RouteFlow

Virtualized IP Routing Services in OpenFlow networks
Agenda

• Background: OpenFlow, Logical/Virtual Routers, Network Virtualization
• Project Overview
• Motivation
• Architecture
  – Controller
  – Server
  – Slave
  – Protocol
• Evaluation
• Work ahead
• Demo and hands-on Tutorial
RouteFlow is an open-source project to provide IP routing & forwarding services in OpenFlow networks.
About CPqD

• Major telecom R&D center in LATAM with expertise in various areas:
  – Optical (WDM, PON), Wireless (WiMax, LTE), IP (IMS/NGN, OpenFlow), OSS/BSS, Digital TV...
  – Today with ~1200 highly-skilled employees

• Created in 1976 as R&D branch of Telebras - Brazilian telecom monopoly

• Private foundation since 1998 after Telebras was privatized

• Purpose to foster innovation to help (mainly) Brazilian companies and society
  – Focus on technology R&D
  – Bridge the gap between universities and the industry

• Near highly-ranked universities in Brazil
  – History of collaborations
About the GIGA Project Testbed

800km total fiber span over 7 cities in 2 states (SP, RJ)
66 labs from 26 institutions connected (fiber to the lab) at 1 and 10 Gbps
Manually provisioned (VLAN) circuits for stable traffic

e2e dynamic (VLAN) multidomain protected circuits for L2 and above on demand experiments
Manually provisioned wavelengths for L1 and above experimentation
Focus on technology R&D and the industry

www.projetogiga.org.br
RouteFlow Project Timeline

Jan / 2010
• Start Msc. Thesis work by Marcelo N.
  • First Prototype
    • First Short-Paper @ WPEIF
  • QuagFlow Poster @ SIGCOMM

Aug / 2010
• Open-Source Release
• Evalulation on NetFPGA testbed

Dez / 2010
• Demos @ ONS11
• Indiana University
  • Pronto OF switches + BGP peering with Juniper MX

May / 2011
• Tutorial & Demo @ OFELIA/CHANGE SS
• Demo @ SuperComputing 11
  • Nation-wide field trials (Brazil, Internet2) @ ONS12

Oct / 2011
• RouteFlow + OpenStack (Cloud RouteFlow)

Nov / 2011
... / 2012
... building a community

1,390 visits came from 333 cities

189 days since Project Launch

http://go.cpqd.com.br/routeflow/
Apache-licensed code @ github
Overview

Virtual Topology

RouteFlow Server

Controller

Legacy Network

Physical Infrastructure

OSPF

BGP

Lecacy L2/L3 Switch
Motivation v1

Original motivation around RouteFlow (formerly QuagFlow) (Seeded in experience building a Broadcom-based L2/L3 switch prototype)

• Current “mainframe” model of networking equipment:
  • Costly systems based on proprietary HW and closed SW;
  • Lack of programmability limits customization and in-house innovation;
  • Ossified architectures.

• Goal: Open commodity routing solutions:
  + open-source routing protocol stacks (e.g. Quagga)
  + commercial networking HW with open API (i.e. OpenFlow)
  = line-rate performance, cost-efficiency, and flexibility!
Current router architectures

Management

Telnet, SSH, Web, SNMP

Control Logic

- RIP
- BGP
- OSPF
- ISIS

Driver

O.S.

Proprietary IPC / API

Hardware
OpenFlow model

Control Logic

Management

- RIP
- BGP
- OSPF

O.S.

API

Controller

Switch

Driver

O.S.

API

Hardware

Standard API (i.e. OpenFlow)

RouteFlow
Motivation v2

• A transition path, incrementally deployable: from current IP networks to SDN
  • Hybrid modes of operations: traditional IP control planes along SDN

• Innovation around IP control planes
  • Simplified network mgm, protocol optimization, shadow networks

• Advancing IP Network Virtualization
  • From flexible Virtual Routers to IP Network-as-a-Service
Use Cases

Logical Split Router (1:1)

Router Multiplexation (1:n)

Router Aggregation (m:1 or m:n)

Virtual Network Provider (Network Slices)

Infrastructure Provider (Physical Substrate)
### Key Features

- Separation of data and control planes;
- Loosely coupled architecture:
  - Unmodified routing protocol stacks;
    - Routing protocol messages can be sent 'down' or kept in the virtual environment;
- Portable to multiple controllers:
  - RF-Controller acts as a “proxy” app.
- Multi-virtualization technologies
- Multi-vendor data plane hardware
RF-Controller application

- Shim application on an OpenFlow controller
- Mainly acts like a proxy for the OpenFlow API
- Interacts with the OpenFlow datapaths
- Filters relevant events to the RF-Server
- Receives flow mod commands
- Delivers traffic to/from VM interfaces via OVS
RouteFlow Server

- The “brain” of RouteFlow;
- Manages available virtual machines (VM);
- Configures the virtual environment
- Receives events from the RF- controller
  - Switch join/leave, packet-in;
- Associates VMs and OpenFlow switches;
- Determines packet delivery from/to VMs
- Requests flow installation / modification in OpenFlow switches.
RF-Server: Association of VMs and DPs

<table>
<thead>
<tr>
<th>VM</th>
<th>DP</th>
<th>port</th>
<th>OVS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
RF-Server: Flow of Routing Control Packets

<table>
<thead>
<tr>
<th>VM</th>
<th>DP</th>
<th>port</th>
<th>OVS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

RouteFlow-Slave

- Runs as a daemon in Linux-based VM
- Registers the VM with the RF-Server
- Configures the VM (e.g., interfaces)
- Listens to ARP and IP table updates via Linux Netlink events
  - Linux Routing stack independent (Quagga, XORP)
- Translates routing updates into flow rules;
  - Match: DST_MAC + DST_IP + MASK
  - Actions: Re-write MACs + port-out
- Translates ARP entries into flow rules
  - Match: DST_MAC + DST_IP
  - Actions: Re-write MACs + port-out
- Sends flow update commands to RF-Server
- Runs VM-OVS attachment discovery protocol
RF-Slave: VM configuration

- Configure the amount of interfaces (enable/disable);
- Start/Stop Routing Engine;
- Clean interface configuration and ARP/ROUTE tables
RF Add/Remove Routes

<table>
<thead>
<tr>
<th>VM</th>
<th>DP</th>
<th>port</th>
<th>OVS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

Switch OpenFlow [1]

Switch OpenFlow [2]

Switch OpenFlow [n]

NOX

RF - Controller

VM A

VM B

VM C

RF - Server
IP Forwarding Rules in OpenFlow

RF-Slave info from the Linux network stack
- Route = IP + MASK [Rede]+IP[Gateway]+Interface
- ARP= IP[Host]+MAC[Host]+Interface

OpenFlow 1.0 entry:
- Match: DST_MAC + DST_IP + SUBNET_MASK
- Actions:
  - Re-Write [SRC_MAC (Interface)], Re-Write [DST_MAC (Nexthop)]
  - Forward [Port-out(Interface)]

Longest Prefix Match (LPM)
- Add priority to flow entry based on the length of the subnet mask

In OpenFlow 1.1:
- Additional actions: TTL decrement, checksum update
- Multiple-Table: Table[0] Matches DST_MAC, Table[1] Matches DST_IP
Virtual Environment

- V1 used TUN/TAP devices and payload encapsulation in the RF-Protocol
- V2 manages VM connectivity through an OpenFlow-capable soft-switch
- Routing engines (e.g. Quagga) exchange routing protocol packets
  - Two modes of operation for VM packet exchange:
    - UP: Directly through the OVS (requires Topology Disc)
    - DOWN: Through the physical switches
- Centralized but logically distributed
  - Can be physically distributed
- Support of different virtualization technologies
  - From QEMU to LXC
- VM-OVS Attachment Discovery Protocol
RF-Slave: Interface Attachment discovery (1)

- Discovery of VM interfaces attachment to OVS.
- Virtual interfaces are dynamically attached to the OVS
  - No guarantee of order
  - VMs may have an arbitrary number of interfaces
- When VM registers to the RF-Server the OVS ports in use are unknown.
RF-Slave: Interface Attachment discovery (2)

Discover the VM interfaces (ETH0)

RF-Slaves sends discovery frames to all ifaces except ETH0;

OVS forwards the packet-in to RF-Controller along the OVS port-in information.

RF-Server sets the mapping of VM-DP-Port-OVS_port.

<table>
<thead>
<tr>
<th>VM</th>
<th>DP</th>
<th>port</th>
<th>OVS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
The RouteFlow protocol

• Allowing a loosely couple architecture with two simple interfaces:
  • Protocol between RF-Server and RF-Slave
    • VM registration and configuration,
    • Generate OpenFlow rules:
      • Translate changes in IP and ARP tables into OF modification messages.
  • Protocol between RF-Server and RF-Controller
    • Basically, an API to controller OpenFlow stack
      • Subset of OpenFlow commands and events
      • Plus VM-OVS attachment discovery event
  • In short, an IPC/RPC mechanism
    – Application-level on top of TCP, Client-Server, Asynchronous, Without Confirmation
  Evolving to Apache Thrift & REST + JSON
## RF-Protocol: Frame

### RF Base Header
- srvId
- vmId
- type
- length

### RF VM Config Msg
- wildcards
- dpId
- mode
- nports

### RF IP Flow Config Msg
- netAddr
- netMask
- outPort
- srcMAC
- dstMAC
## API between RF-Controller and RF-Server

<table>
<thead>
<tr>
<th>Group</th>
<th>Type</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td>packet_in</td>
<td><code>datapath_id</code> (8 bytes) <code>port_in</code> (2 bytes) <code>pkt_id</code> (8 bytes) <code>type</code> (4 bytes)</td>
</tr>
<tr>
<td>EVENT</td>
<td>datapath_leave</td>
<td><code>datapath_id</code> (8 bytes)</td>
</tr>
<tr>
<td>EVENT</td>
<td>datapath_join</td>
<td><code>datapath_id</code> (8 bytes) <code>no_ports</code> (4 bytes) <code>hw_desc</code> (100 bytes)</td>
</tr>
<tr>
<td>EVENT</td>
<td>link_event</td>
<td><code>reason</code> (1 byte) <code>dp1</code> (8 bytes) <code>port_1</code> (2 bytes) <code>dp2</code> (8 bytes) <code>port_2</code> (2 bytes)</td>
</tr>
<tr>
<td>EVENT</td>
<td>map_event</td>
<td><code>VmId</code> (8 bytes) <code>VmPort</code> (2 bytes) <code>OvsPort</code> (2 bytes)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>flow</td>
<td><code>datapath_id</code> (8 bytes) <code>flow_mod</code> (2036 bytes)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>send_packet</td>
<td><code>datapath_id</code> (8 bytes) <code>port_out</code> (2 bytes) <code>pkt_id</code> (8 bytes)</td>
</tr>
</tbody>
</table>
VM Registration and Configuration

RF-Slave → RF-Server:
- *RFMessage (register)*
- *RFMessage (accept)*
- *RFVMMMsg (reset)*
- *RFVMMMsg (config)*
- *RFVMMMsg (reset)*

RF-Server:
- Check VmId
  - Datapath Join
  - Datapath Leave
Flow Modification messages

- New Route
  Translation: Route --> Flow

- New Host
  Translation: ARP --> Flow

- Route Removed
  Translation: Route --> Flow

- Host Expire
  ignored

- RFFlowMsg (flow_install)

- RFFlowMsg (flow_install)

- RFFlowMsg (flow_remove)
Agenda

• Background: OpenFlow, Logical/Virtual Routers, Network Virtualization
• Project Overview
• Motivation
• Architecture  
  – Controller
  – Server
  – Slave
  – Protocol
• Evaluation
• Work ahead
• Demo and hands-on Tutorial
NetFPGA-based testbed evaluation

RF-Server

NOX OpenFlow-Controller

5 x NetFPGA "Routers"
Prototype evaluation

• Setup
  • NOX controller
  • Quagga routing engine
  • 5 x NetFPGAs

• Results
  • Interoperability with traditional networking gear
  • Route convergence time is dominated by the protocol time-out configuration (e.g., 4 x HELLO in OSPF) not by slow-path operations
  • Larger latency only for those packets that need to go to the slow-path:
    • Lack FIB entry, need processing by the OS networking / routing stack e.g., ARP, PING, routing protocol messages.
Caveat: Lab-scale conditions!

- Low-latency links to RF-Controller
- No cross-traffic
- No CPU competition in OF switches
- Small FIBs, few topology changes
Scaling the Virtual Environment
Evaluation of the Virtualization Environment

Fig. 3. Grid topologies’ CPU use and link events convergence details
(a) CPU utilization
(b) Convergence after connectivity modification events

Fig. 4. Initial OSPF convergence for both connectivity layouts evaluated
(a) Initial convergence on grid topologies
(b) Initial convergence on full-meshed topologies

(a) Convergence after disconnection of a node
(b) Convergence after reconnection of node
The Path Ahead

- OpenFlow 1.1
- Controller API: Rest-API JSON & Apache Thrift
- Advancing the IP Network Virtualization
  - Protocol Optimization, Modes of operation, Router Migration
- Scalability and Resiliency
- System Limits and Stress testing
- Live Trials
  - Reality-Checks at Scale
- Embrace related work (past & ongoing)
  - SoftRouter, VROOM, DROP, FIBIUM, ONIX, etc.
- Build a community!
  - Student Projects corner (https://sites.google.com/site/routeflow/projects)
Protocol Optimization

• Separation of concerns between topology maintenance and routing state distribution
  – E.g. HELLOs sent “down” while LSA are kept “up”
  – E.g. BFD-like fault detection substitute HELLOs
Resiliency and Scalability

- Distributed Virtual environment with distributed OVS for load balancing, replication, and advanced VM management (e.g., migration)
- NoSQL-like distributed database for core RouteFlow state
- Multi-controller environments
- Fault-tolerance: Master / Slave, Master / Master, ...?
System limits and Stress testing

- Increase network size
  - Increase flowmod/sec
    - Variable OpenFlow control packet handling / processing:
      Impact on Routing Protocol?
      Impact on topology maintainance protocol, e.g., LLDP-based?

- Scale limitation (Flow table size) of logical / large routing tables
  - Smart shared multiple table lookup in OF.1.1
  - Smart caching, hybrid software-hardware flow state
  - Related Work (e.g., ViAggre)
  - etc.
Advancing the Use Cases and Modes of Operation

• From logical routers to flexible virtual networks
Aggregated Router

• Scenarios:
  – a single BGP router aggregating a number of OpenFlow switches
  – L3 services in data center distributed single virtual switch

• Distributed lookup?
  – E.g., Smart FIB generation and distribution

• Intra-router switching strategy?
NaaS - Network-as-a-Service
NaaS - Network-as-a-Service

• Enabling Virtual networks as a Service
• Many open research questions and ongoing work (e.g., Quantum @ OpenStack)
• CloudRouteFlow as a Service?
CPqD Dynamic Converged (Packet and Circuits) Services

**Goal:** Common control plane for Layers 1 to 3 networks aiming at NaaS, RaaS, VNO

**Approach:** OpenFlow + RouteFlow + SPIDER (virtualization comes in a subsequent phase)
FIBRE: FI testbeds between BRazil and Europe

- Joint EU-Brazil project between 9 partners from Brazil (6 from GIGA), 5 from Europe (4 from Ofelia and OneLab) and 1 from Australia (from OneLab)
  - Design, implementation and validation of a shared **Future Internet sliceable/programmable research facility**, supporting the joint experimentation of European and Brazilian researchers.

- The objectives include:
  - the development and operation of a new experimental facility in Brazil
  - the development and operation of a FI facility in Europe based on enhancements and the federation of the existing OFELIA and OneLab infrastructures
  - The federation of the Brazilian and European experimental facilities, to support the provisioning of slices using resources from both testbeds

- Officially started on Oct 1 2011
- Duration: 36 months
FIBRE site in Brazil

FIBRE Common Resources
- OF-enabled Switch
- Compute Servers
- NetFPGA Servers
- Orbit Nodes

Site-Specific Resources
- Wireless Testbeds
  - Wi-fi APs
  - Wimax
- Optical Testbeds
- Other Internal Testbeds
  (e.g. Emulab)

To Fibre Partners
- RNP Ipé
- GIGA
- Kyatera

RouteFlow
OFELIA-enabled Experiments

- **One** RouteFlow platform running in each OpenFlow island controlling only the OpenFlow switches in the **same facility**.

- **Experiment outputs**: Platform behaviour in geo-distributed setup, route convergence times, interoperability tests
OFELIA-enabled Experiments

• **Only one** RouteFlow platform running in a **single facility at a time** and controlling OpenFlow switches from every facility.

• **Experiment outputs:** Protocol behaviour under remote operation, route convergence times, slow-path performance
## Reality check at Euro-scale

<table>
<thead>
<tr>
<th>Experimental work</th>
<th>Current</th>
<th>Ofelie-enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>5 to 10 x 4-port NetFPGAs</td>
<td>10s of OpenFlow switches</td>
</tr>
<tr>
<td>Equipment</td>
<td>Software-based switches, NetFPGAs</td>
<td>Multi-vendor commercial OpenFlow switches</td>
</tr>
<tr>
<td>Realism</td>
<td>Few, small topologies Synthetic traffic (control + user) and failures</td>
<td>Geo-distributed topologies Real traffic (control &amp; data) ? and failure scenarios</td>
</tr>
<tr>
<td>Performance Fidelity</td>
<td>Low latency LAN</td>
<td>Variable network conditions</td>
</tr>
</tbody>
</table>
Tutorial 2

Traditional Scenario

```
 port1: 172.31.1.1 / 24
 port2: 10.0.0.1 / 24
 port3: 30.0.0.1 / 24
 port4: 50.0.0.1 / 24
```

```
 port1: 172.31.2.2 / 24
 port2: 10.0.0.2 / 24
 port3: 40.0.0.2 / 24
```

```
 port1: 172.31.3.3 / 24
 port2: 20.0.0.3 / 24
```

```
 port1: 172.31.4.4 / 24
 port2: 40.0.0.4 / 24
```

```
 eth0: 172.31.1.100 / 24
```

```
 eth0: 172.31.2.100 / 24
```

```
 eth0: 172.31.3.100 / 24
```

```
 eth0: 172.31.4.100 / 24
```
Conclusions

- RouteFlow proposes a commodity routing architecture that combines the line-rate performance of commercial hardware with the flexibility of open-source routing stacks (remotely) running on PCs;

- Allows for a flexible resource association between IP routing protocols and a programmable physical substrate:
  - Multiple use cases around virtualized IP routing services.
  - IP routing protocol optimization
  - Migration path from traditional IP deployments to software-defined networks
Thank you!

Ask and contribute!
routeflow-discuss@googlegroups.com

Learn more!
https://go.cpqd.com.br/routeflow

Get the Code!
https://github.com/CPqD/RouteFlow
Christian Esteve Rothenberg
esteve@cpqd.com.br
+55 19 3705-4479

www.cpqd.com.br
BACKUP
Proposed experiments

Mainly two types of experiments:

1. One RouteFlow platform running in each OpenFlow island controlling only the OpenFlow switches in the same facility.

2. Only one RouteFlow platform running in a single facility at a time and controlling OpenFlow switches from every facility.

The resulting combination of scenarios will allow to validate the scalability and performance limits of the remote operation of the IP routing stacks provided by RouteFlow.
Interop Experiments & Realistic Router Virtualization

Normal L2/L3 Processing

Research VLAN 1
Research VLAN 2
Production VLANs

OpenFlow-Controller + FlowVisor

RF-Server
Expected results

• Technical viability: Exploring the scalability and performance limits
  • convergence times not penalized by remote routing protocol stacks.
  • suitable distribution of control plane entities and the physical counterparts.
  • real-world networking conditions (e.g., latencies, failures, traffic)

• Interoperability and generality of RouteFlow
  • different open-source routing protocol stacks (XORP and Quagga),
  • different virtualization technologies (e.g., LXC and QEMU)
  • different OpenFlow controllers (e.g., NOX and Beacon),
  • all inter-working with commercial OpenFlow-enabled switches and legacy networking equipment.

• Assessment of the OFELIA testbed facilities
  • E.g., Capabilities of the Expedient CMF, effective resource sharing, controller application deployment, etc.