When Open Source Meets Network Control Planes

Christian Esteve Rothenberg(*) (UNICAMP – University of Campinas, Brazil)
Roy Chua (SDNCentral & Wiretap Ventures, US)
Josh Bailey (Google, New Zealand)
Martin Winter (NetDEF – Network Device Education Foundation, US)
Carlos Correa (UFF – Fluminense Federal University & UNIMED, Brazil)
Sidney C. de Lucena (UNIRIO – Federal University of the State of Rio de Janeiro, Brazil)
Marcos Rogério Salvador (Lenovo Innovation Center, Brazil)
Thomas Nadeau (Brocade, US)

(*) Corresponding author: chesteve@dca.fee.unicamp.br

Abstract— **The advent of Software Defined Networking (SDN) is opening up interfaces to historically proprietary networking devices, promising improved orchestration and agility, lower cost of operations and, most importantly, a new wave of innovation. Networking and open source have been strange bedfellows in the past, but at the present time, a rich ecosystem is developing around SDN. In this article, we look at the state of open source as it pertains to the SDN stack and discuss its potential role in changing the networking software and hardware landscape. We discuss the availability and maturity of southbound and northbound APIs along the overall impact of SDN to standardization. We then delve into the RouteFlow architecture, an ongoing project to glue together open source IP routing stacks with OpenFlow networks. Along the way, we share some operational experiences of its use at a live Internet exchange (Cardigan Project), concluding that broader production deployments of open source enabled SDN stacks may be not that far away.**

Keywords— **Software Defined Networking, OpenFlow, Routing, BGP, Network Architecture, Internet eXchange Point, Open Source**

I. INTRODUCTION

One of the key aspects of Software Defined Networking (SDN) is centered on introducing new abstractions and rethinking the separation and distribution of control plane functions from data forwarding functions, which are traditionally closely together on the same device. Successful implementations of real products and experiments that embrace these concepts have typically done so using protocols and APIs that are, or are becoming, standardized [1,2]. The term SDN [3] has been progressively broadening since its (OpenFlow-triggered) inception in 2009. Today, SDN encompasses multiple networking proposals, spanning from the original control split via the OpenFlow protocol [2], to software-based edge tunnels to define virtual network overlays, and hybrid proposals to augment and expose existing distributed protocols via programmatic interfaces (e.g. IETF I2RS) [4].

Despite the SDN flavor considered, but notably in the case of OpenFlow, one promising opportunity is the ability to assemble a set of open source software components that can be used to define the network control logic and interact with multi-vendor networking hardware via open APIs (see Fig. 1). The expected result is the powerful combination of flexibility, adaptability and lower costs of ownership by allowing solutions made from high-performance commercial networking hardware and (open source) software running in commodity server technology. This potentially disruptive combination can be seen as a means of replicating in networking the success story of LAMP – an approach that allowed the IT industry to build scalable server farms out of low-cost commodity gear at minimal licensing costs rather than continuing to use proprietary vendor solutions. In fact, this scenario is playing itself out again within the SDN industry with the advent of the Open Daylight Project (ODP) and related large scale open source infrastructure projects such as OpenStack and OpenCompute.

In this article, we explore this new wave of SDN-based solutions and describe how the evolving open source networking ecosystem is unlocking various opportunities otherwise unavailable except by vendor-proprietary solutions. As an example of this, we relate to our experience in building and deploying RouteFlow [5], a software-defined IP routing architecture that puts together multiple open source components to build a viable and functional control software system for OpenFlow-enabled networks. More specifically, we share some deployment experiences within the Cardigan project [6] that has been now in operation for over one year, peering at a live Internet eXchange Point (IXP) where it appears as a single legacy router while actually being two OpenFlow switches located in different buildings controlled by a single remote controller.

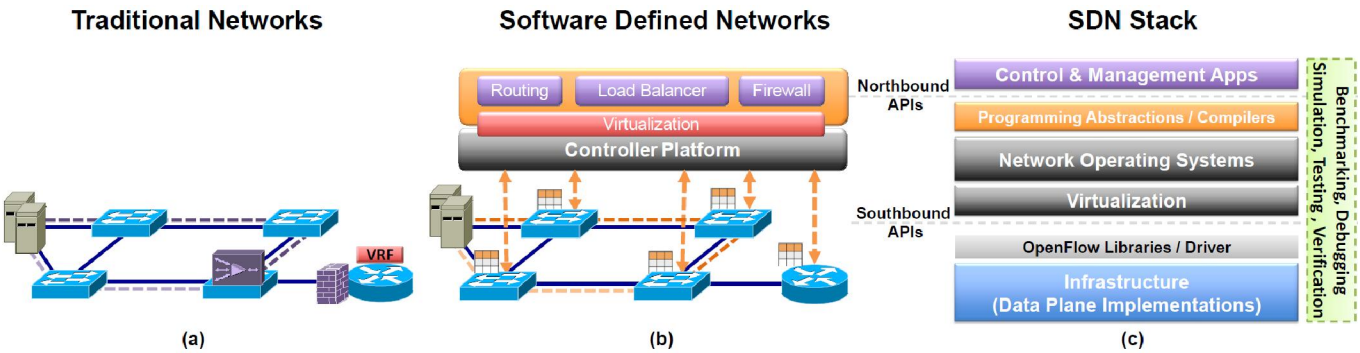

Figure 1. (a) Traditional networks with distributed control and device-specific management contrast with (b) the flexible and logically-centralized control and management in SDN enabled networks due to southbound protocols (e.g. OpenFlow, RESTCONF, PCE-P, etc…) that interact with data plane devices, enabled by (c) an emerging SDN stack based on open APIs between functional layers in addition to a number of supporting tools.

Enrico Fermi reputedly asked if intelligent life existed elsewhere in the universe, why has it not arrived here yet? Many in the networking industry would ask: if SDN is really so useful and such an advance, why have we not seen it deployed more widely in production networks? Some of reasons may be the need (i) to directly address operational comfort with SDN, (ii) to provide concrete benefits, and (iii) to demonstrate a practical migration path – not to mention the ability to interoperate with non-SDN networks. Some concepts, such as the redistribution of control plane functions and offline route computation, can be argued to have already existed and just not been fully exploited. However, perhaps the most straightforward counter-argument is that SDN provides simpler, more reliable and easier to operate networks (cf. Google B4, Microsoft SWAN). Our efforts within RouteFlow and Cardigan, similarly to running code oriented initiatives like ODP, go some way to address these questions.

Google´s recently unveiled inter-datacenter WAN called B4 [7] is based on SDN principles, OpenFlow technology, and open source components such as the Quagga routing stack – resembling the RouteFlow architecture. B4 has been running in operation for almost 3 years and claims simpler, more reliable and easier means to operate the network when compared to traditional distributed control planes or MPLS traffic engineering tools. Furthermore, B4 attains unprecedented levels of link utilization (close to 100%) by augmenting the SDN routing system with custom traffic engineering algorithms, altogether yielding a very appealing, low-cost proposition that was simply not achievable by traditional networking approaches.

While Google's B4 is a remarkable example for SDN in production, at the same time, it is also an exception, since the resources and talent to accomplish such an in-house networking project is only at the hands of a few big players. This may be however a timing issue of technology adoption and ecosystem maturity. In this article, we argue that fully open source SDN deployments may be not that far away. We support this idea by means of the RouteFlow developments and the Cardigan pilot experience. Along this journey, we will also discuss open challenges and the work ahead towards real-world open source defined networks in operation.

II. Open Source in Networking

FLOSS (free/libre/open source software) plays a relevant role in the networking industry by fueling the eco-system with low-cost and customizable software artifacts to provide network management and configuration tools, security suites, protocol testing and interoperability, software appliances, and so on. Similar to any software-centric industry, the reasons behind the success are the customization opportunities and the economies of open source development and commercialization [8].

Until recently, success of FLOSS in networking has been limited to the management plane of networks and software appliances running in commodity x86 servers. Noteworthy examples of the latter include software-based IP routing (e.g., Quagga, Bird) and security solutions (e.g. Bro, Snort) available as free, open source code –often in addition to enhanced counterpart versions commercialized under a revenue model that includes value-added services, such as support.

A. Open Source meets SDN

The generalized SDN proposition (and OpenFlow in particular) take the traditional networking scene (Figure 1a) a step forward by allowing the brain (i.e., control plane, and related applications) of a network to be implemented in high-level programming languages on commodity server technology untangled from the physical boxes implementing the forwarding plane functions (See Figure 1b). The theoretical SDN abstractions for the forwarding and control planes have led to software-based embodiments giving birth to an impressive amount of contributions towards an open source SDN stack (See Figure 1c). Table 1 provides an overview of FLOSS projects [9] covering different realms of the evolving SDN stack ecosystem.[1]

Beyond open, standard libraries to interact with networking devices, there is active competition on the controller and northbound APIs (the application interfaces) within open source projects, a necessary step to mature the ecosystem. A number of control and

[1] Note that identified open source projects vary in their degree of code quality, active development, and community support. While many are mainly academic exercises, others benefit from ample support by financial and professional developer resources.

management applications have been developed to address different goals (e.g., topology discovery, routing, etc). However, existing applications, similarly to support tools, are controller-specific, i.e., not portable among controllers due to the lack of common APIs. Much like in the early days of Web servers or Java application servers, competition serves to foster ideas and drive innovation. A continued spawning of new SDN projects (controllers, applications, testing, and IDEs) can be expected before consolidation occurs around a winning platform (like Apache / Tomcat).

TABLE I. OVERVIEW OF OPEN SOURCE PROJECTS ACTIVE IN SDN

SDN Realm	Open Source Project Name
Benchmarking	Cbench (GPLv2, C/Perl/UNIX shell), OFLOPS (GPL, C)
Debugging / Testing / Simulation	ndb, OFRewind, STS (Apache, Python), OFDissector (BSD, C), liboftrace (BSD, C), OFTest (BSD, Python), Mininet (BSD, Python), fs-sdn (GPL, Python), ns-3 (GPL, C++), TestON (GPL, Python)
Verification	Hassel (GPL, Python), NetPlumber (GPL, Python), NICE (BSD, Python), FlowChecker, OFTEN
Control & Management Apps	Topology discovery (GPL, many), HostTracker (GPL, many), Plug-n-Serve (BSD, C++), Aster*x (BSD, C++), FlowScale (Apache, Java), SNAC (Python/C++), Odin (Apache, Python), PANE (BSD, Haskell), FRESCO (GPL, Python/C++), OSCARS (BSD, Java), RouteFlow (Apache, Python/C/C++/Java), Open DayLight (Eclipse, Java)
Programming Abstractions / Compilers / Isolation	FatTire, Flog, FML, Frenetic (GPL, OCaml), HFT, NetCore, Nettle, Procera, Pyretic (BSD, Python), Maple (Pyhon), FlowN, LibNetVirt (GPL, C/Python), OpenStack Neutron (Apache, Python)
Controller Platforms	NOX (GPL, C++); POX (GPL, Python); Maestro (LGPL, Java); Beacon (GPL, Java); Floodlight (Apache, Java); Ryu (Apache, Python); Trema (GPL, C/Ruby); FlowER (MIT, Erlang); NodeFlow (GPL, javascript); Mul (GPL, C); OpenDaylight (Eclipse, Java); ONOS (Java); OpenContrail (Apache, C++ / Python)
Data Plane Virtualization	FlowVisor (BSD, Java), PortVirt (BSD, C), Expedient (Apache, Python), OpenVirteX (Apache, Java)
OpenFlow Protocol Libraries / Southbound Driver	OFLib-Node (BSD, JavaScript), OpenFlowJ (BSD, Java), OpenFaucet (Apache, Python), Pylib-OpenFlow (BSD, Python/C++), freeflow (Apache, C/C++), xDPd/ROFL (MPL, C/C++), libfluid (Apache, C/C++)
Data Plane Implementations	NetFPGA (BSD, C/Verilog), Open vSwitch (Apache, C), Reference design (BSD, C), ofsoftswitch13 (BSD, C), OpenWRT/Pantou (GPL, C), Switch Light (Eclipse, C), Indigo Virtual Switch (Eclipse, C), LINC-Switch (Apache, Erlang)

At the controller level, noteworthy activities are underway within the ODP project (http://www.opendaylight.org), a vendor-led initiative housed within the Linux foundation to create an open source SDN controller and application ecosystem. ODP was formed solely as a similar industry reaction to the numerous vendor-proprietary (or semi-open source) SDN controller options. One of the goals of ODP is to produce a commercially viable, open-source based SDN controller. The ODP incorporates other open source projects into their code base (e.g., Open vSwitch), and integrates with other southbound drivers or northbound interfaces (e.g. Openstack Neutron), creating further momentum.

In the management of open source projects, vendor-driven and community-driven are usually the starting point for many successful open source projects. As they become larger and more complex, a foundation is formed to provide more structure in the evolution of the project (Apache, Eclipse etc.). Historically, most successful open source projects tended to be community-driven and, occasionally, a vendor-driven project could mature and attract other community. Community involvement in terms of diversity of code committers and distributions of the base code is key to the success of FLOSS projects. The ODP, for example, includes coders from over one hundred different affiliations. While the project still needs to prove its effectiveness and success, it is showing clear positive signs due to its increasingly wide adoption within the industry.

At number of large vendors (e.g., Brocade, Redhat, Ciena, Ericsson and Cisco) have announced that they will be offering commercial products based on this edition, or even just enterprise versions of the controller. Like in the LAMP example, industry-driven open-source controller projects like ODP represent an ecosystem of components that can be snapped together to construct a functional environment, as the controller includes functionalities beyond OpenFlow (e.g., OVSDB, PCE-P, BGP-LS, NETCONF), multi-tenant management and full integration with Openstack. Using open source allows commercial vendors to utilize a shared platform that benefits a new framework (SDN / programmable networks) reducing fragmentation and risks for end-users. In addition, the open source way allows vendors to benefit from each other's development efforts in areas that are not strategic differentiation factors for their products.

In this way, the role of open source is not necessarily an end, but more of a development ethos or mentality to rapidly construct and evolve tools, techniques and actual running code based on community input, knowledge and interest. All these forces together are expected to introduce a new process in the marketplace where de facto standards are being forged through open-source platforms in parallel to / complementing traditional standardization process as presented in Table II. While standards development has been historically driven mainly by vendors, open source SDN is a movement driven by users (network operators, equipment buyers) to

innovate in their network infrastructure, reduce costs, and reduce vendor lock-in. This also means that the emergent APIs out of running open source code community developments should contribute to fast time-to-market implementations of proof of concepts (and end products), accelerating the path to working standards (*de-facto* and *de-jure*). Next, we focus the discussion one of the key areas of open source SDN, namely, the standardization and maturation of both Southbound and Northbound APIs.

TABLE II. SDN AND OPEN SOURCE ARE EVOLVING AND ACCELERATING THE PATH OF STANDARDIZATION

	Present with SDN	**Past / Traditional**
Drivers	Customer	Vendors
Goals	Address user / operator needs (customization)	Enable multiple solutions (interoperability)
Deliverables	Implementations & PoCs	Documents
Quantity of Standards	Less	More
Timetable	Few years	Many years
Validation	PoCs integral to the process	Products and deployments after release
Point of Control	Contribution to FLOSS codebase. Ability to understand codebase	Seat at standards committee table
Parties Involved	Anyone with domain expertise and coding ability	Vendors who can afford membership fees. Experts and academics with high standing in their fields

B. Southbound and Northbound Interfaces: State of affairs

When SDN first commercialized (by start-ups like Nicira first and Big Switch Networks later), the OpenFlow protocol was pushed to the forefront. The last 12 months have seen other Southbound mechanisms appear. Today, some of these mechanisms are full-fledged protocols like OpenFlow, defining both the transport and the messaging. Others are in the process of being formed. Southbound mechanisms that seem to have some level of market acceptance today range from NETCONF, SNMP and OVSDB, to OpenFlow and I2RS (under active development at IETF), to XMPP and just JSON over HTTPS (RESTful), to the recently unveiled policy-centric, declarative OpFlex by Cisco proponents (http://tools.ietf.org/html/draft-smith-opflex-00).

Unlike many other open source ecosystems, SDN grew up quite quickly in academic environments. As a result, end-users did not have a chance to test out SDN stacks in production environments until recently. While OpenFlow has been around for some time now, there has been limited uptake beyond version 1.0. One cause is the product life cycle of networking hardware and the challenges of mapping OpenFlow version 1.3 (beyond the minimal mandatory features) to existing ASIC-based pipelines. Another part of the reason is the lack of a standard (de-facto or otherwise) protocol library that both the open source development community and the vendor community can use – much like the OpenSSL library contributed to the production use of SSL protocols. After recognizing the need for a standard library that abstracts the intricacies of the OpenFlow protocol, the Open Networking Foundation (ONF) leading OpenFlow/SDN standardization recently announced libfluid (http://opennetworkingfoundation.github.io/libfluid/) as the winner of competition to create an open source multi-version OpenFlow driver to ease the development of both OpenFlow switch agents and controllers.

On the Northbound side, today's Northbound APIs are mostly proprietary, dependent on which SDN controller is being used and the use cases the controller is designed for. Historically, every open source controller had defined its own basic northbound interfaces, most of which were simple and reflected a thin layer over OpenFlow, occasionally supporting some elements of topology query. Today, about 20 SDN controllers exist commercially and each has its own set of northbound APIs. For instance, the Plexxi Controller features two APIs: 1) Workload Affinity API to allow external systems to communicate with the controller, and 2) Network Orchestration API to translate workload requirements into network instructions. In general, most commercial controllers only provide rudimentary northbound APIs, mostly for configuration and flow programming on a device level (rather than service-level and network-wide). As the requirements for different network applications are quite different, there is an expectation for several blends of northbound APIs. For instance, an API for security applications is likely to be different from one for routing policies enforcement.

In parallel to the ongoing standardization work at ONF Northbound Interface WG, code coming out from projects like ODP and OpenStack are appearing as candidate *de facto* northbound standards in spirit of the "running code before standardize" motto, a theme also present in the ONF Extensibility WG. Furthermore, some in the industry have argued that the right Northbound APIs should integrate into the orchestration stacks since those will dominate the majority of deployments. For instance, as long as the controllers have a Neutron plugin that adheres to the OpenStack architecture, the appropriate level of Northbound abstraction that end consumers should be interacting with is exposed at the OpenStack level, instead of at the SDN controller level.

With the significant differences in both Northbound and Southbound approaches and different layers of abstraction, it appears that a single standard layered SDN architecture will be unlikely. Nevertheless, efforts like the ODP are attempting to create a unified software architecture that accommodates the different approaches. With its MD-SAL (model-driven service abstraction layer) architecture, ODP hopes to support multiple abstraction models without mandating fixed layers of abstraction. MD-SAL also aims to provide different Northbound APIs via RESTCONF interfaces using YANG as the data modeling language

for model configuration and state data manipulation. The reality is that this will probably create end-to-end siloed stacks initially (matching Northbound and Southbound pairs) until users start seeing commonality and Northbound APIs start coalescing.

III. RouteFlow: Software-Defined IP Routing

IP routing is a core application of networking. As such, examining its progress within the SDN community provides an excellent window into the advancement of SDN concepts. The (simple) idea behind the RouteFlow design is to enable the creation of a virtualized network control plane, composed by virtual routers and to provide ways to install the resulting packet handling rules into datapaths' flow tables that fully represent the forwarding tables of each virtual router. The virtual routers exchange traditional network protocol messages (e.g., OSPF, BGP), like on a legacy network, and are able to interconnect with "external" physical (and legacy) routers, also providing a solution for hybrid networks (formed by SDN and legacy components) and a path for migration.

A. Architecture implementation built upon open source

RouteFlow leverages a number of open source solutions and introduces three components that altogether allow to run Linux-based routing stacks and have the generated forwarding information base (FIB) be arbitrary installed into OpenFlow devices. The resulting SDN stack and the utilized FLOSS components are shown in Figure 3. The three basic software components introduced by RouteFlow to allow for the glue of open routing stacks and programmable hardware devices are:

- RF-Client: User-space daemon that collects routing and forwarding information (ARP and IP route table) generated by the Linux-based routing engine of choice (e.g., Quagga, BIRD, XORP) typically running in a virtualized environment (e.g., LXC, KVM).
- RF-Server: Standalone application responsible for the system's core logic (e.g., event processing, component mapping, etc.). Routing-specific services are implemented as operator-tailored modules, which can use network-wide information to deliver arbitrary, high-level routing logics (e.g., load balancing, preferred exit points, etc.).
- RF-Proxy: Shim 'proxy' application on top of an OpenFlow controller that deals with the switch interactions and collects network state via topology discovery and monitoring applications.

In a straightforward mode of operation, a single datapath switch is naturally associated to a single virtual router, however different ways are also possible: (i) a multiplexed mode, where datapaths are sliced and controlled by more than one virtual router, and (ii) an aggregate mode, where a single virtual router represents a group of datapaths. The aggregate mode of operation (detailed in [5]) enables a single router abstraction to realize domain-wide routing control platforms using eBGP.

To act as a router, OpenFlow rules match on destination IP and MAC addresses, and perform MAC re-writing based on the next hop information. For OpenFlow 1.3 devices, multiple flow tables and group tables are used to install the resulting flow-based FIB entries. Besides being the API to the datapaths, OpenFlow is used as the vehicle to deliver control plane messages from/to the virtual interfaces of the virtualized OS running the routing engine via a virtual switch such as Open vSwitch.

In line with the best design practices of cloud applications, RouteFlow relies on a scalable, fault-tolerant datastore that centralizes (i) the 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 route control applications. Either ZeroMQ or a distributed NoSQL database (MongoDB) can be used as the pubsub-like message queuing inter-process communication (IPC) that loosely couples the modules via an extensible JSON-based implementation of the RouteFlow protocol.

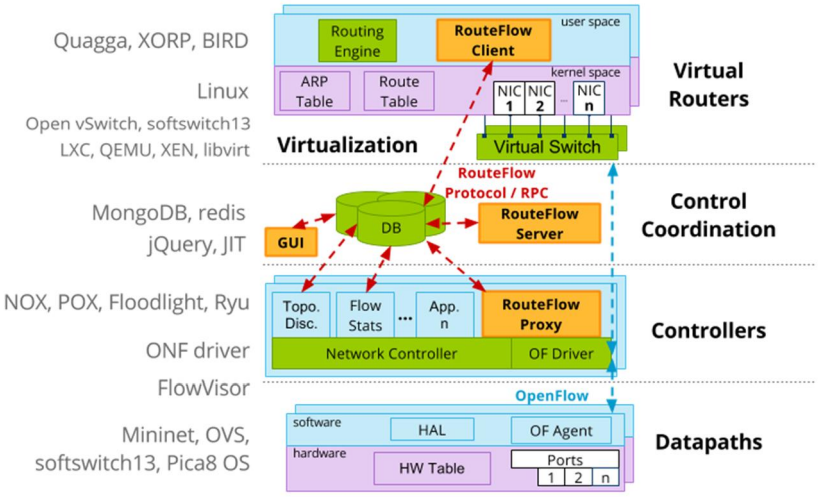

Figure 2. The RouteFlow architecture introduces three software components (RF-Proxy, RF-Server, and RF-Client) into a modular design of routing-oriented SDN stack based on many FLOSS solutions listed on the left side.

B. Overcoming the lack of Standardized Northbound Interfaces and heterogeneites of Southbound OpenFlow Interfaces

The absence of standardized northbound APIs motivated our work on unifying abstractions – basically route-centric simplified OpenFlow messaging – to ease the development of the shim RF-Proxy application for multiple controllers (e.g., Ryu, POX, Floodlight, ODL) while keeping the remaining modules unmodified and independent from the southbound protocol version (further details of the TLV-based, JSON-ready *routemod* northbound API are available at: http://wand.net.nz/~jps22/routemod.pdf). RF-Protocol was introduced to act as an IPC that glues together the different modules with a simple command/response JSON syntax northbound to the RF-Clients and a subset of OpenFlow messages southbound to RF-Proxy.

The result is a hierarchical design that does not depend on the specific controller of choice and is agnostic to the OpenFlow version supported by the forwarding devices. Further work is expected to effectively interact with heterogeneous OpenFlow 1.3 devices as their capabilities and set of implemented optional features (e.g., multiple tables, group table types) may critically differ – an issue already in scope of the ONF Forwarding Abstractions WG. We also expect potentially useful IP routing-oriented APIs from related work at IETF I2RS despite the focus on in-box routing stacks and distributed control planes.

Altogether, the RouteFlow design tries to follow principles that allow architectural evolvability: layers of indirection, system modularity, and interface extensibility. The resulting architecture allows the distribution of the environment in an arbitrary fashion with regard to the number of RFProxy instances, hypervisor switches and RFClient instances. The routing engine and SDN API specifics are abstracted away from the core logic through the RFProtocol northbound APIs.

IV. CARDIGAN: ROUTEFLOW GOING LIVE AT AN INTERNET EXCHANGE

Some of the challenges faced by real-world SDN solutions are, at least, threefold: (i) they must be able to seamlessly replace already deployed technology; (ii) they must not impose any performance penalties to the operation; (iii) they must offer a truly game-changing innovation over traditional operations. Cardigan is the code name of the efforts that leverage RouteFlow to bust the challenges around SDN in production.

The Cardigan deployment project [6] began as a simple network: two OpenFlow switches in different buildings controlled by a single controller, peering with the Wellington Internet Exchange (WIX) on one side and the REANNZ office network on the other, but appearing together as a single router. As shown in Figure 3, the system is directly connected to the live Internet through the WIX point, and its configuration is managed from a single point where routes are added, translated and distributed to the switches.

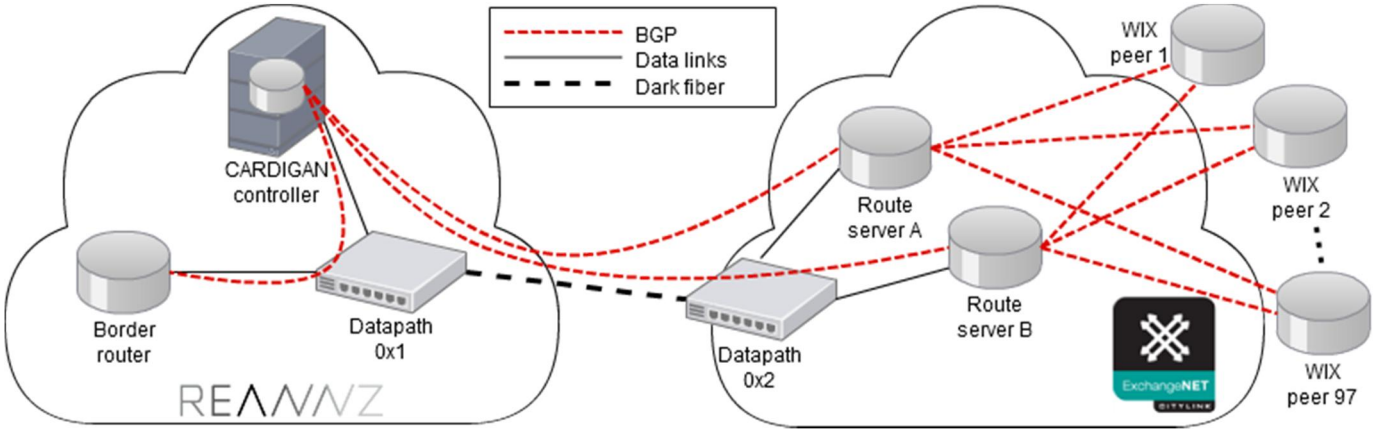

Figure 3. The logical router interconnection between REANZ and WIX is implemented with the Cardigan controller (running a tailored version of RouteFlow) to control two OpenFlow switches (Datapath 0x1 and 0x2) physically interconnected through dark fiber.

This network provides a platform for practical SDN research in a realistic hybrid/live deployment, peering with non-SDN devices and connected to the Internet directly. Cardigan has been running in production for over one year, and while the initial deployment was very simple and modest sized,[2] we argue that the way to make progress towards realizing SDN is practical, production deployment. Even basic systems need to get as early as possible in production to perform actual work outside the lab and learn from the experiences as the develop-deploy cycle continues.

Cardigan uses a fork of RouteFlow called Vandervecken (http://github.com/routeflow/RouteFlow/tree/vandervecken), which aims at continuing the consolidation of production-based research. All flows are programmed proactively based on the RouteFlow IP route and ARP table computations, and all other forwarding is by default denied, limiting the controller's exposure to DoS attacks. As of the time of writing, there are 1138 flows on each switch, broken down as follows: 8 flows tunneling control plane

[2] Some numbers on WIX: 90 organizations peer through it; most organizations handle up to 500 IPv6 and 1000 IPv4 network announcements from its peers; most organizations are IPv6-ready. More details on WIX can be found at: http://wix.nzix.net/peers.html

traffic (e.g., ARP, ICMP, BGP) to the controller, and one rule to drop traffic by default; 98 flows describing directly connected hosts, at the WIX and at REANNZ; and 1032 representing layer 3 routes. The aggregate traffic fluctuates around 100s of Mbps and changes at the IXP provide 3-4 updates every ten seconds, due to ARP timeouts and link changes. Given the live nature of the deployment, deeper performance analysis was not conducted yet.

Support of OpenFlow 1.3 has been recently added as a critical step to work with MPLS and IPv6 in addition to the efficient implementation of IP next hop forwarding via group tables. Beyond L2/L3 interworking, basic label switch router (LSR) interoperability has been achieved. The Cardigan network now spans three continents, and has nodes in the US (at ESnet), Australia (CSIRO) and in New Zealand (REANNZ and VUW) – with Brazil (RNP) coming up next.

V. Challenges and Future Work

Undoubtedly, there are a number of barriers to be broken before we see more SDN deployments. Technical challenges, like scalability, performance, dependability and product maturity, have been discussed elsewhere [10] and are being addressed by the academic research community and the ONF industrial partners.

Plans for the RouteFlow SDN routing stack include fully support of OpenFlow v1.3.4, IPv6, and RSVP-TE. Experiments with different OpenFlow data plane implementations (e.g., ASIC, NPU, SW, and hybrid approaches) are going on to investigate scalability in terms of number of flows, rate of state changes, amount of devices, and distance between distributed devices.

At the controller level, high availability to RouteFlow is being added including OpenFlow Master/Slave/Equal role support. We plan to keep interoperating with several additional controllers, among them On.Lab's ONOS and OpenDaylight. Altogether, the system needs to be resistant to DoS attacks; it must be not only easy but easier than managing a network of separate routers; and it must be more reliable and easier to fix. In addition to taking advantage of common software engineering practices, the path ahead includes leveraging SDN research on formal methods to build provably correct networks and systematically troubleshooting.

Furthermore, we are researching practical questions of operating an Internet of SDN networks including suitable SDN-SDN protocols, ie., how should SDNs in different administrative domains communicate beyond BGP. We are devoting efforts to the vision of software-defined IXPs [11], where participants may peer based on topology, metrics, and policies far more expressive than BGP.

VI. Conclusion

The SDN horizon promises a change on how to design, build and operate networks to invigorate innovation via open, vendor-agnostic architectures with a clear decoupling of the data and control planes. Open source and SDN are critical partners in moving the networking industry forward. This article discusses significant progress in the number and diversity of open source SDN projects and traction with focus on the RouteFlow architecture and operational experiences like Cardigan. Regardless, there are still significant challenges before we see widely deployable open source SDN stacks in production. Some of them relate to the evolving role of standardization around SDN and the availability and readiness of northbound and southbound interfaces.

Acknowledgment

We would like to thank David Meyer and Inder Monga for insightful comments. We are also thankful to anonymous reviewers of earlier versions of this paper. We thank Diego Kreutz for his contribution to Figure 1. This work was partially supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil.

References

[1] D. Meyer, "The Software-Defined-Networking Research Group," Internet Computing, IEEE , vol.17, no.6, pp.84,87, Nov.-Dec. 2013

[2] N. McKeown et al.. "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev. 38, 2, 69-74, Mar. 2008.

[3] T. Nadeau and K. Gray, "SDN: Software Defined Networks", O'Reilly Media, Oct. 2013.

[4] S. Hares and R. White, "Software-Defined Networks and the Interface to the Routing System (I2RS)," IEEE Internet Computing, vol. 17, no. 4, pp. 84-88, July-Aug. 2013

[5] C. Rothenberg et al. "Revisiting routing control platforms with the eyes and muscles of software-defined networking," In HotSDN'12 Workshop, Aug. 2012.

[6] J. Stringer et al. "Cardigan: Deploying a Distributed Routing Fabric," In HotSDN'13 Workshop, Aug. 2013.

[7] S. Jain et al. "B4: Experience with a Globally-Deployed Software Defined WAN," In Proc. of ACM SIGCOMM 2013.

[8] J. Lindman et al., "Matching Open Source Software Licenses with Corresponding Business Models," Software, IEEE , vol.28, no.4, pp.31,35, Jul-Aug. 2011

[9] SDN Central, Comprehensive List of Open Source SDN Projects. Available online: http://www.sdncentral.com/comprehensive-list-of-open source-sdn-projects/

[10] S. Yeganeh et al., "On scalability of software-defined networking," Communications Magazine, IEEE, 51(2):136–141, February. 2013.

[11] N. Feamster et al., "SDX: A Software Defined Internet Exchange," In ACM SIGCOMM, 2014.

AUTHOR BIOS

Christian Esteve Rothenberg is an Assistant Professor at University of Campinas (UNICAMP), Brazil, where he received his Ph.D. in Electrical and Computer Engineering in 2010. From 2010 to 2013, he worked as Senior Research Scientist in the areas of IP systems and networking at CPqD R&D Center in Telecommunication, Campinas, Brazil. He holds the Telecommunication Engineering degree from the Technical University of Madrid (ETSIT - UPM), Spain, and the M.Sc. (Dipl. Ing.) degree in Electrical Engineering and Information Technology from the Darmstadt University of Technology (TUD), Germany, 2006. Since April 2013, he is an ONF Research Associate. Christian has been the technical leader behind a number of successful open source SDN projects such as RouteFlow, ofsoftswitch13, and libfluid. He has two international patents and over 50 publications in scientific journals and top-tier networking conferences such as SIGCOMM and INFOCOM.

Roy Chua is co-founder and partner at SDNCentral.com, a resource site dedicated to serving the networking community by providing coverage of news, educational resources and community events in SDN, NFV and network virtualization. He is also a partner at Wiretap Ventures, a management consultancy focused on working with cloud, SDN and security companies. Roy has spent the last 20 years in engineering and marketing in technology companies in Silicon Valley, bringing innovative networking and security products to the market. Roy holds an MBA from MIT's Sloan School of Management and Masters and Bachelors degrees in Electrical Engineering and Computer Science from the University of California at Berkeley.

Josh Bailey is a Google software engineer based at Victoria University of Wellington in New Zealand. He's been with Google over 6 years, most of that time based in Mountain View, California. Josh is interested in big high performance networks and big high performance computers.

Martin Winter is a Technical Lead and co-founder of the Network Device Education Foundation (NetDEF) in California. OpenSourceRouting is a non-profit project of NetDEF working on improving Quagga to get it into an alternative to commercial routing solutions and help to ignite the open source community for routing platforms. On the side he serves as a Chair for the Open Source Working Group at RIPE (European Internet Provider Forum). Prior to the current role, Martin Winter worked many years as a Technical Leader at Cisco in IOS Development and the High-End Routing division and as a Network Architect for a large worldwide colocation provider.

Carlos N. A. Corrêa holds a Master's degree in Information Systems from Federal University of the State of Rio de Janeiro. Currently, he is a PhD candidate at Federal Fluminense University. He works as a part-time professor for graduate and undergraduate courses in Brazil. He is also an IT Operations manager and consultant, leading IT Operations teams for companies in Brazil, Africa and the USA. Carlos co-authored several research papers on network virtualization and OpenFlow, presented on peer-reviewed IEEE and ACM conferences.

Sidney C. de Lucena received a degree in eletronic engineering (1995) and M.Sc and D.Sc degrees in computer and systems engineering (1997 and 2004) from Federal University of Rio de Janeiro. From 2001 to 2008 he worked as a network analyst at RNP, the brazilian NREN. Currently, he is an Assistant Professor in the Information Systems department at Federal University of the State of Rio de Janeiro. Research interests include network management and software-defined networking.

Marcos Rogério Salvador is Research Collaborator at the State University of Campinas (UNICAMP) and Sr. Manager of Innovation at the Lenovo Innovation Center in Brazil, where is responsible for driving innovation and collaboration with the academia towards state-of-the art datacenter infrastructure technology and products. He has more than 15 years of experience in research and development of optical and packet network technologies and products on behalf of local and global manufacturers. An ONF Research Associate, Marcos managed the (ex-CPqD) team that won the ONF Driver Competition and also developed (together with Ericsson) the OpenFlow 1.3 software switch, used for prototyping new OpenFlow extensions and RouteFlow, the first IP routing solution for OpenFlow networks.

Thomas D. Nadeau is a Distinguished Engineer at Brocade where he is The Chief Architect of Open Source in the Software Business Unit. Tom runs teams responsible for building commercial products based on open source, as well as contributions to upstream open source projects such as Open Daylight and OpenStack. Tom is also a member of Brocade CTO Staff and is co-chair of the IETF NETMOD Working Group. Prior to Brocade, Tom was a Distinguished Engineer at Juniper Networks where he focuses on SDN and new data center and switching products. Prior to that, Tom worked in Juniper's PSD CTO Office where he was responsible for leading all aspects of Software Defined Networks and Network Programmability. Tom has a new book out called SDN: Software Defined Networks, An authoritative Review of Network Programmability Technologies on O'Reilly Publishers.

CONTACT INFORMATION

Christian Esteve Rothenberg
DCA/FEEC/UNICAMP
Av. Albert Einstein - 400, Cidade Universitária Zeferino Vaz
CEP: 13083-852, Campinas, SP, Brazil.

Email: chesteve@dca.fee.unicamp.br
Tlf.: +55 (19) 3521 3778

Thomas D. Nadeau
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134

Phone: +1.603.502.0042
Email: tnadeau@brocade.com

Carlos Correa
Rua Sorocaba, 411/Apto 409
Botafogo, CEP 22271-110
Rio de Janeiro, RJ – Brazil

Email: carlos.nilton@gmail.com

Sidney Cunha de Lucena
Universidade Federal do Estado do Rio de Janeiro,
Centro de Ciências Exatas e Tecnologia,
Av. Pasteur, 458
22290240 - Rio de Janeiro, RJ - Brazil

Phone: +55 (21) 25308051
Email: sidney@uniriotec.br

Marcos SALVADOR
Lenovo Innovation Center
Av. Alan Turing, 776
Cidade Universitária,
13083-898, Campinas, SP, Brazil

Email: mrsalvador.corp@gmail.com

Roy Chua
SDNCentral LLC
955 Benecia Avenue
Sunnyvale, California 95070
Email: rchua@sdncentral.com

Josh Bailey
School of Engineering and Computer Science
Victoria University of Wellington
Room AM410
Wellington 6140, New Zealand

Email: joshb@google.com

Martin Winter
2464 El Camino Real Apt 434
Santa Clara, CA 95051, USA

Email: mwinter@noaccess.com