Cardigan: SDN Distributed Routing Fabric Going Live at an Internet Exchange

Jonathan Stringer
Dean Pemberton
Qiang Fu
Victoria University of Wellington - New Zealand
qiang.fu@ecs.vuw.ac.nz

Christopher Lorier
Richard Nelson
University of Waikato
Hamilton, New Zealand
cml16@waikato.ac.nz

Josh Bailey
Google
New Zealand
joshb@google.com

Carlos N. A. Corrêa[1]
Christian Esteve Rothenberg[2]
[1]Fluminense Federal University - UFF
[2]University of Campinas - UNICAMP
Brazil
chesteve@dca.fee.unicamp.br

***Abstract*—Software Defined Networking (SDN) is an active area for network research, with many organizations exploring the opportunities provided by the decoupling of network control from packet forwarding. However, questions remain over the operation of such systems in production. In order to build operational confidence, we built Cardigan –a distributed router using OpenFlow– and deployed it at a public Internet exchange. Cardigan applies a routing as a service abstraction to a RouteFlow controlled IP network in an effort to reduce operational complexity. The implementation and deployment efforts provide insights into the challenges involved with using these technologies, and suggests the viability of mixed device environments despite the limitations of early OpenFlow implementations.**

I. INTRODUCTION

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 a similar question about Software-Defined Networking (SDN); if really so useful and such an advance, why have we not seen it deployed more widely in production networks? Some of many reasons why may be (*i*) the need to directly address operational comfort with SDN, (*ii*) the need to provide concrete benefits, and (*iii*) the need to demonstrate a practical migration path (not to mention the ability to interoperate with non-SDN networks). Some have been in the hunt for the elusive SDN "use case" or "killer app", but perhaps the answer is right under our noses –simpler, more reliable, and easier to operate networks. Our ongoing efforts within Cardigan go some way to address these questions.

SDN in general, and OpenFlow [1] in particular, have unlocked many new tools for re-imagining our approach to layer 3 networking [2, 3, 4, 5]. However, these technologies are in their infancy, and their unproven nature and misconceptions caused by a lack of familiarity with SDN for real have meant these technologies are seeing little or slow use in production. Cardigan is a project to generate confidence in SDN by deploying an OpenFlow-based networking environment in a production setting.

Our deployment efforts are helping to identify practical issues with the roll out of SDN environments, detecting any incompatibilities with legacy networking devices and protocols, and finding clues to possible implementation barriers for future wider deployments. Furthermore, and maybe more importantly, there needs to be motivating reasons to adopt any new technology. While hardware commoditization is expected to drive costs down, this alone may not provide enough benefit to warrant the replacement of existing hardware. Going live is a crucial first step to validate the viability of technology and assess the advantages that SDN can offer the WAN.

In an effort to reduce operational complexity, Cardigan applies a routing as a service abstraction [6] to a RouteFlow [5] controlled IP network. Configuration needs only to be applied once for multiple devices, and the simplified structure of the network reduces configuration for all devices. This saves operator time and reduces the likelihood of misconfiguration. The simplified structure of the network also makes it easier to understand, aiding modification and diagnosis of problems, thereby providing a direct ongoing benefit to the maintenance of the network. Google's B4 [7] inter-datacenter network based on SDN/OpenFlow is a remarkable example of the opportunities of SDN in the WAN. When exercised in the context of an Internet eXchange Point (IXP), we expect unleashed innovation in the inter-domain routing landscape, tackling current hard issues such as security and economics of IXPs (cf. Software-Defined Internet Exchange [8]).

The contributions of this paper are threefold. First, we present the design and implementation of a SDN-based distributed routing fabric that advances the state of the art in Internet router designs. Second, through evaluation of a pilot deployment interconnecting an IXP with an NREN, we demonstrate the viability of SDN migration through "drop-in" replacement of network hardware. Third, we identify incompatibilities and issues in production environments and present our lessons learned, including implementation barriers to wider deployment – experiences similar to those that the active SDN community is likely to face when going live.

The paper is organized as follows. Section 2 presents basic background information on the RouteFlow architecture. Section 3 describes the design considerations adopted within Cardigan to overcome identified issues and meet the operational requirements. Section 4 presents the deployment of Cardigan in a pilot environment and reviews some of the challenges encountered. Section 5 discusses the identified and pending issues pertaining to the deployment. Section 6 presents related work and, finally, section 7 concludes the paper with final remarks and a brief outlook on our next steps towards materializing software-defined IP routing architectures.

II. ROUTEFLOW PRIMER

In a nutshell, RouteFlow was born as a thought-experiment to glue Linux-based routing engines with OpenFlow-based datapaths. Although altogether it performs the same function of a router, it is implemented with virtualization and SDN control abstractions in mind. The RouteFlow architecture consists of three independent components that manage particular aspects of controlling datapaths using virtualised routing engines. Figure 1 shows the relationship between these components.

- **RFClient** is a simple daemon that runs in any Linux-based system and listens to route and ARP table updates via Netlink. When RFClient joins the RouteFlow topology, it is associated with a particular datapath. The events that are collected are sent to RFServer for processing and propagation towards the associated datapath(s). RFClient is typically run inside a Linux Container attached to a hypervisor switch which allows control plane traffic to be sent to it.

- **RFServer** is a standalone application responsible for the core system logic. Customisation of the RFServer allows implementation of different services and modes of operation. The core state of RouteFlow corresponds to the available components (datapaths, routing engines, controllers, virtual switches, etc.), their current mapping and configuration. This state is held externally in a NoSQL database.

- **RFProxy** is an OpenFlow controller application to manage the interactions with OpenFlow switches and abstract SDN protocol specifics away from the rest of the architecture. It listens to flow updates from RFServer and propagates them to the desired datapath switches.

These components are tied together with a single event notification and flow propagation protocol called RFProtocol. This allows messages from different routing engines to be shared between RouteFlow components and mapped to different SDN APIs. RFClient handles conversion from Netlink to RFProtocol, and RFProxy translates these messages into different OpenFlow versions, depending on the controller used.

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 RFProtocol. To act as a router, OpenFlow rules match on destination IP and MAC addresses, and perform MAC re-writing based on the next hop. In the OpenFlow 1.3 prototypes multiple flow tables and group tables are employed.

III. CARDIGAN: SDN ROUTING FABRIC

Both the SDN approach, in general, and the OpenFlow architecture, in particular, face strong skepticism when opposed to traditional networks that have been running for decades. SDN advocates claim that a programmatic interface could increase the efficiency and utility of the commoditized "packet-shipping" service. That bears some resemblance to the DevOps/NetOps movements, with one fundamental difference: being end systems, server operations can be changed as long as their services keep running; an Internet-connected production network, however, is tightly coupled to the way other networks operate in terms of their active routing protocols.

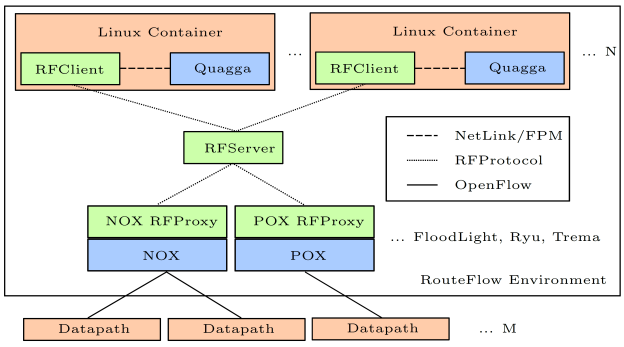

Fig. 1. RouteFlow block diagram.

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 our efforts that leverage RouteFlow to bust the challenges around SDN in production.

This section starts by reviewing the initial Cardigan design –first presented in [9]– that served as a pilot deployment to prove the concept of aggregating multiple OpenFlow switches to perform as a single logical layer 3 device, but faced a series of limitations. We then move to the introduced enhancements, with special focus on the newly introduced features to provide a better abstraction for the routing of an administrative domain. The enhancements include a straightforward way of defining "packet circuits" for data transport inside the AS, by means of a MPLS LSP; and breaking the RouteFlow dependency on the Linux kernel FIB, through a mechanism that allows hooking into the Quagga RIB to access the full set of routes obtained by its routing protocols.

A. Cardigan 101

To generate operational confidence in SDN, it is not adequate to demonstrate the ability to replace already existing technology with SDN sevices; SDN must offer a practical advantage over traditional networking. To demonstrate the flexibility of SDN and benefits of centralized control, Cardigan introduced extensions to RouteFlow aiming for the experimental operation of an IXP-connected network through SDN. We consider this a pertinent enterprise because it encompasses all the challenges previously discussed: (*i*) the ability to exchange routes with external ASes is a must; (*ii*) the computational effort and performance pitfalls on computation and distribution of routing information were extensively studied; and (*iii*) the idiosyncrasies and administrative burden of BGP configuration and tuning are well-known.

Early efforts on RouteFlow as a routing control platform [5] shed light on potential SDN-backed innovation to routing services operation through an aggregated mode of operation where management of routing in an emulated AS was defined through a single BGP process. The model being pursued was a "BGP-free edge" design but lacked the implementation of complete intra-domain SDN forwarding solution.

The first switch aggregation strategy within Cardigan was managed with the static configuration in RFServer specify-

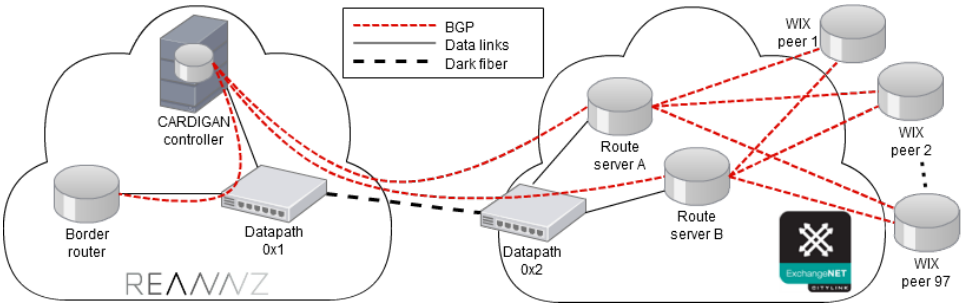

Fig. 2. The Cardigan initial 2-switch design was deployed live, providing a logical router between REANNZ and WIX.

ing inter-switch links. RFServer required the switches to be arranged in a full mesh – like router line cards and fabric cross-connects – and connected to controllers communicating to the same RFServer instance. RFServer communicated with a single RFClient and as RFServer received route modification messages it distribute the flows among the switches, ensuring correct forwarding behaviour.

Figure 2 shows the first Cardigan deployment, connecting the Research and Education Advanced Network of New Zealand (REANNZ) to the Wellington Internet Exchange (WIX). The RouteFlow distributed router was built upon two OpenFlow switches, connected by a dark fibre link, one situated at the border of each network.

OpenFlow entries installed by Cardigan are based on a hierarchy of rules following a proactive flow installation approach, not reactively involving the controller. In addition, all forwarding is default deny, limiting the controller's exposure to DoS attacks. The highest priority rules provide high-level blocking of entire classes of traffic. For instance, packets which do not contain the appropriate layer 2 address will be immediately blocked. The next priority of rules is for control traffic, which must be destined for the controller IP and be explicitly allowed to be passed to the control plane (e.g., ARP, BGP, OSPF, IS-IS). Following these, there is a set of priorities for Hosts and Routes. These are sorted in order of prefix to implement longest prefix matching; longer prefixes adopt a higher priority in the rule table than shorter prefixes.

Cardigan uses a fork of RouteFlow called Vandervecken[1] which aims at continuing the consolidation of production-based research. Support of OpenFlow 1.3 has been recently added as a critical milestone to work with MPLS and IPv6 in addition to the implementation of efficient IP next hop forwarding with group tables.

B. Limitations and easy fixes

The early design and deployment of Cardigan revealed some limitations in the software stack implementation that lead to a number of code refactoring. More importantly, we identified scalability issues related to the datapath aggregation design that could severely impair real-word deployments. We first discuss two of the practical limitations observed and describe the adopted fixes. Then, we move the focus to the problem of realizing a scalable router abstraction.

[1] Available online at: https://github.com/routeflow/RouteFlow/tree/vandervecken

1) Extensible message formats: A new message format in RFProtocol called *extensible route modification* was introduced to support a more flexible combination of protocols between layer 2 and 3. This was developed to extend the granularity of flow specification and minimise development effort for further protocol support. This allows new message formats to be incrementally added, without breaking existing messages. Similar to the `FlowMod` message in OpenFlow, the new format includes a set of matches and actions. Furthermore, it provides a generic set of options to allow RouteFlow components to share additional information such as flow priority. This format includes support for IPv4, IPv6, MPLS on Ethernet, and ingress port matching. In Cardigan the latter of these was used to support unique MAC addresses for each port.

2) Inefficient gateway resolution: We found out that the each route collection in RouteFlow was done in two subsequent stages: the reception of its Netlink announcement, and the discovery of the MAC address of the associated gateway (if it wasn't present at the VM's FIB). In the event when an address resolution was taking too long, other route operations received in the same time frame would be needlessly delayed. The fix to this issue was to cache IPv4 and IP6 routes before they are sent to datapaths. This allows RouteFlow to handle routes with valid gateways as they are received, and queue other routes until their respective gateways' MAC addresses are received. A gateway lookup thread iterates through the queue, sending non-blocking ARP requests and submitting routes with resolved information to RFServer. This refactor also added some level of thread-safety to RF's internal data structures, but that wasn't the main scope of this effort.

3) Scalable router abstraction: While proof of concept implementations [5, 9] showed the viability of simplifying the management of routing by aggregating BGP route computation into less processes and aggregating distributed switches to act as logical entities, some limitations of the work made it an improbable choice for a production network: (*i*) the router abstraction was completely dependent on the physical topology; (*ii*) it required a full-meshed physical topology, relying on static one-hop paths for data transmission, which means it is not possible to define an arbitrary (software-defined) paths inside the ISP network; (*iii*) it classifies traffic by using VLAN tags as per the static inter-switch links (ISLs) configuration. As we will discuss next, by introducing MPLS paths, we avoid the risk of loops appearing due to asynchronous updates of OpenFlow switch entries and empower the network operator with a scalable solution allowing fine granular traffic control.

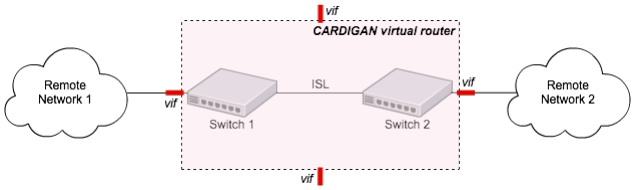

Fig. 3. Cardigan as a virtual router built upon arbitrary OpenFlow switches.

C. Cardigan 2.0: Towards Game-changing Innovation

It became obvious that an extension to RouteFlow was necessary when the Cardigan project considered ways to move beyond a two-switch model towards not only an *n*-switch model, but a model where the physical network topology could be arbitrarily defined, automatically discovered, and adapt to changing network conditions. This vision is illustrated in Figure 3. Packets accepted on an ingress datapath element would need a method of being transmitted across this fabric to the egress port defined within the RouteFlow control plane. The ISL mechanism to perform inter-switch packet forwarding lacked both an ability to scale beyond two devices, as well as an ability to introduce more advanced Traffic Engineering functionality down the track. Another mechanism was required to allow the control plane to define a path through the Cardigan network and have this signaled on the datapath through OpenFlow messages.

To accomplish this MPLS was used in the Cardigan network. It should be noted that this is only as a method of associating a given prefix to a given path, or set of paths through the network. Other mechanisms (VLAN tags, IP in IP [7]) or a combination of mechanisms could be just as easily employed to serve the same purpose. In essence, an MPLS Label Switched Path (LSP) was defined from a given ingress node, through a set of transit nodes, to an egress node. Each of these devices receives the flow entries installed to allow the packets to flow along this LSP.

Changes were made the the RouteFlow framework in a number of places. In place of an input describing the physical topology (ISL ports etc), RouteFlow was also changed to consume a dataset specifying the hop-by-hop behaviour for MPLS packets through the core of the Cardigan network, as shown in Figure 4. In this way, OpenFlow rules which perform the roles of Ingress, Transit and Egress LSRs can be pushed to the data path. LSP labels are programmaticaly attached to packets on their ingress datapath, where the full set of OpenFlow matches can by applied to their header. The Src and Dst MAC addresses are also changed to their 'post routed' values at ingress to allow packets to simply have their MPLS header removed on egress. This mechanism allows an operator to freely define packet circuits over the network, based on packet's headers, in an arbitrary level of detail.

A Cardigan-based IXP should not be restricted to the BGP route selection process. It was already shown to be a serious problem to reconcile the business objectives and the operation of a network based on it [10]. Early efforts on RouteFlow [5] were useful to open our eyes on how SDN programmability could flexibilize IP route selection, but the Netlink interface used by stock RouteFlow advertises only BGP-selected routes. To overcome this limitation we added a Forwarding Path

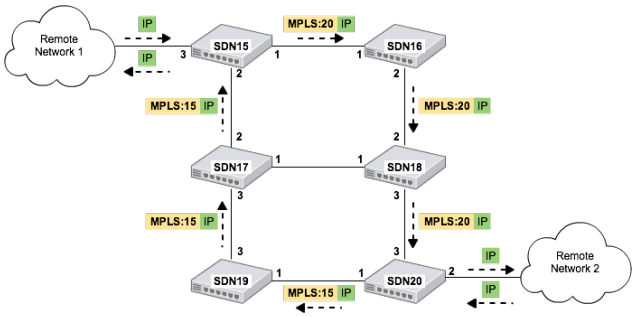

Fig. 4. Cardigan with MPLS design.

Manager (FPM) component to the RouteFlow architecture.

Quagga, the main routing engine supported by RouteFlow, uses FPM to announce all routes obtained by its routing processes. Through the collection of FPM messages, Cardigan can feed to the RouteFlow database even routes that were initially discarded by the BGP daemon (and would otherwise never be known). These routes can then be arbitrarily marked for dissemination through RFProxy.

Given the abilities to define the forwarding path inside a network and to freely choose between available routes, we believe the current design is a step forward towards fully materializing the concept of a software-defined IP routing.

IV. DEPLOYMENT

Cardigan deployment consists of a RouteFlow distributed router, connecting the REANNZ to the WIX. RouteFlow controls two OpenFlow switches, connected by a dark fibre link, one situated at the border of each network (see Figure 2). The switches used were a Pronto 3290 at WIX and a Pronto 3780 with 1G SFPs at REANNZ, each using PicOS 1.6. The controller was deployed at REANNZ in a VM, connected to both switches via an out-of-band layer 2 VLAN. BGP peer sessions were established with a router running within REANNZ and all WIX participants via two route servers. Routes to the REANNZ network were advertised onto the WIX and traffic was forwarded directly through the OpenFlow switches.

Cardigan has been deployed in production for over nine months, forwarding customer traffic and sharing routes with ninety other participants of WIX without major incidents.[2] As of the time of writing there are 1134 flows on each switch, broken down as follows:

- 8 flows tunneling control plane traffic (e.g., ARP, ICMP, BGP, etc.) to the controller, and one rule to drop traffic by default
- 98 flows describing directly connected hosts, at the WIX and at REANNZ
- 1028 representing layer 3 routes.

[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

The boot time for Cardigan is around one minute, including flow installation. The major bottlenecks for this are surrounding bootup of the NoSQL database/IPC backend and initial resolution of next-hop MAC addresses for routes. Initial throughput measurements were performed using Iperf between hosts connected to REANNZ and CityLink as indicated in Fig. 2. These showed modest TCP performance across the path in the order of ≈800Mbps, peaking at 855Mbps. Given the live nature of the deployment, deeper performance analysis was not conducted. Ongoing updates from the IXP provide 3-4 updates every ten seconds, due to ARP timeouts and link changes.

While the original deployment was very simple and modest-sized, it is argued that the way to make progress towards realizing SDN is practical, production deployment. Even basic systems need to get as early as possible into production to perform actual work outside the lab and learn from the experiences as the develop-deploy cycle continues.

The upcoming Cardigan network will span three continents, with nodes in the US (at ESnet), Australia (CSIRO) and in New Zealand (REANNZ and VUW), and Brazil (RNP) coming up next. The network aims at providing a platform for practical SDN research in a realistic hybrid/live deployment, peering with non-SDN devices, and connected to the Internet directly.

V. DISCUSSION

Cardigan deployment at WIX has revealed several SDN implementation issues, but also demonstrated that for a deployment of this size it is an adequate replacement for legacy devices. For larger deployments, we have identified a number of obstacles that need to be addressed such us (ephemeral) limitations in current implementation of SDN stacks, scalability concerns. and the lack of controller redundancy. Furthermore, we have observed incorrect behaviour in the WIX, caused by a lack of policy enforcement, an avenue worth to explore in sought of innovative traffic control at IXPs.

A. Implementation and protocol considerations

1) Protocol Compliance: RouteFlow implementation with OpenFlow 1.0 has minor violations of the protocol specifications of Ethernet and IPv4. While these issues were not critical to the functioning of Cardigan, they remain important considerations for future deployments in OpenFlow 1.0 environments. The inability to decrement TTL when forwarding IP datagrams could cause a failure of IP loop prevention – for instance if the loop existed between two instances of Routeflow. This is caused by the lack of support for TTL decrement in OpenFlow 1.0. This issue is resolved from OpenFlow 1.1 onwards.

2) MAC Addressing: Another concerning behaviour was the use of a single MAC address to identify the entire distributed router. The EUI-48 standards specify that identifiers are "intended to identify items of real physical equipment or parts of such equipment such as separable subsystems or individually addressable ports." [11] This was corrected for Cardigan deployment, but, for more complicated scenarios, numbers of addresses would be limited by the scalability of OpenFlow 1.0 flow tables. For each route learned, $n-1$ rules must be installed, where n is the number of different MAC addresses on the router. For these (and other) reasons we recommend the use of OpenFlow 1.1 or higher, and advise vigilance when implementing protocol behaviour for those developing hybrid SDN applications.

TABLE I. COMPARISON OF FIB CONVERGENCE TIMES.

Gateway Resolver	Time Per Route		
	Max	Min	Mean
Original	180s	20ms	3s
Revised	1s	¡1ms	0.01s

3) OpenFlow Agent Implementation: The vendor switches were also a source of minor issues, in particular relating to their Open vSwitch customization to act as OpenFlow agents. Various bugs such as memory leaks and flow counters that do not update were discovered and reported to the vendor. We expect that as implementations mature, this will become less of a problem.

4) Encapsulation Hazards: Encapsulation of packets forwarded to the controller caused minor setbacks. Care must be taken to ensure that encapsulation size limits include not only path MTU, but allow for Ethernet, VLAN, MPLS, etc. headers as well. We were receiving truncated BGP route updates when synchronizing our Routing Information Base (RIB) with WIX because initial OpenFlow packet-in encapsulation size was 1500.

5) Gateway Address Resolution: As discussed in Section III-B2, the time to resolve the gateway address caused a serious issue in datapath Forwarding Information Base (FIB) convergence – a result of poor separation of concerns and invalid assumptions in RFClient. With the revised version of the gateway resolver, we were able to increase performance by two orders of magnitude. The original version would block on address resolution, which would slow all route processing down. Even when the resolution function was replaced with a non-blocking alternative, routes would be lost due to buffer size limits on the Netlink socket.[3] The appropriate solution was to separate gateway resolution from route processing; gateways whose layer 2 addresses have been resolved can be immediately handled, while the others can be queued until they can be resolved and propagated. Table I compares the convergence times between implementations. For the WIX RIB, the initial implementation would converge in approximately one hour. After modifications were made to the gateway resolution process, this was reduced to less than ten seconds.

B. Scalability

For RouteFlow to become truly viable, it must be proven that it can be deployed in larger environments with greater volume of routes. Common experience is that current hardware supports in the order of low thousands of flows. FIB compression may mitigate this limitation, but flow table sizes are still likely to exceed the capabilities of current hardware. FIB caching [12] and the distribution of the FIB across multiple devices will be investigated for their suitability in the context of OpenFlow. The ideal solution to this issue, however, is for hardware to support significantly larger FIBs.

Experiments with different OpenFlow data plane implementations (e.g., ASIC, NPU, software, and combination approaches) are going on to investigate scalability in terms of

[3]With the default size of 1MB, storing around 1K routes

increasing number of datapath flows, network state changes, forwarding devices, and distance between distributed devices.

C. Resilience

A larger deployment with a more complicated topology also will put greater strain on the resilience of the system. RouteFlow has no provision (yet) for redundant controllers, support for OpenFlow Master/Slave/Equal roles is currently being added. The switch will continue forwarding if the connection to the controller is lost, but all rules will be cleared when the controller reconnects. RouteFlow also does not determine the activity of individual ports, which may result in forwarding black-holes. All these issues are on our roadmap towards a high-available non-stop forwarding solution.

Overall, 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 (e.g. by active verification of the distributed data plane, perhaps in a similar way to how connectivity can be automatically verified by COTS testing in the PSTN). In addition to taking advantage of common software engineering practices like unit testing to ensure correct implementation, the path ahead includes leveraging formal methods to build provably correct networks as well as new SDN approaches for systematically troubleshooting [13].

D. Policy

During our deployment at the WIX we have received approximately 50MB of data per day that does not match our route advertisements. This traffic varied widely, from non-IP traffic to unsolicited routing protocols. These findings highlight an ongoing issue with policy enforcement at IXPs. Through the use of BGP extensions or a new policy framework operating in a distributed routing fabric, the migration of traffic between networks could be more tightly controlled. Proper enforcement would restrict traffic placed onto the fabric to the endpoints that are advertising routes to the destination subnet. Peers could choose more granularly who to share traffic with, and any traffic that did not meet the policy specifications could be dropped at the edge. All these features can be implemented by adding more granular, higher-priority OpenFlow rules that override the basic behaviour.

As shown in Fig. 5, the IXPs represent an interesting scenario for SDN to coalesce policies and requirements from different players and enable innovative services. Initial works on software-defined IXPs [8] suggest a number of applications that are simply not possible in today's routing infrastructure (e.g., domain-based or application-specific peering, remote control peering, enforceable inter-domain policies, origin-specific routing). In addition, the combination of SDN with IXPs may ease a number of tedious tasks (e.g., time-of-day routing, dynamic traffic engineering for peering policy compliance, route preference based on external inputs) by programmatic coordination using high-level software control –as opposed to low-level scripts and indirect mechanisms.

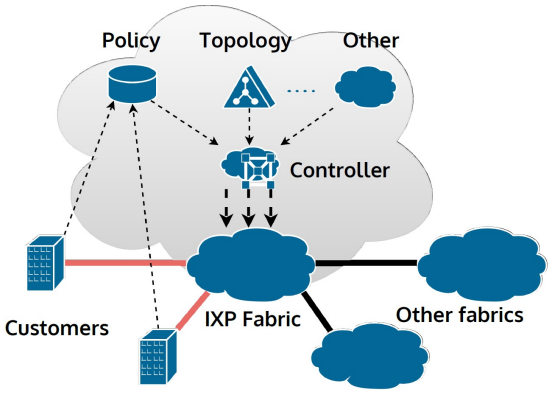

Fig. 5. Adding Policy control to an SDN-based IXP.

VI. Related Work

Several proposals have been made to implement a distributed router following a control/data plane split approach. However, many of them have been focused on software-based forwarding, proof of concept prototypes, or proprietary/closed-source implementations.

One early effort was the SoftRouter [14] architecture that separates the implementation of control plane functions from packet forwarding functions similar to the IETF ForCES framework [15]. This ongoing standardization work at IETF provides valuable insight into distributed router designs but no hardware implementations are known to be in production. Protocols such as Forz [16] allow for the propagation of flow information between distributed router components in a platform-agnostic manner. Distributed SW Router Project (DROP) [17] takes a step away from performance analysis of software routers towards scalability, including the identification of potential architectural bottlenecks applicable in more general contexts. Proprietary multi-chassis architectures have been developed which bear many similarities to a distributed router (e.g. [18, 19]). However, the lack of openness of the solutions hampers the potential of innovation in this area.

The widespread adoption of the OpenFlow protocol has opened up the possibilities for leveraging commodity hardware for SDN-based IP routing architectures similar in spirit to RouteFlow. For instance, Fibium [2] proposes an OpenFlow-based approach to routing where the routing engine is hosted on a physically separate PC. The switch controller listens for kernel route updates and converts these to OpenFlow rules. Through the use of smart route caching, the hybrid software/hardware approach of Fibium shows promise for the feasibility of separating control and forwarding even on datapaths with a small hardware-based flow table.

Recent efforts on seamless inter-working of SDN and IP [3] are undergoing to ease the transitioning of the widely deployed IP infrastructure to SDN. Control split architectures allow the investigation of new economic frameworks and outsourcing models applied to inter-domain IP routing [4]. In Cardigan, we share the same vision on SDN principles capable of introducing incremental enhancements while maintaining backwards compatibility. In the past, the power and benefits of centralized route control have been shown by research and deployment work of BGP-based Routing Control Platforms [20].

VII. CONCLUSION

We implemented Cardigan, a SDN-based distributed router, which was deployed in a live Internet exchange. This deployment acts as a poster-child for the concept of hybrid SDN-IP networks; Not only can SDN and legacy devices be deployed side-by-side, but value can be obtained through either replacing existing devices, or simply deploying new hardware when extending a network. The distributed router approach reduces the operational complexity of maintaining a network of routers through virtualization of the network.

We identified some limitations of our early designs and introduced a new approach to the router abstraction model, in addition to additional code extensibility and performance considerations. Despite various issues and limitations, Cardigan is successfully passing production traffic in a live internet exchange. We recognize that WIX is a modest Internet exchange, however, and anticipate further issues when attempting to scale to larger networks. We will consider appropriate monitoring of network resource usage and providing load-balancing; closest exit usage; and in particular, and more complex setups of distributed routers in non-mesh environments. All these topics are part of our ongoing work, including the investigation and implementation of new types of routing policies and addressing the issues discussed earlier such as performance, scalability or resilience.

VIII. ACKNOWLEDGEMENTS

The authors are thankful to REANNZ and CityLink NZ for their assistance in the deployment, in addition to Dylan Hall for his continued contributions in the project. Finally, we thank all fellows within the RouteFlow open source community project.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM CCR*, vol. 38, pp. 69–74, Mar. 2008.

[2] N. Sarrar, A. Feldmann, S. Uhlig, R. Sherwood, and X. Huang, "FIBIUM: Towards Hardware Accelerated Software Routers," Tech. Rep. 9, Deutsche Telekom Laboratories, Nov. 2010.

[3] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi, "Seamless interworking of sdn and ip," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, SIGCOMM '13, (New York, NY, USA), pp. 475–476, ACM, 2013.

[4] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: better internet routing based on sdn principles," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, (New York, NY, USA), pp. 55–60, ACM, 2012.

[5] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. 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, (New York, NY, USA), pp. 13–18, ACM, 2012.

[6] E. Keller and J. Rexford, "The "Platform as a Service" Model for Networking," in *INM/WREN '10*, Apr. 2010.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference*, (New York, NY, USA), pp. 3–14, ACM, 2013.

[8] N. Feamster, J. Rexford, S. Shenker, R. Clark, R. Hutchins, D. Levin, and J. Bailey, "SDX: A Software-Defined Internet Exchange." ONS '13 Research Track, April 2013.

[9] J. P. Stringer, Q. Fu, C. Lorier, R. Nelson, and C. E. Rothenberg, "Cardigan: Deploying a distributed routing fabric," in *Proc. of the HotSDN '13 (poster session)*, (New York, NY, USA), pp. 169–170, ACM, 2013.

[10] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in ip networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, pp. 307–319, June 2004.

[11] IEEE Standards Association, "Guidelines for use of the 24-bit Organisationally Unique Identifiers (OUI)." http://standards.ieee.org/develop/regauth/tut/eui.pdf.

[12] L. W. Yaoqing Liu, Syed Obaid Amin, "Efficient fib caching using minimal non-overlapping prefixes," *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 14–21, Jan. 2013.

[13] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, and P. Kazemian, "Leveraging sdn layering to systematically troubleshoot networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 37–42, ACM, 2013.

[14] T. V. Lakshman and et al., "The SoftRouter architecture," in *HotNets-III*, 2004.

[15] J. Halpern, E. Deleganes, and J. H. Salim, "Forces forwarding element model," *draft-ietf-forces-model-16 (work in progress)*, 2008.

[16] O. Hagsand, M. Hidell, and P. Sjödin, "Design and Implementation of a Distributed Router," in *ISSPIT '05*, Dec. 2005.

[17] R. Bolla and R. Bruschi, "An open-source platform for distributed Linux Software Routers," *Computer Communications*, vol. 36, pp. 396–410, Feb. 2013.

[18] I. Cisco Systems, "Cisco catalyst 6500 series virtual switching system," Dec. 2012.

[19] I. Juniper Networks, "Virtual chassis technology on ex8200 ethernet switch modular platforms," 2012.

[20] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA '04, (New York, NY, USA), pp. 5–12, ACM, 2004.