SDN and OpenFlow

03/08/2019

Sumit Maheshwari

Includes material from Scott Shenker and Nick McKeown
SDN Recap

• Idea: An OS for the Network

• Key features:
  • Match-Action based rules
  • Flow based decisions
    • Traffic, packet, network..

• Advantages:
  • Simplified control
  • Open interface to packet forwarding
  • Well-defined APIs
Outline

• SDN Control Plane Problem
• OpenFlow basics
• SDN-OpenFlow demo using mininet
The Control Plane Problem

• Control plane must compute forwarding state. To accomplish its task, the control plane must:
  1. Figure out what network looks like (topology)
  2. Figure out how to accomplish goal on given topology
  3. Tell the switches what to do (configure forwarding state)

• We view this as a natural set of requirements....
  - And we require each new protocol to solve all three
Four Crucial Points

• SDN is merely set of abstractions for control plane
  - Not a specific set of mechanisms
  - OpenFlow is least interesting aspect of SDN, technically

• SDN involves computing a function…. 
  - NOS handles distribution of state

• …on an abstract network
  - Can ignore actual physical infrastructure

• Network virtualization is the “killer app”
  - Already virtualized compute, storage; network is next
SDN using OpenFlow
What is OpenFlow Protocol/API?

- SDN is not OpenFlow
  - OpenFlow is just one of many possible data plane forwarding abstraction (others: e.g., POF, P4, ...)
- OpenFlow is an open API that provides a standard interface for programming the data plane switches
OpenFlow Switch & OpenFlow API

- Controller (Server Software)
- OpenFlow Protocol/API
- Ethernet Switch
- Applications

Rutgers
WINLAB
OpenFlow Basics

Control Program A

Control Program B

Network OS

Packet Forwarding

Flow Table(s)

Packet Forwarding

"If header = p, send to port 4"
"If header = q, overwrite header with r, add header s, and send to ports 5,6"
"If header = ?, send to me"
More sophisticated flow identification

**Application level flow**
More sophisticated flow identification

**IP flow**
More sophisticated flow identification

**Custom flow**

![Diagram showing network flows and tables for MAC src, IP Src Prefix, and TCP dport]
More sophisticated flow identification

My flow
OpenFlow Switching

OpenFlow Client

OpenFlow Table

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port 1</td>
</tr>
</tbody>
</table>

Controller

server

Rutgers

WINLAB
OpenFlow Basics
Flow Table Entries

1. Forward packet to zero or more ports
2. Encapsulate and forward to controller
3. Send to normal processing pipeline
4. Modify Fields
5. Any extensions you add!

+ mask what fields to match
OpenFlow Examples

### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>00:1f...</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Routing

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
</tr>
</tbody>
</table>
Alice’s code:
- Simple learning switch
- Per Flow switching
- Network access control/firewall
- Static “VLANs”
- Her own new routing protocol: unicast, multicast, multipath
- Home network manager
- Packet processor (in controller)
- IPvAlice
OpenFlow Protocol/API: Some Specifics

- OpenFlow Switch Components
- OpenFlow Ports
- OpenFlow Tables
  - Match-Action Semantics
  - OpenFlow Pipeline Processing
  - OpenFlow Group Tables
- OpenFlow Actions
Ports

- Network interfaces for passing packets between Openflow Processing and the rest of the Network
- Openflow switches connected through Openflow ports

Types:
- Physical Ports
  - Switch defined ports correspond to a hardware interface (e.g., map one-to-one to the Ethernet interfaces)
- Logical Ports
  - Switch defined ports that do not correspond to a hardware switch interface (e.g. Tunnel-ID)
- Reserved Ports
  - Defined by ONF 1.4.0, specify generic forwarding actions such as sending to the controller, flooding and forwarding using non-openflow methods, such as normal switch processing
Ports - Reserved Port Types (Required)

- **ALL**
  - represents all ports the switch can use for forwarding a specific packets
  - Can be used only as output interface

- **CONTROLLER**
  - Represents the control channel with the OpenFlow controller
  - Can be used as an ingress port or as an output port

- **TABLE**
  - Represents the start of the OpenFlow pipeline
  - Submits the packet to the first flow table

- **IN_PORT**
  - Represents the packet ingress port
  - Can be used only as an output port

- **ANY**
  - Special value used in some OpenFlow commands when no port is specified
  - Can neither be used as an ingress port nor as an output port
Ports - Reserved Port Types (Optional)

- **LOCAL**
  - Represents the switch’s local networking stack and its management stack
  - Can be used as an ingress port or as an output port

- **NORMAL**
  - Represents the traditional non-OpenFlow pipeline of the switch
  - Can be used only as an output port and processes the packet using the normal pipeline

- **FLOOD**
  - Represents flooding using the normal pipeline
  - Can be used only as an output port
  - Send the packet out on all ports except the incoming port and the ports that are in blocked state
OpenFlow Table

- Every OpenFlow switch contains multiple flow tables
- Each flow table contains multiple flow entries
- The OpenFlow pipeline processing defines how packets interact with these flow tables
- **An OpenFlow switch is required to have at least one flow table**
- The flow tables of an OpenFlow switch are sequentially numbered, starting at 0

**Pipeline Processing**
- Start at the first flow table
- Other flow tables may be used depending on the outcome of the match in the first table
- Go only in forward direction not backward
- If packet is not redirected to another flow table, then pipeline processing stops and the packet is processed with the associated action set
Pipeline Processing

(a) Packets are matched against multiple tables in the pipeline

1. Find highest-priority matching flow entry
   - i. Modify packet & update match fields (apply actions instruction)
   - ii. Update action set (clear actions and/or write actions instructions)
   - iii. Update metadata

2. Apply instructions:

3. Send match data and action set to next table

(b) Per-table packet processing
Flow Entry

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Priority</th>
<th>Counters</th>
<th>Instructions</th>
<th>Timeouts</th>
<th>Cookie</th>
</tr>
</thead>
</table>

- **Match fields**: consist of the ingress port, packet headers, and optionally metadata specified by a previous table
- **Priority**: matching precedence of the flow entry
- **Counters**: updated when packets are matched
- **Instructions**: to modify the action set or pipeline processing
- **Timeouts**: maximum amount of time or idle time before flow is expired
- **Cookie**: used by the controller to filter flow statistics, flow modification and flow deletion. Not used when processing packets
Matching

Packet In
Start at table 0

Match in table n?
Yes

Update counters
Execute instructions:
- update action set
- update packet/match set fields
- update metadata

No

Goto-Table n?
Yes

No

Execute action set

Table-miss flow entry exists?
Yes

No

Drop packet
Table-Miss & Flow Removal

• Table-miss
  • Flow entry added by the controller
  • Priority is 0 (lowest)
  • Actions
    • Send packets to the controller
    • Drop packets
    • Direct packets to a subsequent table

• Flow Removal
  • Requested by the controller
  • Time expiry (hard timeout, idle timeout)
  • Optional switch eviction mechanism
Group Table

• Additional method for forwarding to a group of entries (select, all)

• Main components:
  • Group ID, Group Type, Counters, Action buckets (each action bucket contains a set of actions to be executed)
Group Table - Group Type

• **All**
  - Execute all buckets in a group
  - Used mainly for multicast and broadcast – fwd a pkt on all the ports

• **Select (optional)**
  - Execute one bucket in a Group
  - Implemented for load sharing and redundancy

• **Indirect**
  - Execute one defined bucket in this Group
  - Supports only a single bucket ( Eg. multiple routes are pointing to same next hop)

• **Fast failover (optional)**
  - Execute the first live bucket
  - Eg. There is a primary path and secondary path – pass the traffic on primary path and if it fails use the secondary one
Meter Table

- Consists of meter entries and defining per-flow meters

- Per-flow meters enable OF to implement QoS operations (rate-limiting), can be combined with per-port queues for complex QoS operations

- Meters measures the rate of packets assigned to it and enable controlling the rate of those packets

- Meters are attached directly to flow entries
Meter Table

• Components of Meter table:
  • Meter ID, Meter Band, Counters

• Meter band: unordered list of meter bands, where each meter band specifies the rate of the band and the way to process packet

• Components of Meter band:
  • Band Type, Rate, Counters, Type specific arguments
  • Band Type: defined how to process a packet (drop/ dscp remark)
Instructions

• Instructions are executed when a packet matches entry

• Instruction result: Change the packet, Action set, Pipeline processing

• Supported instruction Types:
  □ Meter ID
    □ Direct a packet to the meter id. It may be dropped because of metering.
  □ Apply-Actions
    □ Apply a specific action immediately here packets are modified between 2 flow tables
  □ Clear-Actions
    □ Clear all the actions in the action set immediately
  □ Write-Actions
    □ Add a new action into the existing action set. If same action exists then overwrite it.
  □ Write-Metadata
    □ Write the masked meta data value
  □ Goto-Table
    □ Indicate the next table in the processing pipeline
Action Set

- Action set is associated with each packet
- FE modify the action set using write-action/clear-action
- Actions in the action-set will be executed when pipeline is stopped
- Action set contains maximum of one action of each type
- If multiple actions of the same type need to be added then use “Apply-Actions”
- Need to follow the order below to execute action

Different Types of Action Set:
- Copy TTL inwards – apply copy inward actions to the packet
- Pop – apply all tag pop actions to the packet
- Push MPLS – apply MPLS tag push action to the packet
- Push PBB – apply PBB tag push action to the packet
- Push VLAN: apply VLAN tag push action to the packet
- Copy TTL outwards
- Decrement TTL
- Set – apply set field actions to the packet
- QoS
- Group – apply group actions
- Output – forward a packet on the port specified by the output action
Action List

- “Apply-action”, “packet-out” messages include action list
- Execute an action immediately
- Actions are executed sequentially in the order they have been specified
- If action list contains an output action, a copy of the packet is forwarded in its current state to the desired port
- Action-set shouldn’t be changed because of action-list
Actions

- What to do with the packet when match criteria matches with the packet

- Some of the Action Type:
  - Output
    - Fwd a pkt to the specified open flow port (physical/ logical/reserved)
  - Set Queue
    - Set Queue-id of the port: determines which queue should be used for scheduling and forwarding packet
  - Drop
    - Packets which doesn’t have output action should be dropped
  - Group
    - Process the packet through specified group
  - Push-Tag/ Pop-Tag
    - Insert VLAN, MPLS tag
  - Set-Field
    - Rewriting a field in the packet header
  - Change TTL
    - Decrement TTL
Data-Plane: Simple Packet Handling

- Simple packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller
Unifies Different Kinds of Boxes

- **Router**
  - Match: longest destination IP prefix
  - Action: forward out a link

- **Switch**
  - Match: dest MAC address
  - Action: forward or flood

- **Firewall**
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny

- **NAT**
  - Match: IP address and port
  - Action: rewrite addr and port
SDN Demo using OpenFlow and mininet
SDN Demo using OpenFlow

• Download mininet OVF:
  • https://github.com/mininet/mininet/releases/download/2.2.2/mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64.zip

• Start up VirtualBox, then select File>Import Appliance and select the .ovf image that you downloaded.

• Run mininet VM:
  • Username: mininet
  • Password: mininet

• Make this VM accessible from host machine as done previously with comnetsII VM.

• Controller: POX (Python)
  • git clone http://github.com/noxrepo/pox
OpenFlow Test: 3 Hosts and 1 Switch

Terminal 1

$ sudo mn --topo single,3 --mac --switch ovsk --controller remote

$ nodes
$ h1 ifconfig
$ xterm h1 h2

(ignore the host numbering and IPs in the figure)
Switch access using: ovs-ofctl command - Manually

**Dump Flows to check rules:**

$ sudo ovs-ofctl dump-flows s1

**Ping h1 to h2**

$ h1 ping -c3 h3

-- Did not ping. Why?

**Install flow rules:**

$ ovs-ofctl add-flow s1 in_port=1,actions=output:3
$ ovs-ofctl add-flow s1 in_port=3,actions=output:1

**Check flow rules again:**

$ sudo ovs-ofctl dump-flows s1

Now ping again:

$ h1 ping -c3 h3

---

**Is everything connected?**

$ pingall
$ iperf
$ exit

$ sudo mn --link tc,bw=10,delay=10ms

**iperf**

*** Iperf: testing TCP bandwidth between h1 and h3

*** Results: ['24.3 Gbits/sec', '24.4 Gbits/sec']

---

Happiness is: when the Ping works! 😊
Now connect the Controller

Terminal 2

$ sudo mn –c

Other terminal:

$ cd pox

$ ./pox.py log.level --DEBUG misc.of_tutorial

Terminal 1

$ sudo mn --topo single,3 --mac --switch ovsk --controller remote (see that controller is connected)

$ xterm h1 h2 h3

Capture packets on h2 and h3:

$ tcpdump –XX –n –i h2-eth0 (on h2)

$ tcpdump –XX –n –i h3-eth0 (on h3)

From h1 ping

$ ping –c1 10.0.0.2

Also, $ ping –c1 10.0.0.5 (observe the tcpdump)

Is everything connected?

Terminal 1

$ pingall

$ iperf

Observe the bandwidth, why is it dropped?
Backup - Research Challenges
Heterogeneous Switches

Number of packet-handling rules
Range of matches and actions
Multi-stage pipeline of packet processing
Offload some control-plane functionality (?)
Controller Delay and Overhead

Controller is much slower than the switch
Processing packets leads to delay and overhead
Need to keep most packets in the “fast path”
Testing and Debugging

- OpenFlow makes programming possible
  - Network-wide view at controller
  - Direct control over data plane

- Plenty of room for bugs
  - Still a complex, distributed system

- Need for testing techniques
  - Controller applications
  - Controller and switches
  - Rules installed in the switches
Programming Abstractions

- Controller APIs are low-level
  - Thin veneer on the underlying hardware

- Need better languages
  - Composition of modules
  - Managing concurrency
  - Querying network state
  - Network-wide abstractions