QuagFlow: Partnering Quagga with OpenFlow

Marcelo Ribeiro Nascimento,* Christian Esteve Rothenberg,*†
Marcos Rogerio Salvador* and Maurício Ferreira Magalhães†
* Telecommunications Research and Development Center (CPqD) - Campinas - SP - Brazil
† University of Campinas (Unicamp) - Campinas - SP - Brazil
{marcelon,marcosrs}@cpqd.com.br, {chesteve, mauricio}@dca.fee.unicamp.br

ABSTRACT

Computing history has shown that open, multi-layer hardware and software stacks encourage innovation and bring costs down. Only recently this trend is meeting the networking world with the availability of entire open source networking stacks being closer than ever. Towards this goal, we are working on QuagFlow, a transparent interplay between the popular Quagga open source routing suite and the low level vendor-independent OpenFlow interface. QuagFlow is a distributed system implemented as a NOX controller application and a series of slave daemons running along the virtual machines hosting the Quagga routing instances.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Experimentation, Design

1. INTRODUCTION

Traditionally, implementing IP routing meant buying expensive, vertically integrated equipment from a reduced number of players and an equally expensive piece of hardware-dependent, closed source software in a way that resembles the mainframes business model [4]. The value proposition of open source foundations has lead to the emergence of fairly complete open source routing stack implementations such as Quagga [2], XORP [8], and BIRD [7] among others. Most of these run on standard PC hardware and have been successfully embedded in commercially available netgear. More recently, the OpenFlow initiative [6] has attracted attention as a potential de facto standard interface for updating network routing tables in a hardware-independent manner. The potential impacts of general purpose computer programmability with the line-rate performance of commercial hardware are far-reaching.

Towards such an open and multi-sourced layered stack we propose QuagFlow, a transparent combination of the popular and mature Quagga routing software suite [2], providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4, and OpenFlow-enabled hardware. While OpenFlow provides means to treat operational traffic with legacy protocols embedded control plane of switches, in this work we explore the feasibility of completely moving a legacy protocol stack (Quagga) to logically centralized controllers using the OpenFlow protocol (together with the config protocol currently under development) as the solely communication channel with the forwarding engines.

In the short term, we regard QuagFlow as an intermediate step towards programmable (software-defined) networks to guarantee interoperability with legacy networks, with the immediate benefits of a remote partnership between Quagga and OpenFlow including:
- Cheap network gear with minimal embedded software
- Avoid "centralized" re-writing of proven protocols
- Ensure interoperability with legacy network elements
- Power of innovation to stakeholders, in particular network operators and service providers

In the longer term however, we look forward to extend QuagFlow to a multi-tenant environment offering third parties a *routing service platform*, in the spirit of a Routing Control Platform [1] — where the users could get controlled access to manage Quagga instances on a virtual domain-wide or per element basis. Along this direction, we reckon that, similar to cloud computing, a QuagFlow-based network virtualization service should evolve beyond overlaying remote virtual routing instances on top of a shared physical infrastructure towards a more convenient 'Platform as a Service' model for networking as argued by Keller and Rexford [5]. This poster presents our first pragmatic steps towards a (remotely running) complete open-source routing stack.

2. SYSTEM OVERVIEW

One of the design objectives of our first QuagFlow prototype is a system without code changes to neither OpenFlow nor Quagga, in a way that both developments can evolve independently. Hence, QuagFlow is divided in a QuagFlow controller (QF-C) application, developed as a user component of NOX [3], and a series of QuagFlow slaves (QF-S), running as transparent daemons in virtual machines (VM) hosting the Quagga routing software. Together, the discovered OpenFlow-enabled switching topology is replicated in the virtual control plane environment by configuring the VM interfaces and their virtual connectivity to mimic the conditions as if they were running directly in the physical devices. The QF-S monitors Quagga route table changes and informs the QF-C. Routing protocol packets to external subnets are delivered over the corresponding physical switch interfaces and vice versa. Figure 1 presents a high-level overview of a QuagFlow network and the system architecture.

Copyright is held by the author/owner(s).
SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
ACM 978-1-4503-0201-2/10/08.

QuagFlow: Partnering Quagga with OpenFlow

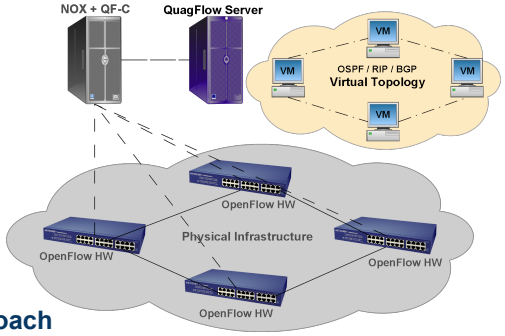
Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos R. Salvador and Maurício F. Magalhães
Telecommunications Research and Development Center (CPqD) - University of Campinas (Unicamp), Brazil

Goal

Seamless union of unmodified Quagga open source routing suite (OSPFv2-v3, RIPv1-v2, RIPng and BGP-4) with OpenFlow networks

Benefits

- Cheap network gear with minimal embedded software
- Provide interoperability with legacy network elements
- Avoid re-writing legacy protocols in a centralized fashion
- Innovation power to stakeholders (network operators, service providers)

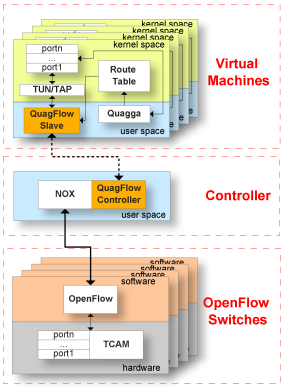


Approach

- QuagFlow (QF) is implemented as a NOX controller application (QF-C) and one slave daemon (QF-S) per VM running Quagga.
- QF replicates the physical topology by configuring and "stitching" the VMs into a virtual topology running the Quagga control plane
- QF sets the OpenFlow tables according to Quagga FIB updates and dispatches routing control messages to from/to legacy subnets

QuagFlow Architecture

- QuagFlow transparently arbitrates between Quagga & OpenFlow switches.



QuagFlow-Slave:

- Create TAP interfaces to represent the switch's ports.
- Send/receive packets to/from Quagga through TAPs.
- Monitor route table for changes and translate to flows.

QuagFlow-Controller:

- Registers to PacketIn, DatapathJoin and DatapathLeave events.
- Manages VM connectivity and QF-S.
- Forwards routing protocol packets from switches to QF-S and vice versa.
- Installs flow entries received from the QF-S in Openflow switches.

Preliminary results

- Control plane has worked successfully between OpenFlow and legacy switches.
- Routes from Quagga FIB are properly converted into flow entries.
- Data plane packets are correctly forwarded

Open issues & outlook

- Replicate data plane events in the virtual environment
- Extensive evaluation on realistic networking conditions
- OpenFlow table abstractions
- Towards a routing control PaaS

Figure 1: QuagFlow' poster preview.

3. PRELIMINARY RESULTS AND FUTURE WORK

In the prototype implementation, we were able to verify the correct interaction between a QuagFlow subnet and legacy networks running unmodified routing protocols. Route changes in the Quagga-generated FIB tables are properly detected by the QF-S daemons, converted into flow entries, and pushed in turn to the OpenFlow switches. Conversely, routing control packets from legacy network elements are correctly passed from OpenFlow switches to the corresponding Quagga appliance. Routing control messages among QuagFlow nodes are kept inside the high-performance server running all VMs. This way, the complete QuagFlow network control plane is contained within a safe, low-delay, resource-rich computing environment, in which we can leverage virtualization techniques to, e.g., take VM snapshots and easily migrate or roll back to stable configurations.

QuagFlow is however far from being a complete solution nor ready for an operational deployment (yet). There are a number of issues that still deserve careful considerations and extensive practical evaluation, including the replication of data plane events (e.g., link failures) in the virtual environment, optimizing protocol timers and the QF-S polling mechanisms, the VM managing framework, and so on. Certainly, more challenges will arise as we scale the system to real networking conditions. Along this journey, we expect to contribute to the evolution of the OpenFlow ecosystem (e.g., flow table abstraction, config protocol, FlowVisor).

This exploratory work is not limited to the feasibility of progressively putting OpenFlow networks in operation but also aims at devising opportunities in offering "virtual routing services," starting by enabling network operators/users to conveniently manage Quagga instances to customize the routing on and from/to their allocated network slices.

All in all, QuagFlow brings line-rate forwarding to open-source routing in an example of work towards a networking model of rapid innovation and rich network control at a fraction of the cost.

4. REFERENCES

[1] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *NSDI'05*, May 2005.

[2] GNU Quagga Project. http://www.quagga.org.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.

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

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

[6] OpenFlow Switch Consortium. Official website. http://www.openflowswitch.org.

[7] The BIRD (BIRD Internet Routing Daemon) Project. http://bird.network.cz.

[8] The XORP (eXtensible Open Router Platform) Project. http://www.xorp.org.