SDN/OpenFlow

Dean Pemberton
Andy Linton
Agenda

• What is SDN and Openflow?
• Understanding Open vSwitch and RouteFlow
• Understanding RYU and SDN applications
• Simple SDN programming  
  – python vs IOS or Junos!
• Building a (distributed) switch
• Building a (distributed) router
Software defined networking (SDN)

- Separates control and data plane:
  - Open interface between control and data plane
  - Network control and management features in software
…SDN
Cardigan overview
Cardigan details
Lessons from history 😊

• "If you know what you're doing, 3 layers is enough; if you don't, 17 layers won't help you."

• [B]eware of the panacea peddlers: just because you wind up naked doesn't make you an emperor.
  – Michael A Padlipsky
3 Layers anyone?
Openflow overview

- One of the key technologies to realize SDN
- Open interface between control and data plane
Why Open vSwitch…

Open vSwitch's forwarding path (the in-kernel datapath) is designed to be amenable to "offloading" packet processing to hardware chipsets, whether housed in a classic hardware switch chassis or in an end-host NIC.

This allows for the Open vSwitch control path to be able to both control a pure software implementation or a hardware switch.
...Why Open vSwitch

The advantage of hardware integration is not only performance within virtualized environments. If physical switches also expose the Open vSwitch control abstractions, both bare-metal and virtualized hosting environments can be managed using the same mechanism for automated network control.
Components

- Control Cluster
  - ovsdb-server
  - ovs-vswitchd

- Off-box
- User
- Kernel
  - Management Protocol (6632/TCP)
  - OpenFlow (6633/TCP)
  - Netlink

OVS Kernel Module
Linux Bridge Design

- Simple forwarding
- Matches destination MAC address and forwards
- Packet never leaves kernel
Open vSwitch Design

- Decision about how to process packet made in userspace
- First packet of new flow goes to ovs-vswitchd, following packets hit cached entry in kernel
OVS Kernel Module

- Kernel module that handles switching and tunneling
- Fast cache of non-overlapping flows
- Designed to be fast and simple
  - Packet comes in, if found, associated actions executed and counters updated. Otherwise, sent to userspace
  - Does no flow expiration
  - Knows nothing of OpenFlow
- Implements tunnels
- Tools: ovs-dpctl
Userspace Processing

- Packet received from kernel
- Given to the classifier to look for matching flows
- Accumulates actions
- If “normal” action included, accumulates actions from “normal” processing, such as L2 forwarding and bonding
- Actions accumulated from configured modules, such as mirroring
- Prior to 1.11, an exact match flow is generated with the accumulated actions and pushed down to the kernel module (along with the packet)
Kernel Processing

- Packet arrives and header fields extracted
- Header fields are hashed and used as an index into a set of large hash tables
- If entry found, actions applied to packet and counters are updated
- If entry is not found, packet sent to userspace and miss counter incremented
Kernel Datapath

• ovs-dpctl speaks to kernel module
  See datapaths and their attached interfaces:
  – ovs-dpctl show
    See flows cached in datapath:
  – ovs-dpctl dump-flows
Agenda

• What is RouteFlow?
• RouteFlow Layout
• To the whiteboard Robin!
Traditional Scenario

```
<table>
<thead>
<tr>
<th>Device</th>
<th>Interface</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>eth0</td>
<td>172.31.1.100 / 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>172.31.0.24</td>
</tr>
<tr>
<td>h2</td>
<td>eth0</td>
<td>172.31.2.100 / 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>172.31.0.24</td>
</tr>
<tr>
<td>h3</td>
<td>eth0</td>
<td>172.31.3.100 / 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>172.31.0.24</td>
</tr>
<tr>
<td>h4</td>
<td>eth0</td>
<td>172.31.4.100 / 24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>172.31.0.24</td>
</tr>
</tbody>
</table>

```

Router A:
- port1: 172.31.1.1 / 24
- port2: 10.0.0.1 / 24
- port3: 30.0.0.1 / 24
- port4: 50.0.0.1 / 24

Router B:
- port1: 172.31.2.2 / 24
- port2: 10.0.0.2 / 24
- port3: 40.0.0.2 / 24

Router C:
- port1: 172.31.3.3 / 24
- port2: 20.0.0.3 / 24
- port3: 30.0.0.3 / 24

Router D:
- port1: 172.31.4.4 / 24
- port2: 40.0.0.4 / 24
- port3: 20.0.0.4 / 24
- port4: 50.0.0.4 / 24
What is Ryu?

- Name comes from a Japanese word meaning “flow”
- Ryu manages “flow” control to enable intelligent networking
Philosophy

• Agile
  – Framework for SDN application development instead of all-purpose big monolithic ‘controller’.

• Flexible
  – Vendor-defined “Northbound” APIs are not enough to differentiate.
Where does Ryu sit?
Ryu: Component-based framework

- Your application consists of component(s)
- Ryu provides a bunch of components useful for SDN applications.
- You can modify the existing components and implement your new components.
- Combines the components to build your application.
Components and libraries included in Ryu

- OpenStack Quantum
- Firewall
- OF REST
- Topology Viewer
- HA with Zookeeper
- L2 switch
- CLI
- Stats
- Snort
- VRRP
- Endpoint
- Topology
- OF-wire
- Netconf
- OF-conf
- OVSDB JSON
- sFlow
- NetFlow
Current Status…

- **OpenFlow protocol**
  - OF1.0 + nicira extensions, OF1.2, OF1.3, OF-Config 1.1
- **Other protocols**
  - netconf, vrrp, xFlow, snmp, ovsdb
- **Ryu applications/libraries**
  - Topology viewer
    - OF REST
    - Firewall
    - Some sample apps are in the ryu/app directory
Current Status

• Switch Interoperability
  – Referenced by some switch vendors
  – Open vSwitch
    • Integration testing with Open vSwitch (OF1.0, OF1.2) nicira extensions, OVSDB

• Integration with other components
  – HA with Zookeeper
  – IDS (Intrusion Detection System)
  – OpenStack Quantum
Restful interface available

OF REST API
  • add a flow entry
    POST http://example.org/stats/flowentry/add
  • delete flow entries
    DELETE http://example.org/stats/flowentry/delete
  • get flow stats
    GET http://example.org/stats/flow/{dpid}
Firewall

Firewall REST API
- add a rule
  POST http://example.org/firewall/rules/{switch-id}
- delete a rule
  DELETE http://example.org/firewall/rules/{switch-id}
- get rules
  GET http://example.org/firewall/rules/{switch-id}
Intrusion Detection System

Diagram:

1. L1~L4 matching
2. (2)
3. Alert
4. (3) Deep packet inspection

OpenFlow switch

Snot control app

Ryu

IDS(Snort)
L2 switch

Diagram:
- Host A
- Host B
- L2 switch
- Ryu
- FlowTable
- OpenFlow switch

1. (1) Host A to L2 switch
2. (2) Ryu to FlowTable
3. (3) Ryu to OpenFlow switch
4. (4) OpenFlow switch to Host B
class L2Switch(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    ofp_parser = dp.ofproto_parser
    in_port = msg.match['in_port']
...What does the code look like?

```python
actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
out = ofp_parser.OFPPacketOut(
    datapath=dp, buffer_id=msg.buffer_id, in_port=in_port,
    actions=actions)
dp.send_msg(out)
```

- So is this a hub or a switch?
- Should you use OFPPacketOut a lot?
So what’s missing?

• Mac address table
• Port up/down events
• VLANs
• LLDP
• ???
Python Performance?

• You need scalability probably
  – Language runtime efficiency can’t solve scalability problem
  – Scalability about the whole system architecture.

• Still need to improve runtime efficiency
  – Pypy: another python runtime using JIT.
  – Using C for such components.
Future work

• Make SDN development more agile
  – Adds more components (protocols, IaaS, stats, security, etc).
  – Introducing network abstraction model (hide southbound difference, etc).
  – Improves distributed deployment component (cluster support).
  – New testing methods (Ryu has more than 15,000 lines test code).
Links

- http://openvswitch.org/
- http://osrg.github.io/ryu/
- https://sites.google.com/site/routeflow/