OpenFlow and Onix

Bowei Xu
boweixu@umich.edu


The Problem

- Experimenters’ dream
- Vendor’s Nightmare
  - Complexity of support
  - Market protection and barrier to entry
- Hard to build my own
  - Software only: Too slow
  - Hardware/software: fanout too small

How to run experiments in campus networks?

We also want

- Isolation:
  - Regular production traffic untouched
- Virtualized and programmable:
  - Different flows processed in different ways
- Open development environment for all researchers (e.g. Linux, Verilog, etc)
- Flexible definitions of a flow
  - Individual application traffic
  - Aggregated flows
  - Alternatives to IP running side-by-side
  - …
OpenFlow Switching

Flow Table Entry

• Type 0 OpenFlow Switch

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet + byte counters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

OpenFlow Usage Models

• Experiments at the flow level
  – User-defined routing protocols
  – Admission control
  – Network access control
  – Network management
  – Energy management
  – VOIP mobility and handoff
  – ...

• Experiments at the packet level
  – Slow: Controller handles packet processing
  – Fast: Redirect flows through programmable hardware
  – Modified routers, firewalls, NAT, congestion control...

• Alternatives to IP
Experiments at the Packet Level

Strengths

- A pragmatic compromise
  - Allow researchers to run experiments in their network ... without requiring vendors to expose internal workings.
- A simple basic idea
  - Exploit the fact that most modern Ethernet switches and routers contain flow-tables.
- Enabling innovation if widely accepted

Weaknesses

- Fixed size flow table header
  - Will increase the cost of searching in TCAM
- Assuming the basic processing unit is flow
  - Flow table may not be the most proper abstraction of primitive and workflow
- Membership of consortium is not open to companies

Onix: a Distributed Control Platform for Large-Scale Production Networks

The Problem

• Computer networks lack of a general control paradigm
• Each new function must provide its own state distribution, element discovery, and failure recovery mechanisms

Software-Defined Networking

• Network-wide control platform
• Handles state distribution
• Provide a programmatic interface
• Simplifies the duties of both switches and the control logic

Software-Defined Networking

• Most important challenges:
  – Generality
  – Scalability
  – Reliability
  – Simplicity
  – Control plane performance

Design of Onix

• Components
Design of Onix

• Useful and general API
  – Read
  – Write
  – Register for notifications
  – Customize the data model

Network Information Base (NIB)

• A graph of all network entities within a network topology
• Entity: Key-value pair with global identifier

Network Information Base (NIB)

• Functions provided by the Onix NIB API

<table>
<thead>
<tr>
<th>Category</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Find entities.</td>
</tr>
<tr>
<td>Create, destroy</td>
<td>Create and remove entities.</td>
</tr>
<tr>
<td>Access attributes</td>
<td>Inspect and modify entities.</td>
</tr>
<tr>
<td>Notifications</td>
<td>Receive updates about changes.</td>
</tr>
<tr>
<td>Synchronize</td>
<td>Wait for updates being exported to network elements and controllers.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Configure how state is imported to and exported from the NIB.</td>
</tr>
<tr>
<td>Pull</td>
<td>Ask for entities to be imported on-demand.</td>
</tr>
</tbody>
</table>

• The NIB neither provides fine-grained nor distributed locking mechanisms

Scalability

• Partition
  – An instance keeps only a subset of the NIB
• Aggregation
  – The network managed by a cluster of Onix nodes appears as a single node in a separate cluster’s NIB
• Consistency and durability
Reliability

- Network element and link failures
  - The same way as modern control planes
- Onix failures
  - Running instances detect and take over
  - More than one can manage simultaneously
- Connectivity infrastructure failures
  - Using standard networking gear
  - Reestablishing connectivity with the help of control logic

Distributing the NIB

- State Distribution Between Onix Instance
  - A transactional data store (for durability of the local storage)
  - A one-hop DHT (for holding volatile network state in a fast manner)

Distributing the NIB

- Network element state management
  - Similar to the integration with OpenFlow
- Consistency and Coordination
  - Application need to register inconsistency
  - Application must instruct the corresponding import and export modules to adjust Onix instances’ behavior
  - For coordination, embeds Zookeeper

Implementation

- 150,000 lines of C++ and third party libraries
- A single instance can run across multiple processes
- Language independent, components are loosely-coupled
Applications

<table>
<thead>
<tr>
<th>Control Logic</th>
<th>Flow Setup</th>
<th>Distribution</th>
<th>Availability</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethane</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed virtual switch</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Multi-tenant virtualized datacenter</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scale-out carrier-grade IP router</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Strengths

• Not about ideology of SDN, but about its implementation
• Deal with the scalability and reliability problem, provide more useful and general API

Weaknesses

• Relies on application-specific logic to detect and provide conflict resolution of the network state
• Still difficult to build control logic
• No pictures to illustrate

Thank you