be installed in a single round lasting for 1 second. The results in Table 2 show how many flows can be installed in all of the switches in the network during a test with 100 rounds lasting 1 second each. The results show that the number of switches influence the number of flows installed per second, more than the choice of the controller.

Table 1.1: Total number of flows installed per second when testing in throughput mode (Average, standard deviation and 90% percentile taken from 100 rounds).

Controller	**1 switch**		**4 switches**	
	Flows (Avg.)	# Flows (90%)	# Flows (Avg.)	# Flows (90%)
POX	915.05 ± 62.11	1013.0	573.87 ± 64.96	672.0
NOX	967.68 ± 54.85	1040.0	542.26 ± 44.96	597.0

1.4.3 What is the actual performance in a real network?

Test on a real network infrastructure were performed using the control framework of the FIBRE project,[10] with resources in two islands separated by 100 km (the distance between the CPqD lab in Campinas and the LARC lab at USP in So Paulo). To evaluate the delay introduced by the virtualized RouteFlow control plane, we measured the round-trip time from end-hosts when sending ICMP (*ping*) messages to the interfaces of the virtual routers (a LXC container in the RouteFlow host). This way, we are effectively measuring the compound delay introduced by the controller, the RouteFlow virtual switch, and the underlying network, but not the IPC mechanism. The results are illustrated in Table 1.2 for the case where the RouteFlow instance runs in the CPqD lab with one end-host connected in a LAN, and the second end-host located at USP. The CPqD-USP connectivity goes through the GIGA network and involves about ten L2 devices. The end-to-end delay observed between the hosts connected through this network for *ping*) exchanges exhibited line-rate performance, with a constant RTT around 2 ms. The results in Table 1.2 also highlight the performance

[10]http://www.fibre-ict.eu/

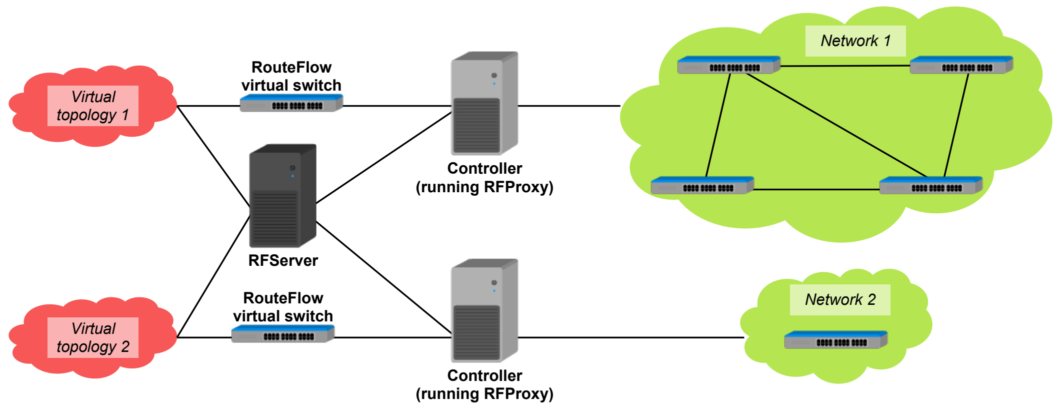


Figure 1.15: Test environment showing several controllers and networks

gap between the controllers. The NOX version of RFProxy introduces little delay in the RTT, and is more suited for real applications. These results are in line with the previous OpenFlow control channel performance tests in Section 1.4.1.

Table 1.2: RTT (milliseconds) from a host to the virtual routers in RouteFlow (average and standard deviation taken from 1000 rounds)

Controller	**host@CPqD**	**host@USP**
POX	22.31 ± 16.08	24.53 ± 16.18
NOX	1.37 ± 0.37	3.52 ± 0.59

1.4.4 How to split the control over multiple OpenFlow domains?

In order to validate the new configuration scheme, a simple proof-of-concept test was carried to show the feasibility of more than one network being controlled by RouteFlow. This network setup is illustrated in Figure 1.15, and makes use of the flexibility of the new configuration system. A central RFServer controls two networks: one contains four OpenFlow switches acting as routers being controlled by a POX instance, and the other contains a single OpenFlow switch acting as a learning switch being controlled by a NOX instance. In this test, RouteFlow was able to properly isolate the routing domains belonging to each network, while still having a centralized view of the networks.

1.4.5 How does the virtualized control plane scale?

To evaluate the virtualization elements considered for the RouteFlow control plane architecture, experiments were done under different connectivity scenarios and scaling factors. The control planes tested were composed of virtual routers based on the Quagga routing stack version 0.99.10 [37], virtualized with the LXC containers integrated into mainline kernel 2.6.32 [26], and attached to a single OpenvSwitch [36] instance (version 1.1.0).

Two types of topologies were explored corresponding to different scalability requirements:
- Grid-based topologies: Two network topologies were defined with nodes arranged in a 3x3 grid and a 4x4 grid. This arrangement was adopted to evaluate the scaling behaviour of the resource consumption by increasing number of nodes and virtual network connections.
- Full-mesh topologies: Three fully-meshed network topologies of 15, 25 and 35 nodes were evaluated to under the operation of the OSPF protocol. All subnets were attached to the OSPF area 0 and none of the interfaces had summarization activated. While this configuration would not be recommended for real-world deployments due to the LSA N-squared problem [2], this setup is valid as a stress-test for the scalability of the virtualized control plane.

The low-scale set of experiments (i.e., grid-based topologies) was run on a Intel Core 2 Duo 2.93Ghz platform with 3GB of RAM, and the second set of experiments (i.e., full-mesh topologies) was executed on a system with an Intel Xeon X5660 CPU and 48GB of RAM, both running Debian GNU/Linux 5.0 distribution.

Three different stages in the operation of the virtual control plane were introduced: (i) initialization, (ii) regular operation (ten minutes after the initialization of each virtual topology), and (iii) topology changes (i.e., link down and link up events). The following metrics and data were collected:

- **Boot-time of the nodes.** Time interval needed for nodes to be fully operational after the request for its instantiation.
- **CPU and memory usage.** Allow to infer the degree of overhead in moments of

intense demand.

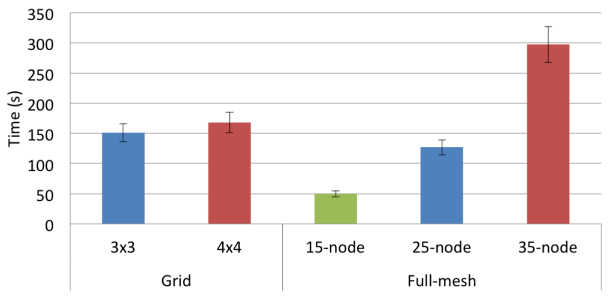
- **Convergence time.** Measures the convergence time of the OSPF protocol at different stages. During initialization, the measured time is the interval between topology boot up and the "first" convergence of the OSPF protocol, i.e., the time when the last OSPF announce message (LSAs) is exchanged. During topology changes, the convergence time interval is measured between the first and the last link state update (LSU) messages, after link disconnections and reestablishment.

In order to confer statistical relevance to the obtained results, each metric had its data collected through 30 distinct executions of each network topology. Table 1.3 presents a consolidated view of the results. After a 77% growth in network (from 9 to 16 nodes), initialization time grew 96%. The result is almost linear, considering that the increase in the number of nodes also lead to a higher count of virtual links (from 12 to 24) and network interfaces. During boot time of the virtual routers, CPU usage was in the range 99-100%. Immediately after, it dropped below 3% and never gone up this level again for the duration of each experiment execution. As none of the resource management features of LXC were used, this result is consistent with container-based virtualization premises. Evolution of memory consumption is also approximately linear with the increasing scale of topology. RAM usage peeks on connectivity modification events, and the 3x3 grid uses 23% more memory during OSPF reconvergence, while the larger 4x4 network increases by 29%. Again, considering the increased count of links and interfaces, this is not unexpected.

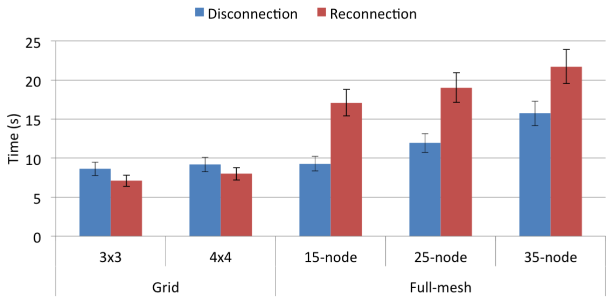
Figure 1.16 covers how long OSPF convergence takes. Figure 1.16a shows that the greater number of links and increased network diameter have low impact on the first convergence when opposing 3x3 and 4x4 topologies. Fully-meshed 15- and 25-node topologies were also evaluated and performed better, on average, than grids. This can be explained by the topology itself: as all nodes are connected, few OSPF discoveries offer "good enough" alternatives

Table 1.3: Control Plane Operation Metrics

Topology	**Init. (s)**	**RAM Utilization (MB)**		
		Init.	Reg. Oper.	Conn. Events
3x3	32	175	185	228
4x4	63	269,75	286	369



(a) Time for initial convergence.



(b) Time for convergence after topology modification.

Figure 1.16: OSPF convergence in grid and full-meshed network topologies.

to be disseminated between multiple nodes (only those happened before all node's connections are up). For 35-node networks, however, the exponential growth in link numbers shows a much worse result, compatible with what the LSA N-squared problem suggests.

The timings shown in Figure 1.16b indicate that connectivity events take longer to converge on all full-meshed topologies, when related to 3x3 and 4x4 grids. This could be explained by the fact that connectivity changes always propagate through the entire network on full-meshed topologies (as all nodes report a "dead" neighbour), while this does not always hold true for grids. The same logic can be applied to explain why host reconnections have slower convergence in full-meshes, while in grids disconnections take longer to propagate: a reconnected node always bring up new "best routes" to all nodes in a full mesh, while this does not hold on grids.

The stable growth in grid's convergence time, allied to stable and predictable results in relation to full-meshes, suggest that LXC and Open vSwitch allow for a scalable control plane. Moreover, the Open vSwitch approach to attach and connect the LXC routers allows to implement a a distributed control plane environment to scale CPU and memory requirements over a cluster of servers. Such a divide and conquer approach has been shown previously in this section when splitting control over multiple OpenFlow domains.

1.5 Conclusions and Future Work

In this chapter we have tried to introduce the motivation and problem statement of hybrid networking models in OpenFlow/SDN. More specfically, we have presented two controller-centric approaches to combine traditional control plane functions with OpenFlow control. Hybrid networking will certainly be a concern for SDN adotion as it will touch both economical and technical issues. The RouteFlow and LegacyFlow approaches present point solutions that allow for different migration strategies and deployment scenarios.

Further than enabling a gradual adoption path for OpenFlow technology, we have discussed how centralized IP routing engines coupled with OpenFlow-based installation of IP-oriented flow rules leads to a new degree of control that enables a number of applications that are a real challenge (if not impossible) to realize in classic multi-vendor networks.

As for the future, we are devoting efforts to increase the stability and scalability of the software components. This is especially critical as we start deploying pilots with operational traffic. Extensions to support MPLS and IPv6 are also underway. To support new protocols and realize use cases that involve QoS, load balancing, and fast failover, we will upgrade the prototypes with the newest OpenFlow protocol version.

We hope that the presented pieces of work are only examples of disruption on the horizon enabled by OpenFlow (in particular) and SDN (in general). Fueled by indsutry modularization and the forces of community-driven open-source developments, the software-defined networking era promises unprecedented levels of innovation in networking and computing technologies.

1.6 Acknowledgments

This work is partially supported by the GIGA Project (FINEP/FUNTTEL) and was done while the first author was employed by CPqD. Carlos N. A. Corrêa was supported by Unimed Federação RJ. We also would like to acknowledge colleagues and co-authors that contributed to previous publications related to the RouteFlow (CPqD[11]) and LegacyFlow (GERCOM-UFPA[12]) projects. We are thankful to Steven Noble from Router Analysis, who contributed with the experiments described in Section 1.4.1, Figure 1.13, and further detailed on an extended report and analysis on OpenFlow testing [40].

References

[1] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a large european ixp. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 163–174, New York, NY, USA, 2012. ACM.

[2] Alfred V. Aho and David Lee. Abstract hierarchical networks and the lsa n-squared problem in ospf routing.

[3] Autobahn. Autobahn project. Avaliable: http://www.geant2.net/server/show/nav.756, dec 2012.

[4] Big Switch Networks. Open Software-Defined Networking (SDN) Product Suite. http://www.bigswitch.com/.

[5] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *NSDI'05*.

[6] Matthew Caesar, Martin Casado, Teemu Koponen, Jennifer Rexford, and Scott Shenker. Dynamic route recomputation considered harmful. *SIGCOMM CCR*, 40:66–71, April 2010.

[7] Chao-Chih Chen, Lihua Yuan, Albert Greenberg, Chen-Nee Chuah, and Prasant Mohapatra. Routing-as-a-service (raas): A framework for tenant-directed route control in data center. In *Proceedings of the IEEE INFOCOM 2011*, INFOCOM '11, pages 1386–1394, Shangai, 2011. IEEE.

[8] D. Saucez et al. Low-level design specification of the machine learning engine. FP7 ECODE Deliverable D2.3, Dec 2011.

[9] Saurav Das, Yiannis Yiakoumis, Guru Parulkar, Nick McKeown, Preeti Singh, Dan Getachew, and Premal D. Desai. Application-aware aggregation and traffic engineering in a converged packet-circuit network. In *National Fiber Optic Engineers Conference*, page NThD3. Optical Society of America, 2011.

[10] M. Davy. and et. al. A case for expanding openflow/sdn deployments on university campuses. Technical report, GENI Workshop Whiterpaper, 2011.

[11] http://www.cpqd.com.br
[12] http://www.gercom.ufpa.br

[11] Dragon. Dragon project. Avaliable: http://dragon.maxgigapop.net/, dec 2012.

[12] Toshal Dudhwala. Discovery in openflow networks: A report from the onf plugfest. http://blogs.ixiacom.com/ixia-blog/discovery-in-openflow-networks-at-onf-plugfest/, Nov 2012.

[13] Nick Duffield, Kartik Gopalan, Michael R. Hines, Aman Shaikh, and Jacobus E. Van Der Merwe. Measurement informed route selection. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, pages 250–254, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] Nick Feamster, Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. The case for separating routing from routers. In *FDNA '04*, 2004.

[15] FIBRE. Future internet tesbeds experimentation between brazil and europe. Avaliable: http://www.fibre-ict.eu/, dec 2012.

[16] Clarence Filsfils and et al. BGP Prefix Independent Convergence (PIC) Technical Report. Technical report, Cisco, 2011.

[17] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Intelligent design enables architectural evolution. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets '11, pages 3:1–3:6, New York, NY, USA, 2011. ACM.

[18] Phillipa Gill, Michael Schapira, and Sharon Goldberg. Let the market drive deployment: a strategy for transitioning to BGP security. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 14–25, New York, NY, USA, 2011. ACM.

[19] Google Inc. Inter-Datacenter WAN with centralized TE using SDN and OpenFlow. https://www.opennetworking.org/images/stories/downloads/misc/googlesdn.pdf.

[20] Kate Greene. TR10: Software-Defined Networking. *MIT Technology Review*, 2009.

[21] P. Gross. Choosing a Common IGP for the IP Internet. RFC 1371 (Informational), October 1992.

[22] Eric Keller and Jennifer Rexford. The 'Platform as a Service' model for networking. In *INM/WREN 10*, April 2010.

[23] 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 *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[24] K. K. Lakshminarayanan, I. Stoica, S. Shenker, and J. Rexford. Routing As A Service. Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.

[25] I2RS Discussion List. Interface to the internet routing system. Avaliable: https://www.ietf.org/mailman/listinfo/i2rs, aug 2012.

[26] LXC. lxc linux containers, 2011. `http://lxc.sourceforge.net/`.

[27] E. Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945 (Proposed Standard), October 2004.

[28] Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, Andrew R. Curtis, and Sujata Banerjee. Devoflow: cost-effective flow management for high performance enterprise networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 1:1–1:6, New York, NY, USA, 2010. ACM.

[29] Murtaza Motiwala, Amogh Dhamdhere, Nick Feamster, and Anukool Lakhina. Towards a cost model for network traffic. *SIGCOMM Comput. Commun. Rev.*, 42(1):54–60.

[30] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corrêa, Sidney C. de Lucena, and Maurício F. Magalhães. Virtual Routers as a Service: the RouteFlow approach leveraging Software-Defined Networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[31] Marcelo Ribeiro Nascimento, C. Esteve Rothenberg, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. QuagFlow: partnering Quagga with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:441–442, August 2010.

[32] NEC. ProgrammableFlow Networking. http://www.necam.com/pflow/.

[33] Nicira. Network Virtualization Platform. http://nicira.com/en/network-virtualization-platform.

[34] ONF. Open Networking Foundation. https://www.opennetworking.org.

[35] OSCARS. Oscars project. Avaliable: http://wiki.internet2.edu/confluence/display/DCNSS/OSCARS, dec 2012.

[36] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[37] GNU Project Quagga. Quagga software routing suite, 2010. `http://www.quagga.net/`.

[38] Jennifer Rexford and Joan Feigenbaum. Incrementally-deployable security for interdomain routing. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 130–134, Washington, DC, USA, 2009. IEEE Computer Society.

[39] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, New York, NY, USA, 2012. ACM.

[40] Router Analysis, Inc. The State of OpenFlow 2012 - Report and Analysis. RA-011513-01. http://www.routeranalysis.com/the-state-of-openflow-2012-report-and-analysis/, Jan 2013.

[41] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer 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.

[42] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[43] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, and S. Zelingher. Dynamic connectivity management with an intelligent route service control point. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, INM '06, pages 29–34, New York, NY, USA, 2006. ACM.

[44] Yi Wang, Ioannis Avramopoulos, and Jennifer Rexford. Design for configurability: rethinking interdomain routing policies from the ground up. *IEEE J.Sel. A. Commun.*, 27:336–348, April 2009.

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