Floodlight tutorial

Chen Liang
cliang@cs.duke.edu
What is Floodlight?

• an Open source SDN controller platform
  – Apache-licensed
  – OpenFlow protocol
  – Java based
  – Enterprise class controller
Floodlight overview
Basic functionality

- Topology discovery
  - LLDP protocol

- Flow installation/deletion
  - install/modify/delete a flow on a switch
    - flow is defined as all packets with the same match

- Stats query
  - packet counts
  - flow counts
  - port stats query
  - etc.
Basic functionality

• Topology discovery
  – LLDP protocol

• Flow installation/deletion
  – install/modify/delete a flow on a switch
    • flow is defined as all packets with the same match

• Stats query
  – packet counts
  – flow counts
  – port stats query
  – etc.
Flow installation: an example

FlowMod

Match: dst ip = 10.0.0.1

Action: output on port 5
Flow installation: Match

- A flow a set of packets that have the same value in certain fields
- all these fields combined compose a Match

- examples of Matches:
  - <src ip: 10.0.0.2, dst ip 10.0.0.3, src port: 90>
  - <src mac addr: 00:0a:95:9d:68:16>
  - <vlan tag: 4000, protocol: ipv4>
Flow installation: Match

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>When applicable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingress Port</td>
<td>(Implementation dependent)</td>
<td>All packets</td>
<td>Numerical representation of incoming port, starting at 1.</td>
</tr>
<tr>
<td>Ethernet source address</td>
<td>48</td>
<td>All packets on enabled ports</td>
<td></td>
</tr>
<tr>
<td>Ethernet destination address</td>
<td>48</td>
<td>All packets on enabled ports</td>
<td></td>
</tr>
<tr>
<td>Ethernet type</td>
<td>16</td>
<td>All packets on enabled ports</td>
<td>An OpenFlow switch is required to match the type in both standard Ethernet and 802.2 with a SNAP header and OUI of 0x000000. The special value of 0x05FF is used to match all 802.3 packets without SNAP headers.</td>
</tr>
<tr>
<td>VLAN id</td>
<td>12</td>
<td>All packets of Ethernet type 0x8100</td>
<td></td>
</tr>
<tr>
<td>VLAN priority</td>
<td>3</td>
<td>All packets of Ethernet type 0x8100</td>
<td>VLAN PCP field</td>
</tr>
<tr>
<td>IP source address</td>
<td>32</td>
<td>All IP and ARP packets</td>
<td>Can be subnet masked</td>
</tr>
<tr>
<td>IP destination address</td>
<td>32</td>
<td>All IP and ARP packets</td>
<td>Can be subnet masked</td>
</tr>
<tr>
<td>IP protocol</td>
<td>8</td>
<td>All IP and IP over Ethernet, ARP packets</td>
<td>Only the lower 8 bits of the ARP op-code are used</td>
</tr>
<tr>
<td>IP ToS bits</td>
<td>6</td>
<td>All IP packets</td>
<td>Specify as 8-bit value and place ToS in upper 6 bits.</td>
</tr>
<tr>
<td>Transport source port / ICMP Type</td>
<td>16</td>
<td>All TCP, UDP, and ICMP packets</td>
<td>Only lower 8 bits used for ICMP Type</td>
</tr>
<tr>
<td>Transport destination port / ICMP Code</td>
<td>16</td>
<td>All TCP, UDP, and ICMP packets</td>
<td>Only lower 8 bits used for ICMP Code</td>
</tr>
</tbody>
</table>
Background: Subnet masks

- specify a subnet (a subset of IP addresses):
  - For 192.168.5.130/24:

<table>
<thead>
<tr>
<th></th>
<th>Binary Form</th>
<th>Dot-decimal notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP address</td>
<td>11000000.10101000.00000101.10000010</td>
<td>192.168.5.130</td>
</tr>
<tr>
<td>Subnet mask</td>
<td>11111111.11111111.11111111.00000000</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>Network prefix</td>
<td>11000000.10101000.00000101.00000000</td>
<td>192.168.5.0</td>
</tr>
<tr>
<td>Host part</td>
<td>00000000.00000000.00000000.10000010</td>
<td>0.0.0.130</td>
</tr>
</tbody>
</table>

- For 192.168.5.130/26:

<table>
<thead>
<tr>
<th></th>
<th>Binary Form</th>
<th>Dot-decimal notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP address</td>
<td>11000000.10101000.00000101.10000010</td>
<td>192.168.5.130</td>
</tr>
<tr>
<td>Subnet mask</td>
<td>11111111.11111111.11111111.11000000</td>
<td>255.255.255.192</td>
</tr>
<tr>
<td>Network prefix</td>
<td>11000000.10101000.00000101.10000000</td>
<td>192.168.5.128</td>
</tr>
<tr>
<td>Host part</td>
<td>00000000.00000000.00000000.00000010</td>
<td>0.0.0.2</td>
</tr>
</tbody>
</table>
Flow installation: Match

• In Floodlight, each match is an object of org.openflow.protocol.OFMatch

• i.e. to create a match for flow:
  – <src ip: 192.168.12.0/24, dst ip: 10.0.0.0/8>

```java
OFMatch match = new OFMatch()
match.setNetworkSource(IPv4.toIPv4Address("192.168.12.0"));
match.setNetworkDestination(IPv4.toIPv4Address("10.0.0.0"));
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
```
Flow installation: Match

• Make sure the wildcards is set correctly:
  – the following three are all different matches

```java
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(24));
match.setWildcards(Wildcards.FULL.matchOn(Flag.IN_PORT)
  .withNwSrcMask(24).withNwDstMask(24));
```

• An example: A match on the fields of in_port, src_ip (full match) and dst_ip (full match) shoud be set as

```java
match.setWildcards(Wildcards.FULL
  .matchOn(Flag.NW_DST)
  .matchOn(Flag.NW_SRC)
  .withNwDstMask(32)
  .withNwSrcMask(32)
  .matchOn(Flag.DL_TYPE));
```
Flow installation: Match

• For the same set of flows, matches on different switches can be different:

```
OFMatch match = new OFMatch()
match.setNetworkSource(IPv4.toIPv4Address("192.168.12.0"));
match.setNetworkDestination(IPv4.toIPv4Address("10.0.0.0"));
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
```

is different from:

```
OFMatch match = new OFMatch()
match.setNetworkSource(IPv4.toIPv4Address("192.168.12.0"));
match.setNetworkDestination(IPv4.toIPv4Address("10.0.0.0"));
match.setInputPort((short)2);
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
```
Flow installation: Action

• A set of operations associated with a match, for all packets with the same match, the operations will be applied

• examples of Actions:
  – <output on port 2>
  – <set dst IP address to 10.0.0.3>
  – <set mac address to 00:0a:95:9d:68:16>
Flow installation: Action

• In Floodlight, each actions is a object of org.openflow.protocol.OFAAction
  – org.openflow.protocol.action.OFAAction

• When there are multiple actions, output should always be the last one

• i.e.: create two actions to
  – first, modify mac address;
  – then, output packet to the specified port

```java
List<OFAAction> actions = new ArrayList<>(2);
OFAAction action1 = new OFActionDataLayerDestinationDestination(macaddr);
actions.add(action1);
OFAction action2 = new OFActionOutput(port, (short)0);
actions.add(action2);
```
Flow installation: FlowMod

• There are a number of different types of messages a controller can send to a switch, i.e.:
  – to query port stats: OFPortStatus
  – to query vendor: OFVendor
  – to modify status of a port: OFPortMod

• FlowMod is the message regarding flow installation/deletion
Flow installation: FlowMod

• In Floodlight, each FlowMod message is an object of OFFlowMod:
  – org.openflow.protocol.OFFlowMod

• To create an empty FlowMod message (for installing a flow)

```java
OFFlowMod flowMod = (OFFlowMod) floodlightProvider
    .getOFMessageFactory()
    .getMessage(OFType.FLOW_MOD);
flowMod.setCommand(OFFlowMod.OFPFC_ADD);
```
Putting together

• To install a flow
  – 1. create a FlowMod message
  – 2. specify the match of the flow in the message
  – 3. specify the actions for the flow
    • <output> in this case
  – 4. send the message to the switch
Putting together

• create the message, set match and actions

```java
OFFlowMod flowMod = (OFFlowMod) floodlightProvider
    .getOFMessageFactory()
    .getMessage(OFType.FLOW_MOD);

OFMatch match = ...
List<OFAction> actions = ...
flowMod.setCommand(OFFlowMod.OFPFC_ADD)
flowMod.setMatch(match);
flowMod.setActions(actions);
```

• send the message to the switch:

```java
IOFSwitch sw = this.floodlightProvider.getSwitch(swid);
sw.write(flowMod, null);
```
In dealing with IP packets

• Need to properly set datalayer type and netmask mask
  – Example: setup a flow matching on dst_ip=10.0.0.100 (no subnet)
  – optionally, you can further specify network layer protocol by further specifying:

```
match.setNetworkDestination(IPv4.toIPv4Address("10.0.0.100"));
match.setWildcards(Wildcards.FULL
  .matchOn(Flag.NW_DST)
  .withNwDstMask(32)
  .matchOn(Flag.DL_TYPE));
match.setDataLayerType(Ethernet.TYPE_IPV4);
```

```
match.setWildcards(match2host.getWildcardObj().matchOn(Flag.NW_PROTO));
match.setNetworkProtocol((byte)(IPv4.PROTOCOL_ICMP |
  IPv4.PROTOCOL_TCP | IPv4.PROTOCOL_UDP));
```
Optional Fields of FlowMod

- fields in FlowMod to specify optional properties for a flow, i.e.:
  - set idle timeout
    ```java
    flowMod.setIdleTimeout(idleTimeout);
    ```
  - set hard time out
    ```java
    flowMod.setHardTimeout(hardTimeout);
    ```
  - set priority
    ```java
    flowMod.setPriority(priority);
    ```
  - etc.
Flow deletion/modification

• Almost the same as adding a flow, except:
  – Changing
    flowMod.setCommand(OFFlowMod.OFPFC_ADD);
    • to
    flowMod.setCommand(OFFlowMod.OFPFC_DELETE);
    • or
    flowMod.setCommand(OFFlowMod.OFPFC_MODIFY);
  – And need to specify outport for deletion
    flowMod.setOutPort(
      (command == OFFlowMod.OFPFC_DELETE) ?
      outPort : OFPort.OFPP_NONE.getValue());
Basic functionality

• Topology discovery
  – LLDP protocol

• Flow installation/deletion
  – install/modify/delete a flow on a switch
    • flow is defined as all packets with the same match

• Stats query
  – packet counts
  – flow counts
  – port stats query
  – etc.
Statistics query

OFStatisticsRequest

specify entity (matchs/ports)

entity1: stat
group 2: stat
group 3: stat
Statistics query

• Query
  – from controller to switch
  – through OFStatisticsRequest message
    • specify the entity
    • specify the type of statistics

• Stats Reply
  – from switch to controller
  – through OFStatistics message
    • a list of stats for all the requested entities
Example: byte counts of every flow

• Specify the entity:
  – by match/port

• In our example:
  – wildcards matching all flows/ports

```java
// specify all the flows on the switch
OFSFlowStatisticsRequest specificReq = new OFSFlowStatisticsRequest();
specificReq.setMatch(new OFMatch().setWildscards(OFMatch.OFPFW_ALL));
specificReq.setOutput(OFPort.OFPP_NONE.getValue());
List<OFStatistics> specificReqs = new ArrayList<OFStatistics>();
specificReqs.add(specificReq);
```
Example: byte counts of every flow

• Specify the type of statistics we are interested:
  – flow, aggregate, port, queue, etc.

• In our example:
  – OFStatisticsType.Flow

```java
// add the list to request object, specify the type of stats: FLOW
OFStatisticsRequest req = new OFStatisticsRequest();
req.setStatisticsRequestType(OFStatisticsTypeType.FLOW);
req.setStatistics(specificReqs);
int reqLen = req.getLengthU();
reqLen += specificReq.getLength();
```
Example: byte counts of every flow

• Send request & get return value
  – Send the query to switch
  – Using java.util.concurrent.Future for asynchorous operation of getting return vaue

```java
IOFSwitch sw = this.floodlightProvider.getSwitch(swid);
Future<List<OFStatistics>> future = sw.queryStatics(req);
List<OFStatistics> values = future.get(10, TimeUnit.SECONDS);
for (OFStatistics stat : values) {
    if (stat instanceof OFFlowStatisticsReply) {
        OFFlowStatisticsReply flowstat = (OFFlowStatisticsReply)stat
        ...
    }
}
```
Processing Messages from Switches

Control Logic

Message handler

Floodlight API

FlowMod
StatReq
PacketIn
Processing Messages from Switches

• Basic operations:

  – Modules register themselves as interested in some type of message, along with a message handler

  – Every message of that type from any switch to the controller triggers all registered message handlers
Example: handling Packet_In messages

- Any packet received on a switch not matching any flow will trigger a packet_in message sent to the controller.
- Controller triggers all the module registered on this message.

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=10.10.10.1</td>
<td>out:2</td>
</tr>
<tr>
<td>src=10.10.10.2</td>
<td>out:3</td>
</tr>
</tbody>
</table>
Example: handling Packet_In messages

- Handling Packet_In in a prototype module:

```java
public class MyModule implements IOFMessageListener, IFloodlightModule {
    ...
    @Override
    public void startUp(FloodlightModuleContext context) {
        // register the module itself as one of message listener
        ...
    }
    @Override
    public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
        // the message handler implementation
        ...
    }
    ...
}
```
Example: handling Packet_In messages

- Message handler registering:

```java
@Override
public void startup(FloodlightModuleContext context) {
    floodlightProvider.addOFMessageListener(Oftype.PACKET_IN, this);
    ...
}
```

- Message handler

```java
@Override
public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
    Command c = Command.CONTINUE;
    if (msg.getType() == OFtype.PACKET_IN) {
        OFPacketIn pi = (OFPacketIn) msg;
        OFMatch match = new OFMatch;
        match.loadFromPacket(pi.getPacketData(), pi.getInPort());
        ...
    }
    return c;
}
```

obtain the match from a packet_in message
Basic functionality

• **Topology discovery**
  – LLDP protocol

• **Flow installation/deletion**
  – install/modify/delete a flow on a switch
    • flow is defined as all packets with the same match

• **Stats query**
  – packet counts
  – flow counts
  – port stats query
  – etc.
Topology Management

• Floodlight internally discovers and maintains the network topology
  – LinkDiscoveryManager
  – using link layer discovery protocol (LLDP)

• Expose APIs for:
  – topology query
  – listening on topology changes
Topology Management

- Init floodlight utility:
  - IFloodlightProviderService
  - ILinkDiscoveryService

```java
public class MyModule implements IIoTMessageListener, IFloodlightModule,
                        IIoTLinkDiscoveryListener, IOFSwitchListener {
    protected ILinkDiscoveryService linkDiscoverer;
    protected IFloodlightProviderService floodlightProvider;
    ...

    @Override
    public void init(FloodlightModuleContext context) {
        ...
        this.floodlightProvider =
            context.getServiceImpl(IFloodlightProviderService.class);
        // add self as one of switch events listeners
        this.floodlightProvider.addOFSwitchListener(this);

        this.linkDiscoverer =
            context.getServiceImpl(ILinkDiscoveryService.class);
        // add self as one of link events listeners
        this.linkDiscoverer.addListener(this);
        ...
    }
    ...
```
Topology Management

• Topology query: device status
  – get all switches (ids)
    ```java
    this.floodlightProvider.getAllSwitchDpids();
    ```
  – get a particular switch
    ```java
    IOFSwitch sw = this.floodlightProvider.getSwitch(swid);
    ```
  – get ports on a switch
    ```java
    Collection<ImmutablePort> ports = sw.getPorts();
    ```
  – etc.
Topology Management

- Topology query: connectivity status
  - get all links:
    ```java
    Map<Link, LinkInfo> links = this.linkDiscoverer.getLinks();
    ```
  - get end points of a link
    ```java
    Link l = ...;
    long dstDpid = l.getDst();
    long srcDpid = l.getSrc();
    short dstPort = l.getDstPort();
    short srcPort = l.getSrcPort();
    ```
  - etc.
Topology Management

• Listen to network topo changes:
  – step 1: register the module as listener

```java
public class MyModule implements IOFMessageListener, IFloodlightModule,
   ILinkDiscoveryListener, IOFSwitchListener {
    ...
    @Override
    public void init(FloodlightModuleContext context) {
      ...
      this.floodlightProvider =
          context.getServiceImpl(IFloodlightProviderService.class);
      //add self as one of switch events listeners
      this.floodlightProvider.addOFSwitchListener(this);
      ...
      this.linkDiscoverer =
          context.getServiceImpl(ILinkDiscoveryService.class);
      //add self as one of link events listeners
      this.linkDiscoverer.addListener(this);
    }
    ...
  }
}
Topology Management

- Listen to network topo changes:
  - step 2: implement event handler

```java
public class MyModule implements IOFMessageListener, IFloodlightModule,
  ILinkDiscoveryListener, IOFSwitchListener {

  ...

  @Override
  public void switchActivated(long switchId) {
    //handler of new switch connection event
    ...
  }

  @Override
  public void switchRemoved(long switchId) {
    //handler of switch disconnection event
    ...
  }

  @Override
  public void linkDiscoveryUpdate(List<LDUpdate> updateList) {
    //handler of link status change event
    ...
  }

  ...
}
```
Dealing with ARP

• Example: Host A (10.0.0.1) wants to talk to Host B (10.0.0.2)
  – A broadcast request:
    • "I need the MAC address of the guy with IP 10.0.0.2"
    • with a fake target MAC address ff:ff:ff:ff:ff:ff
  – B is the one (and the only one) that respond with its MAC address
  – A cache the mapping and sets up TCP communication
Dealing with ARP

• Address resolution protocol (ARP):
  – In Ethernet, hosts use MAC address to talk to each other
  – However, when setting up TCP connection, only IP address is specified.
  – Need to map TCP address to MAC address (address resolution)
Dealing with ARP

• In Floodlight, ARP requests will be forwarded to the controller

• Meaning we need to handle ARP request properly, otherwise hosts will have trouble prior to setting up connections

• By forwarding them on the appropriate port
Helpful links/References

• Step-by-step setting up in Eclipse:
  – http://www.openflowhub.org/display/floodlightcontroller/How+to+Write+a+Module

• Floodlight REST API:
  – http://www.openflowhub.org/display/floodlightcontroller/Floodlight+REST+API

• Message Processing/adding REST API:
  – http://www.openflowhub.org/display/floodlightcontroller/Advanced+Tutorial

• Dealing with wildcards:
  – http://www.openflowhub.org/display/floodlightcontroller/Wildcards+Mini-Tutorial