Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks

Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador
Telecomm. Research and Development Center (CPqD)
Campinas - SP - Brazil
esteve@cpqd.com.br

Carlos N. A. Corrêa, Sidney C. de Lucena
Federal University of the Rio de Janeiro State (UniRio)
Rio de Janeiro - RJ - Brazil
carlos.correa@uniriotec.br

Maurício F. Magalhães
University of Campinas (UNICAMP)
Campinas - SP - Brazil
mauricio@dca.fee.unicamp.br

ABSTRACT

The networking equipment market is being transformed by the need for greater openness and flexibility, not only for research purposes but also for in-house innovation by the equipment owners. In contrast to networking gear following the model of computer mainframes, where closed software runs on proprietary hardware, the software-defined networking approach effectively decouples the data from the control plane via an open API (i.e., OpenFlow protocol) that allows the (remote) control of packet forwarding engines. Motivated by this scenario, we propose RouteFlow, a commodity routing architecture that combines the line-rate performance of commercial hardware with the flexibility of open-source routing stacks (remotely) running on general purpose computers. The outcome is a novel point in the design space of commodity routing solutions with far-reaching implications towards virtual routers and IP networks as a service. This paper documents the progress achieved in the design and prototype implementation of our work and outlines our research agenda that calls for a community-driven approach.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet-switching networks

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 [11]. This model does not only lead to the recognized Internet "ossification" but also implies higher R&D costs and slower time to market of product features.

Recent standardization developments of vendor-neutral APIs (e.g., ForCES [13], OpenFlow [15]) allow for "lobotomizing" a big part of the decision logic of network devices to external controllers implementable with commodity hardware (e.g. x86 technology), a plentiful and scalable resource.

RouteFlow, the work in progress depicted is this paper, is an architecture following the software-defined networking (SDN) [9] 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 [5]. It is composed by an OpenFlow controller application and an independent RouteFlow server that manages a virtual network environment to interconnect virtualized IP routing engines (e.g. Quagga).

Routing protocol messages can be sent 'down' to the physical devices or can be kept in the virtual network plane, that may be a reproduction of the discovered physical infrastructure or a simplified / arbitrary mapping to hardware resources. The routing engines generate the forwarding information base (FIB) according to the configured routing protocols (e.g., OSPF, BGP). In turn, the IP and ARP tables are collected and translated into OpenFlow rules that are finally installed in the associated datapath devices.

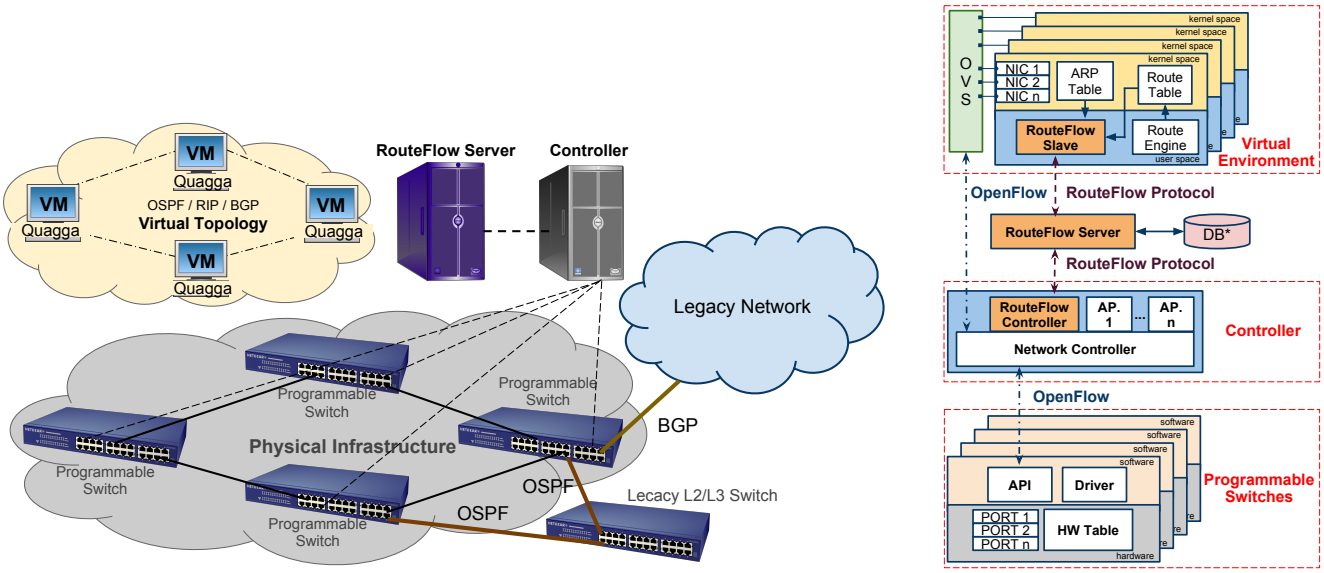
The main goal of RouteFlow is enabling remote IP routing services in a centralized way, as a consequence of effectively decoupling the forwarding and control planes. This way, IP networks become more flexible and allow for the addition and customization of protocols and algorithms, paving the way for virtual router [3] and IP network as a Service (IP-NaaS) [6] in the software-defined networking era. RouteFlow is the evolution of our early work on partnering Quagga with OpenFlow [16] and works transparently to any Linux-based routing engine (e.g., XORP, BIRD).

The balance of this paper is as follows. Section 2 presents the RouteFlow design along its different modes of operation and its main architectural components. Section 3 describes the prototype implementation. Section 4 discusses the research agenda and Section 5 concludes the paper.

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 physi-

CFI '11 June 13-15, 2011, Seoul, Korea



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

Figure 1: RouteFlow architecture conceptual design.

cal infrastructure – or a modified 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). 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 protocol messages (e.g. BGP, OSPF) can be sent from/to the virtual control plane.

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

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

Aggregation: This $m:1$ mapping of hardware resources to virtual instances allows to simplify the network protocol engineering by bundling a group of switches, such that neighbouring devices or domains can treat the aggregated as if it were a single element.[1] This way intra-domain routing can be independently defined while legacy inter-domain or inter-zone routing (e.g. BGP) can be consolidated into single control unit for signaling scalability and simplified, centralized management purposes (cf. [12]).

For every use case, two sub-modes of operation can be defined depending on whether the routing protocol messages are sent out through the physical ports or are kept in the virtual plane. The latter allows to separate and optimize the problem of physical topology discovery and maintainance and the problem of routing state distribution.

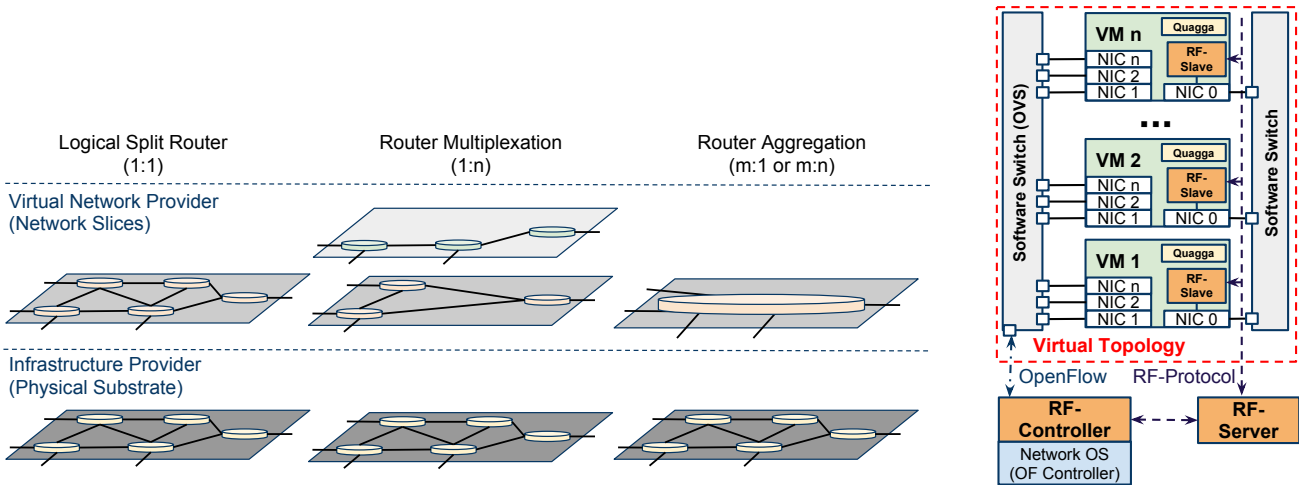
2.2 Architectural details

As shown in Fig. 1(b), the RouteFlow-Controller (RF-C) runs as an application on top of an OpenFlow network controller (NC). The NC is responsible for interfacing the OpenFlow-enabled switches, servicing the RF-C with the APIs, and discovering the network topology.

The core control logic resides in the RF-Server that is notified about relevant events and keeps the required network-wide state. For each OpenFlow switch found, the RF-Server instantiates one (or selects a pre-provisioned) VM.[2] Each VM runs a stack of open-source routing protocols and is configured with as many virtual network interfaces (NICs) as there are active ports in its corresponding device. The NICs are bound to software switches, through which their connectivity is dynamically controlled. Once the virtual topology is set up, the routing protocols in the VMs start running and adjust the FIBs accordingly. For each FIB update, the slave

[1]There is no conceptual barrier to support arbitrary $m:n$ mappings as pursued by IETF ForCES [13] that defines Control Elements (CE) and Forwarding Elements (CE), which compound form a Network Element (NE).

[2]This case corresponds to the logical split and multiplexing modes of operation. In case of aggregation, a single VM covers multiple physical switches which can be programmed to act as a single router (e.g. [3]).



(a) Modes of operation and usage scenarios of router virtualization.

(b) Virtual control plane.

Figure 2: Modes of operation of the virtualized networking environment

daemon (RF-S) sends an update message to the RF-Server that requests the installation of a flow entry matching the destination network mask with the corresponding actions for port-forwarding, MAC-rewriting, TTL-decrement, and IP header checksum update.

A simple RouteFlow protocol is defined for the interactions of the RF-Server with the RF-Controller and the RF-Slave instances. RF-protocol messages can be of the type *command* or *event* and can be seen as a simplified subset of OpenFlow protocol messages plus a number of new messages for VM/RF-Slave configuration and management purposes (e.g., accept/reject VM, RF-Slave config., send update).

To allow for legacy network integration, RouteFlow uses flow entries to match known routing protocol messages received from legacy devices in the physical infrastructure and pass them to the corresponding virtual entities. Conversely, pertinent routing messages originated in the virtual topology are sent through the physical infrastructure.

3. PROTOTYPE IMPLEMENTATION

The RouteFlow prototype is a combination of open-source software and newly-developed components:[3]

RF-Server and RF-Controller: The RF-Controller component is implemented as a C++ application (*routeflowc*) running on top of NOX [10]. The RF-Server is a standalone application responsible for the core logic of the system (e.g., event processing, mapping VMs to switches, resource management). Interactions between the RF-Server and RF-Controller are defined via RF-Protocol messages.

RF-Slave and FIB gathering: Each Linux VM in the virtual topology executes a RF-S daemon (*rfslaved*) along a routing engine (e.g. Quagga). *rfslaved* is a C++ standalone application that basically gathers FIB updates via the Netlink Linux API[4] and sends the event data via the RF-Protocol. In addition, the *rfslaved* executes a resource discovery technique for VM and switch-port identification.

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

[4]Netlink renders *rfslaved* agnostic of the specific routing suite as long as it updates the Linux networking stack.

Table 1: *ICMP Response Times.*

Equipment	Slow Path [ms]		Fast Path [ms]	
	$T_{avg.}$	$T_{90\%}$	$T_{avg.}$	$T_{90\%}$
CISCO 3560-e Catalyst	5.46	7.75	0.100	0.130
Extreme x450-e	11.30	14.00	0.106	0.141
CPqD Enterprise	14.20	17.30	0.101	0.147
RouteFlow	116.00	138.00	0.082	0.119

Virtual networking environment: OpenVSwitch (OVS) is the software switch used to connect all VM NICs in a virtual topology according to the reachability goals determined by the chosen mode of operation. We use the OpenFlow protocol support of OVS to dynamically manage the inter-VM connectivity and to select which packets should be sent to the forwarding plane. Moreover, OVS allows distributing the virtual network environment by having multiple OVS instances interconnected through tunnel ports.

Evaluation: Experiments with the prototype implementation in our NetFPGA-based testbed has proved interoperability with traditional networking gear and revealed that the routing protocol convergence time is dominated by the protocol time-out configuration (e.g., 4 x HELLO in case of OSPF) and does not suffer from the longer path to the control plane. As shown in Table 1, RouteFlow introduces larger latency only for those packets that need to be handled in the slow-path as a result of lacking a FIB entry or processing by the OS networking / routing stack (e.g., ARP requests, PING, Quagga routing protocol messages).

4. THE ROUTEFLOW R&D AGENDA

We look forward to turning RouteFlow into an open-sourced community-driven framework to deliver novel virtualized IP routing services in OpenFlow networks. We believe that the combination of the line-rate performance of commercial networking hardware with the flexibility of open-source routing stacks arranged through modern cloud programming practices may cross the research arena and unveil new business models. To fully realize this vision, we have identified several areas requiring further research and development work:

Applying PaaS to networking: Similar to the rationale behind cloud computing, RouteFlow shares the vision that the PaaS model meeting the networking world could be a game-changer (cf. [12]). Towards this goal, one feature in our roadmap is advanced VM management. Implementing Libvirt [2] allows for VM control via an unified API for a myriad of virtualization tools (e.g., QEMU, LXC, VMware, OpenVZ) along enhanced functionality like live migration or load balancing of the virtual control plane. Further developments include a comprehensive GUI and management facilities similarly to service platforms that implement an IP-oriented IaaS paradigm [6].

Moving beyond state-of-the art router virtualization (i.e., 1:1 mapping between control and physical elements) towards more flexible resource mapping (e.g., 1:N, M:N) is a goal full of challenges. As argued by Keller and Rexford [12], enabling a Single Router Platform would allow customers to focus on their application/service while addressing the management burden of infrastructure owners.

Protocol Optimization: The RouteFlow architecture allows for a separation of concerns between topology maintainance and routing state distribution (cf. [20]). This enables optimizing the routing protocols through fast connectivity maintainance techniques in the data plane (e.g. BFD-like) while route state distribution such as OSPF LSAs is flooded only inside the virtual domain. The challenge now is reproducing in the virtual domain the physical failures [17]. Once detected by any means, link failures can be programmatically induced via the OVS or by directly hooking into the specific routing stack (e.g. Zebra DB).

Resiliency and Scalability: Advances in VM technologies are also fundamental to circumvent different failure scenarios and to scale up by physically distributing the components. We need some strategy like master-backup [17] or distributed master controllers [20] to offer resilience in case of failures of the RF components. One component of the envisioned solution is a distributed database that holds the essential Network Information Base (NIB) (cf. Onix [20]). Yet another relevant research topic includes SDN-enabled strategies to deal with datapath failures such as decoupling failure recovery from path computation [4].

Embrace related work and build a community: Last but not least, we recognize the importance of (1) learning from previous work pursuing similar goals of separating control software from routers (e.g., SoftRouter [14], 4D [8], RCP [7]), (2) applying technologies from operational distributed systems such as cloud data center applications (e.g., event-based systems and NoSQL data stores [20]), and (3) building a community to joins efforts towards similar goals.[5]

To cite a few planned actions, we intend to investigate the interplay options with FlowVisor, the Mantychore APIs [6], the FIB-saving techniques of Fibium [18], the advances in split router architectures [19], and the implications of blending optical and electrical networks by integrating OpenFlow with GMPLS [1]. By revisiting the technical approach and promised benefits of the SoftRouter architecture (i.e., reliability, scalability, security, flexibility) [14] we hope to contribute to answering one question around the OpenFlow model: *Can a RouteFlow-like architecture transform the data networking industry in the same way the SoftSwitch transformed the voice telecom industry?*

[5]OpenFlowHub is one effort in broadcasting open-sourced SDN technologies: http://www.openflowhub.org

5. CONCLUSIONS

RouteFlow is an example of the power of innovation resulting from the blend of open interfaces to commercial hardware and open-source software development. The RouteFlow architecture allows for a flexible resource association between IP routing protocols and a programmable physical substrate, opening the door for multiple use cases around virtualized IP routing services. We expect RouteFlow contributing to the migration path from traditional IP deployments to software-defined networks enabled by means of a community-driven open-source framework. This path is however not free of research and development challenges.

6. REFERENCES

[1] S. Azodolmolky and et al. Integrated OpenFlow - GMPLS Control Plane: An Overlay Model for Software Defined Packet over Optical Networks. ECOC'11, Sep 2011.

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

[3] Z. Bozakov. Architecture and Algorithms for Virtual Routers as a Service. IWQoS, June 2011.

[4] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker. Dynamic route recomputation considered harmful. *SIGCOMM CCR.*, 40:66–71, April 2010.

[5] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *PRESTO '10*, 2010.

[6] E. Grasa and et al. MANTICORE II: IP Network as a Service pilots at HEAnet, NORDUnet and RedIRIS. TERENA Networking Conference 2010, 2010.

[7] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In *FDNA '04*, 2004.

[8] A. Greenberg and et al. A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5):41–54, 2005.

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

[10] N. Gude and et al. NOX: towards an operating system for networks. *SIGCOMM CCR.*, 38, July 2008.

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

[12] E. Keller and J. Rexford. The 'Platform as a Service' model for networking. In *INM/WREN 10*, Apr. 2010.

[13] H. Khosravi and T. Anderson. Requirements for separation of ip control and forwarding. RFC 3654, Nov. 2003.

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

[15] 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*, 38:69–74, March 2008.

[16] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, and M. F. Magalhães. QuagFlow: partnering Quagga with OpenFlow. *SIGCOMM CCR*, 40:441–442, August 2010.

[17] R. Ramjee and et al. Separating control software from routers. In *COMSWARE'06*, 2006.

[18] N. Sarrar, A. Feldmann, S. Uhlig, R. Sherwood, and X. Huang. FIBIUM - Towards Hardware Accelerated Software Routers. In *EuroView 2010 (poster session)*, August 2010.

[19] SPARC. Split architecture carrier grade networks. http://www.fp7-sparc.eu/.

[20] T. Koponen and et al. Onix: A distributed control platform for large-scale production networks. In *OSDI '10*, Oct 2010.