The RouteFlow approach to IP routing services on software-defined networks

Marcelo R. Nascimento[1], Christian E. Rothenberg[1], Marcos R. Salvador[1], Maurício F. Magalhães[2], Carlos N. A. Corrêa[3], Sidney C. de Lucena[3]

[1]Fundação CPqD – Centro de Pesquisa e Desenvolvimento em Telecomunicações
Campinas, SP – Brazil

[2]Faculdade de Enganharia Elétrica e Computação – Unicamp
Campinas, SP – Brazil

[3]Federal University of the Rio de Janeiro State – UniRio
Rio de Janeiro, RJ - Brazil

marcelon@cpqd.com.br, mauricio@dca.fee.unicamp.br, carlos.correa@uniriotec.br

1. Introduction

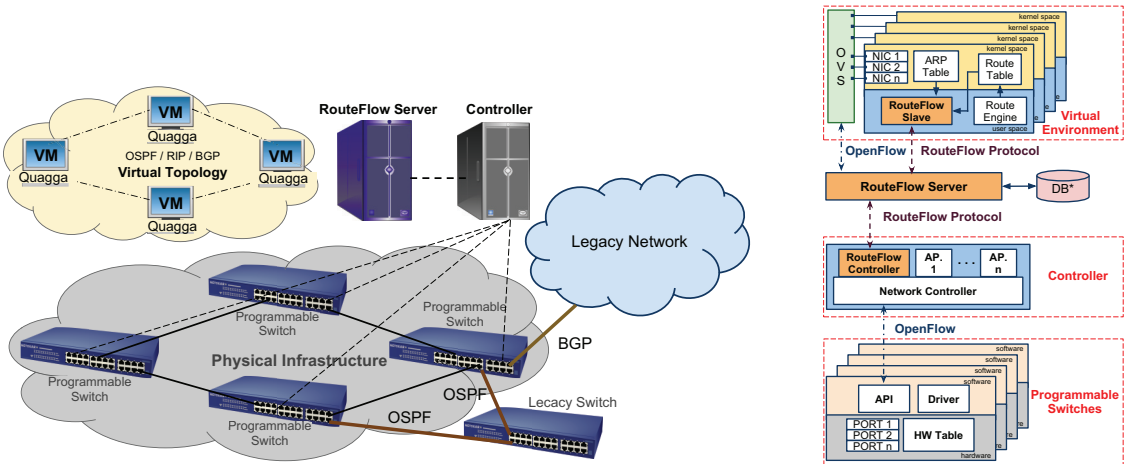
Besides the formidable evolution of the Internet with respect to its pervasiveness and applications, its core technology, mainly represented by the layered TCP/IP protocol suite, has not gone through an equally radical transformation. Since the Internet became commercial, network devices have been "black boxes"in the sense of vertically integrated implementations based on closed-source software over proprietary hardware [Hamilton]. This model not only leads to the already recognized Internet "ossification" but also implies higher R&D costs and slower time to market of new product features.

Recent developments in the standardization of vendor-neutral APIs (e.g., ForCES [Khosravi and Anderson 2003], OpenFlow [McKeown et al. 2008]) allow for "lobotomizing" a big part of the decision logic of network devices to external controllers implementable with commodity hardware (e.g. x86 server technology), a plentiful, scalable, and affordable resource.

RouteFlow, the work in progress depicted is this paper, is an architecture following the software-defined networking (SDN) [Greene 2009] paradigm based on a programmatic approach to logically centralize the network control, unify state information, and decouple forwarding logic and configuration from the hardware elements. It is composed by an OpenFlow controller application, an independent RouteFlow server, and a virtual network environment that interconnects traditional IP routing engines (e.g. Quagga).

The main goal of RouteFlow is enabling remote IP routing services in a centralized way, as a consequence of effectively decoupling the forwarding and control planes. This way, IP networks become more flexible and allow for the addition and customization of protocols and algorithms, paving the way for "router-as-a-service" models of networking in the virtual era. RouteFlow is the evolution of our early work on partnering Quagga with OpenFlow [Nascimento et al. 2010] and works transparent to the specific routing engine (e.g., XORP, BIRD) as long as it is based on the Linux networking stack.

The balance of this paper is as follows. Section 2 presents the design and modes of operation of the software-defined RouteFlow routing architecture. Section 3 describes the prototype implementation and Section 4 concludes the paper.



(a) Overview of a RouteFlow-controlled network. (b) RouteFlow components.

Figura 1. RouteFlow architecture conceptual design.

2. The RouteFlow design

RouteFlow runs OpenFlow switches' control logic through a virtual network composed by virtual machines (VMs), each of them executing a routing engine (see Fig. 1(a)). Those VMs (or virtual environments) are dynamically interconnected to form a logic topology that mirrors a physical infrastructure – or a simplified version of it. The virtual environment is held in (a set of) external servers and communicate with the forwarding plane through an OpenFlow controller application that receives from the RF server the decisions made by the routing protocols. As a result, flow rules are maintained in the data plane to specify how traffic must be handled (i.e. port forwarding, MAC re-writing, TTL decrement). While the control is centralized, it stays logically distributed. This way, it does not require modification of existing routing protocols. Moreover, legacy infrastructure can be transparently integrated, given that routing messages, like those of the BGP protocol, can be sent from/to the virtual control plane.

This approach leads to a flexible, high-performance and commercially competitive solution to provide IP routing based on: (a) programmable low cost switches and small-footprint embedded software (i.e. OpenFlow); (b) open-source routing protocols stacks (e.g. Quagga); and (c) commodity x86 server technology.

2.1. 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 2 shows the 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.

Multiplexing: This $1:n$ mapping of physical to virtual substrate represents the common approach to router virtualization where multiple control planes run simultane-

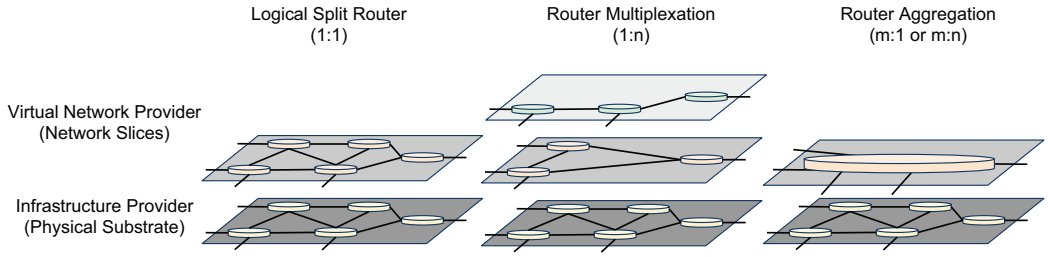


Figura 2. Different modes of operation and scenarios of router virtualization.

ously 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 traditional inter-domain or inter-zone routing (e.g. BGP) can be converged into single control unit for signaling scalability and simplified, centralized management purposes.

3. Prototype implementation

The current RouteFlow prototype is a combination of open-source software publicly available and newly-developed software daemons:[1]

The RouteFlow Controller and Server: The RF-Controller component is implemented as a C++ application running over NOX [Gude et al. 2008] named *routeflowc*. The RouteFlow network protocol is used to let *routeflowc* communicate with the RF-Server that interacts with the RF-Slave instances and the virtual switch. To support virtualization tool independence, Libvirt [Bolte et al. 2010] is the framework choosen to support the needed VM operations. Having the RF-Server as a standalone application facilitates the interaction with other (remote) OpenFlow controllers.

FIB gathering and rfslaved: Each VM in the virtual topology executes a RF-S daemon named *rfslaved* along with its routing engine (e.g. Quagga). *rfslaved* gathers FIB updates via the Netlink Linux API and sends the event data to the *routeflowc* via the RF-Server. Using Netlink to keep processes aware of connectivity changes renders *rfslaved* agnostic of the routing suites as long as it updates the OS FIB in response to the routing protocols execution.

Virtual topology networking: OpenVSwitch (OVS) [Pfaff et al. 2009] is the software switch controlled by the RF-Server and used to connect all NICs in a virtual topology that must be mutually reachable. OVS instances running in different hosts can be interconnected through a tunnel port. In addition to a distributed virtual topology and OpenFlow programmability support, our experimental evaluation shows better convergence times when using OVS.

[1] Available in the RouteFlow project page: https://sites.google.com/site/routeflow/

4. Conclusions

RouteFlow is an ongoing example of the power of innovation resulting from the blend of open interfaces to commercial hardware and open-source community-driven software development. The RouteFlow architecture allows for a flexible resource association between legacy routing control protocols and a programmable physical substrate, opening the door for multiple use cases around virtual routing services. However, a number of challenges have still to be overcome, and, to fully realize this vision, further work is required to see the PaaS model meet the networking world [Keller and Rexford 2010].

Referências

Bolte, M., Sievers, M., Birkenheuer, G., Niehörster, O., and Brinkmann, A. (2010). Non-intrusive virtualization management using libvirt. In *DATE '10*.

Greene, K. (2009). TR10: Software-Defined Networking. *MIT Technology Review*.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox: towards an operating system for networks. *SIGCOMM CCR.*, 38.

Hamilton, J. Networking: The last bastion of mainframe computing. http://perspectives.mvdirona.com/2009/12/19/ NetworkingTheLastBastionOfMainframeComputing.aspx.

Keller, E. and Rexford, J. (2010). The 'Platform as a Service' model for networking. In *INM/WREN 10*.

Khosravi, H. and Anderson, T. (2003). Requirements for separation of ip control and forwarding. RFC 3654.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *SIGCOMM CCR*, 38:69–74.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., and Magalhães, M. F. (2010). Quagflow: partnering quagga with openflow. *SIGCOMM CCR*, 40:441–442.

Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., and Shenker, S. (2009). Extending networking into the virtualization layer. In *HotNets-VIII*.