Floodlight tutorial

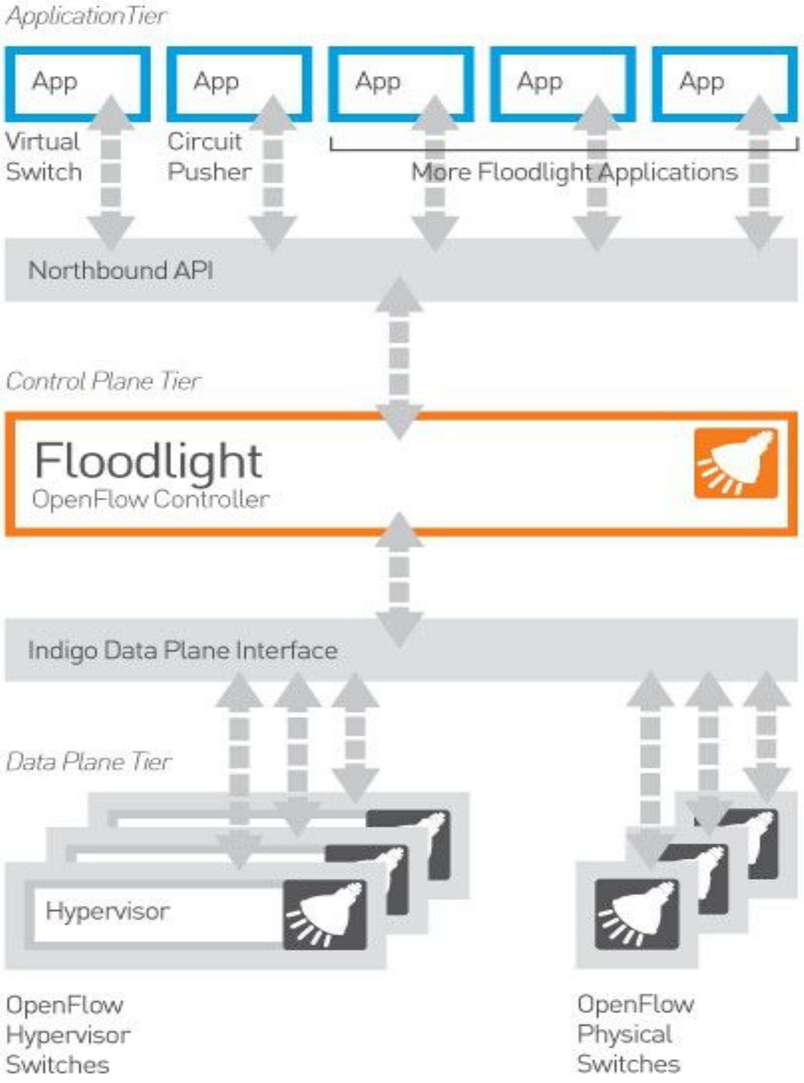
Chen Liang

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 definted as all packets with the same match
- Stats query
  - packet counts
  - flow counts
  - port stats query
  - etc.

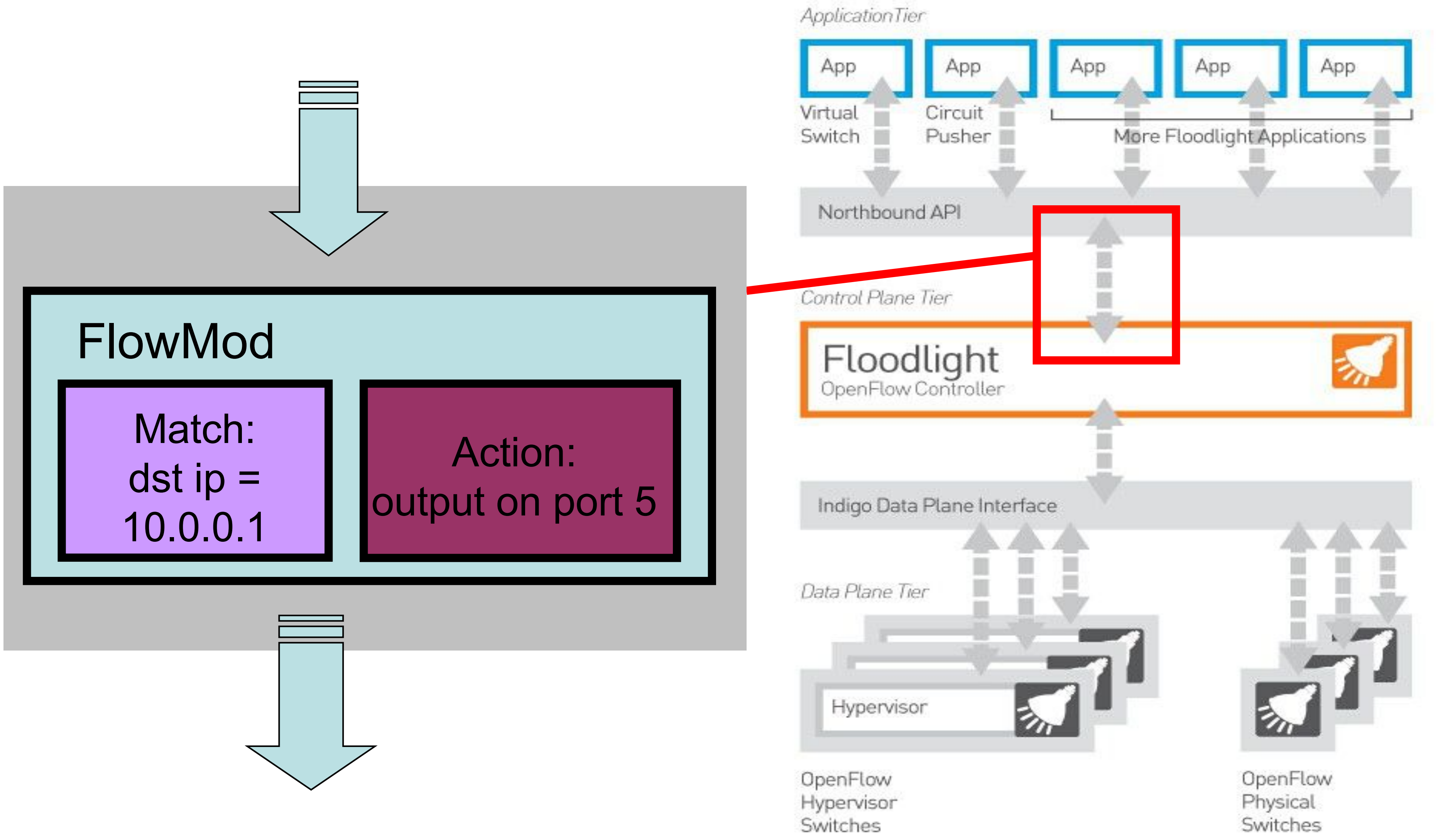
Basic functionality

- Topology discovery
  - LLDP protocol

- Flow installation/deletion
  - install/modify/delete a flow on a switch
    - flow is definted 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

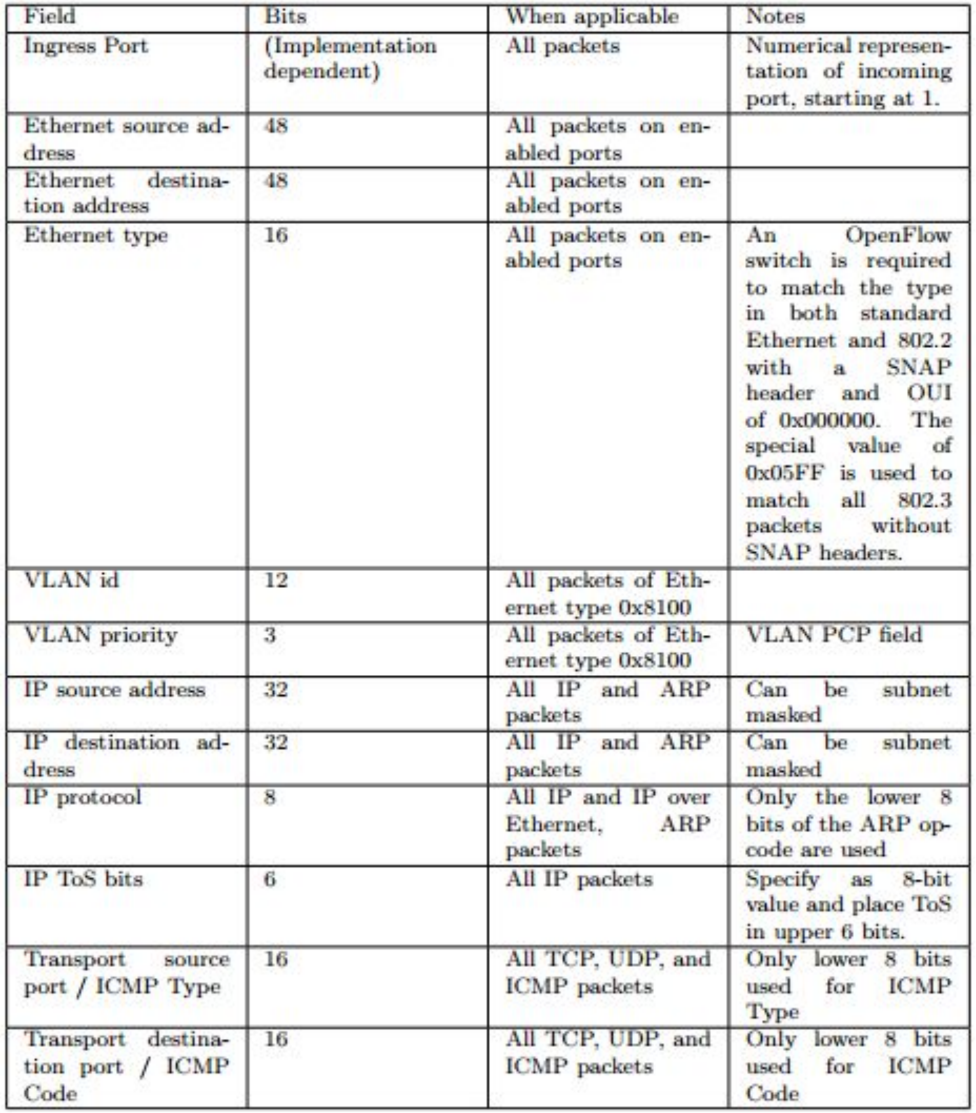
Action:
output on port 5

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



Field	Bits	When applicable	Notes
Ingress Port	(Implementation dependent)	All packets	Numerical representation of incoming port, starting at 1.
Ethernet source address	48	All packets on enabled ports	
Ethernet destination address	48	All packets on enabled ports	
Ethernet type	16	All packets on enabled ports	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.
VLAN id	12	All packets of Ethernet type 0x8100	
VLAN priority	3	All packets of Ethernet type 0x8100	VLAN PCP field
IP source address	32	All IP and ARP packets	Can be subnet masked
IP destination address	32	All IP and ARP packets	Can be subnet masked
IP protocol	8	All IP and IP over Ethernet, ARP packets	Only the lower 8 bits of the ARP opcode are used
IP ToS bits	6	All IP packets	Specify as 8-bit value and place ToS in upper 6 bits.
Transport source port / ICMP Type	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Type
Transport destination port / ICMP Code	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Code

Background: Subnet masks

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

	Binary Form	Dot-decimal notation
IP address	11000000.10101000.00000101.10000010	192.168.5.130
Subnet mask	11111111.11111111.11111111.00000000	255.255.255.0
Network prefix	11000000.10101000.00000101.00000000	192.168.5.0
Host part	00000000.00000000.00000000.10000010	0.0.0.130

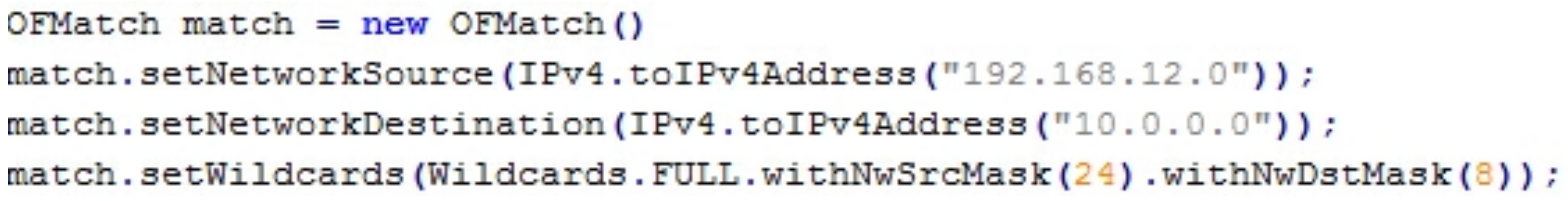
  - For 192.168.5.130/26:

	Binary Form	Dot-decimal notation
IP address	11000000.10101000.00000101.10000010	192.168.5.130
Subnet mask	11111111.11111111.11111111.11000000	255.255.255.192
Network prefix	11000000.10101000.00000101.10000000	192.168.5.128
Host part	00000000.00000000.00000000.00000010	0.0.0.2

Flow installation: Match

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

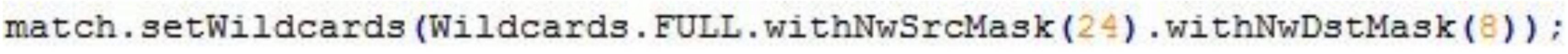
- i.e. to create a match for flow:
  - <src ip: 192.168.12.0/24, dst ip: 10.0.0.0/8>



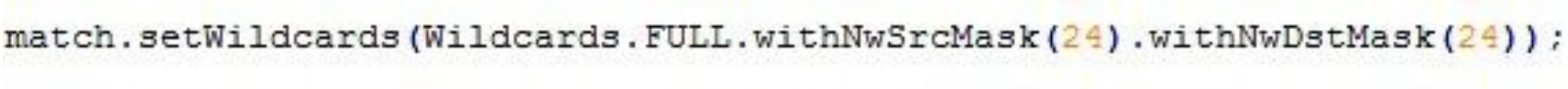
```
OFMatch match = new OFMatch()
match.setNetworkSource(IPv4.toIPv4Address("192.168.12.0"));
match.setNetworkDestination(IPv4.toIPv4Address("10.0.0.0"));
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
```

Flow installation: Match

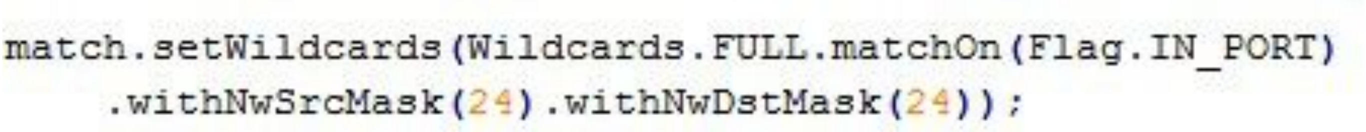
- Make sure the wildcards is set correctly:
  - the following three are all different matches



```
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8));
```

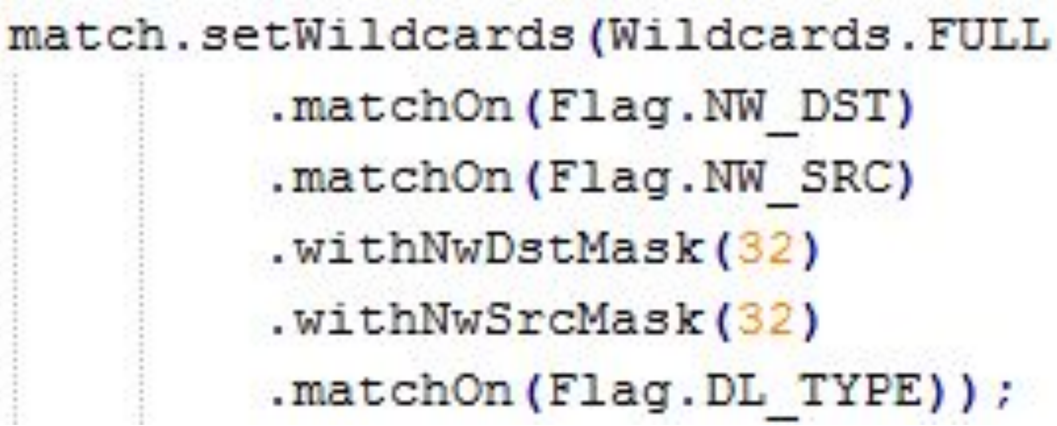


```
match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(24));
```



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

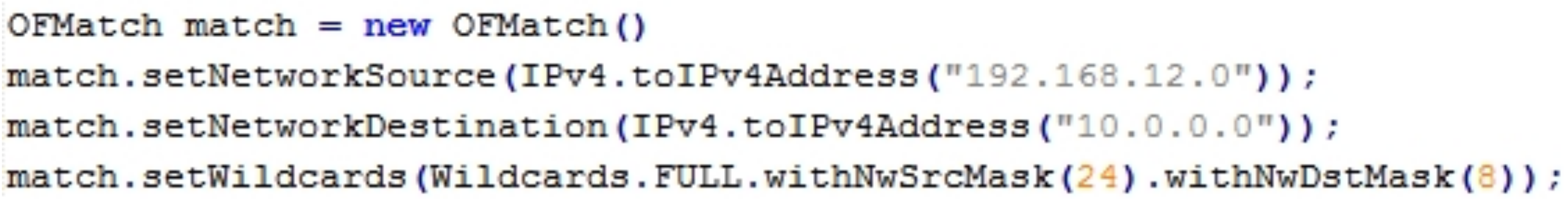
- An example: A match on the fields of in_port, src_ip (full match) and dst_ip (full match) shoud be set as



```
match.setWildcards(Wildcards.FULL
            .matchOn(Flag.NW_DST)
            .matchOn(Flag.NW_SRC)
            .withNwDstMask(32)
            .withNwSrcMask(32)
            .matchOn(Flag.DL_TYPE));
```

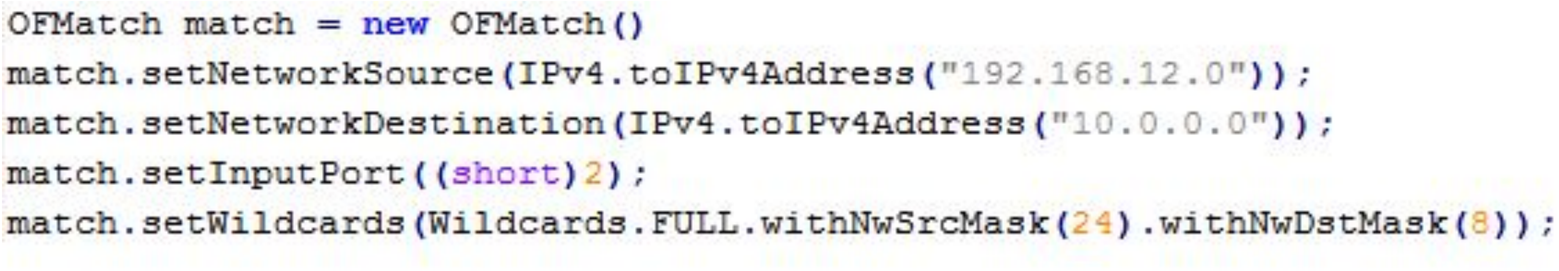
Flow installation: Match

- 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

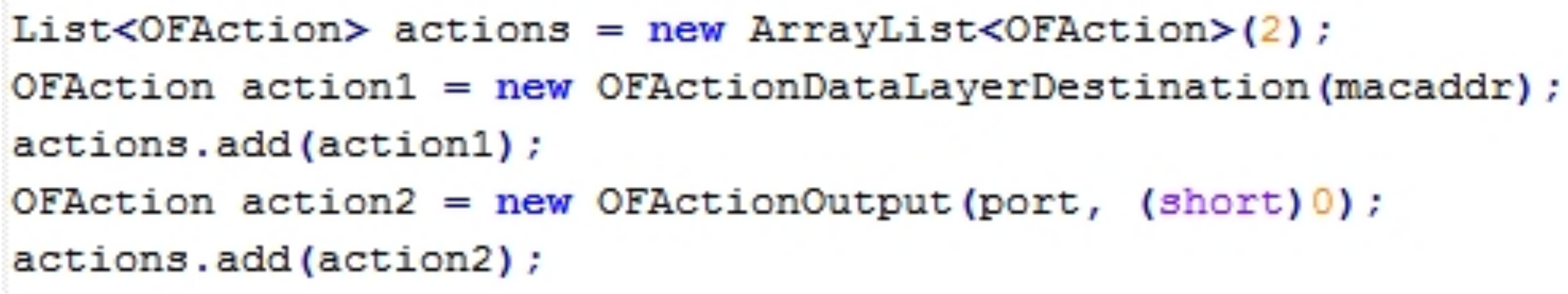
- examples of Actions:
  - <output on port 2>
  - <set dst IP address to 10.0.0.3>
  - <set mac address to 00:0a:95:9d:68:16>

Flow installation: Action

- In Floodlight, each actions is a object of org.openflow.protocol.OFAction
  - org.openflow.protocol.action.OFAction

- 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 specfied port



```
List<OFAction> actions = new ArrayList<OFAction>(2);
OFAction action1 = new OFActionDataLayerDestination(macaddr);
actions.add(action1);
OFAction action2 = new OFActionOutput(port, (short)0);
actions.add(action2);
```

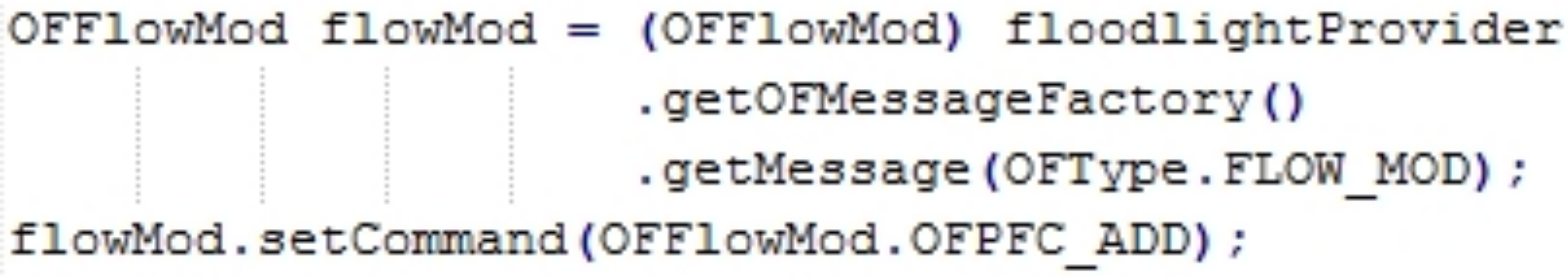
Flow installation: FlowMod

- 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 a object of OFFlowMod:
  - org.openflow.protocol.OFFlowMod

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



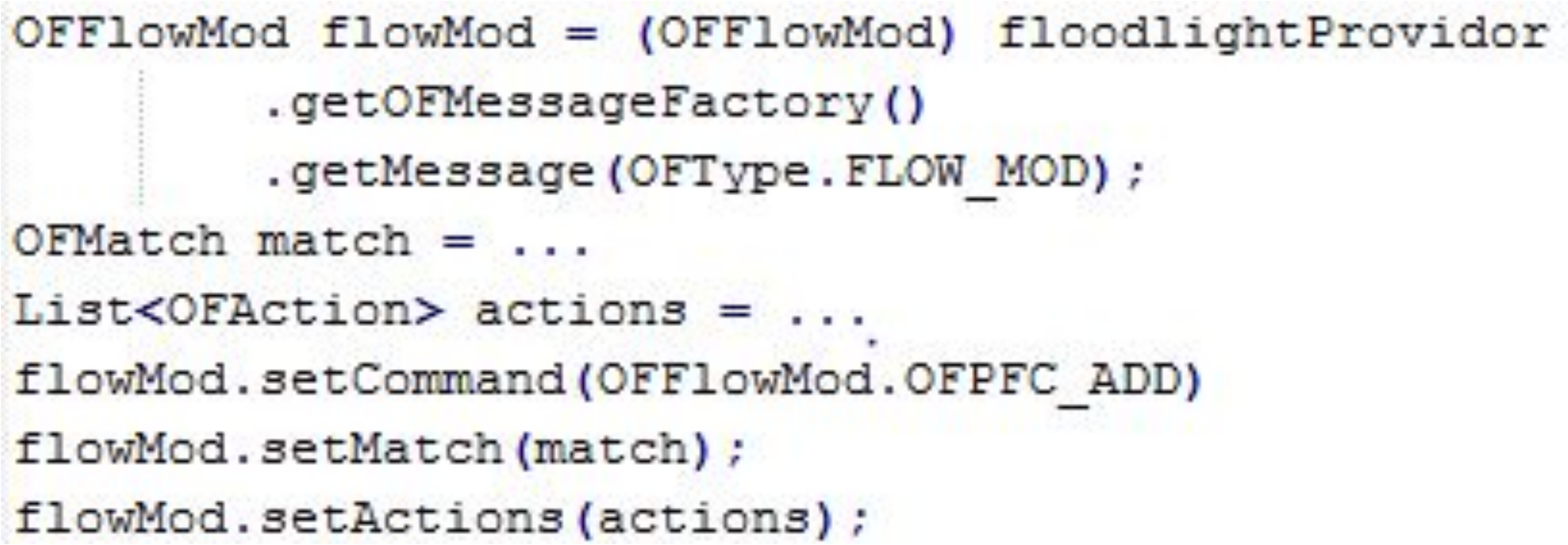
```
OFFlowMod flowMod = (OFFlowMod) floodlightProvider
                    .getOFMessageFactory()
                    .getMessage(OFType.FLOW_MOD);
flowMod.setCommand(OFFlowMod.OFPFC_ADD);
```

Putting togather

- 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

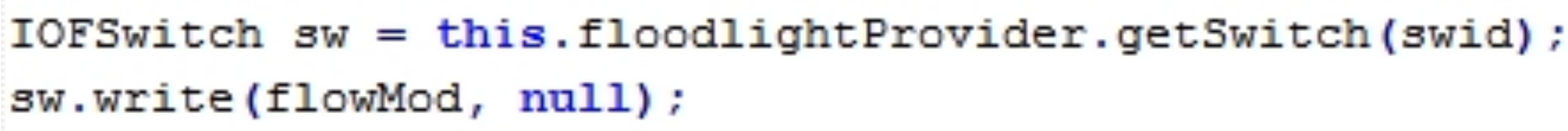
Putting togather

- create the message, set match and actions



```
OFFlowMod flowMod = (OFFlowMod) floodlightProvidor
        .getOFMessageFactory()
        .getMessage(OFType.FLOW_MOD);
OFMatch match = ...
List<OFAction> actions = ...
flowMod.setCommand(OFFlowMod.OFPFC_ADD)
flowMod.setMatch(match);
flowMod.setActions(actions);
```

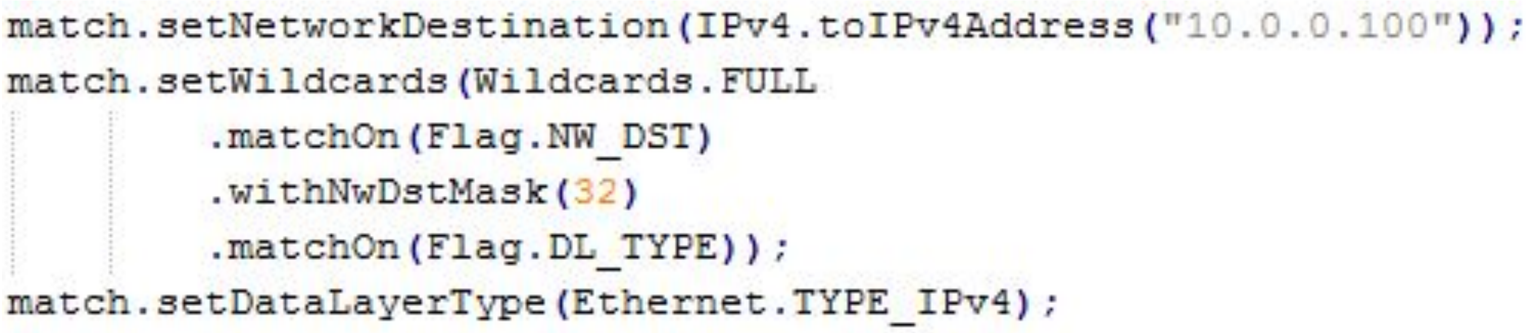
- send the message to the switch:



```
IOFSwitch sw = this.floodlightProvider.getSwitch(swid);
sw.write(flowMod, null);
```

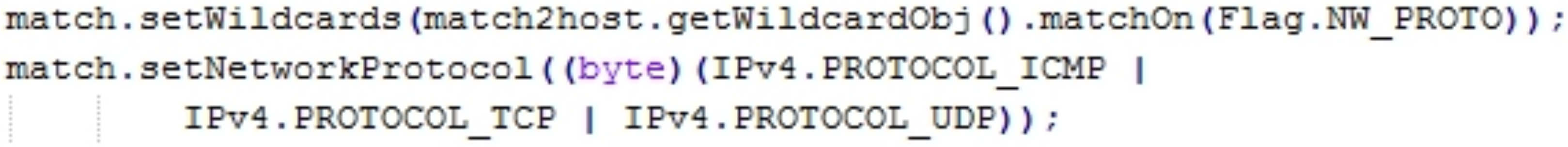
In dealing with IP packets

- Need to properly set datalayer type and netmask mask
  - Example: setup a flow matching on dst_ip=10.0.0.100 (no subnet)



```
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);
```

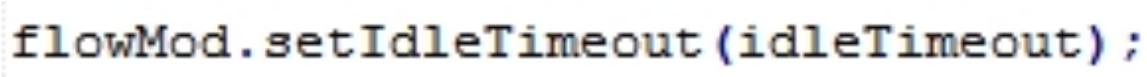
  - optionally, you can further specify network layer protocol by further specifying:



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

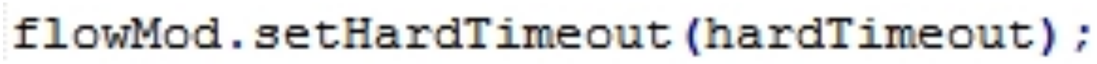
Optional Fields of FlowMod

- fields in FlowMod to specify optional properties for a flow, i.e.:
  - set idle timeout



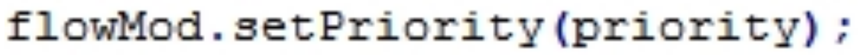
```
flowMod.setIdleTimeout(idleTimeout);
```

  - set hard time out



```
flowMod.setHardTimeout(hardTimeout);
```

  - set priority

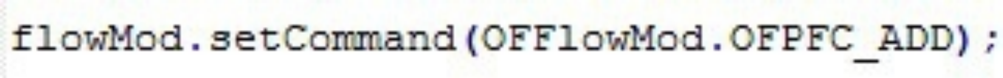


```
flowMod.setPriority(priority);
```

  - etc.

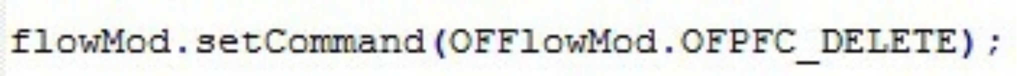
Flow deletion/modification

- Almost the same as adding a flow, except:
  - Changing



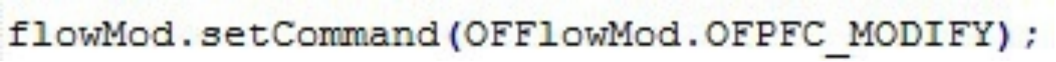
```
flowMod.setCommand(OFFlowMod.OFPFC_ADD);
```

- to



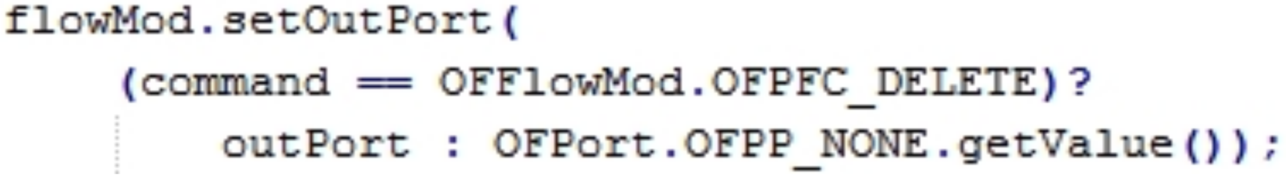
```
flowMod.setCommand(OFFlowMod.OFPFC_DELETE);
```

- or



```
flowMod.setCommand(OFFlowMod.OFPFC_MODIFY);
```

  - And need to specify outport for deletion

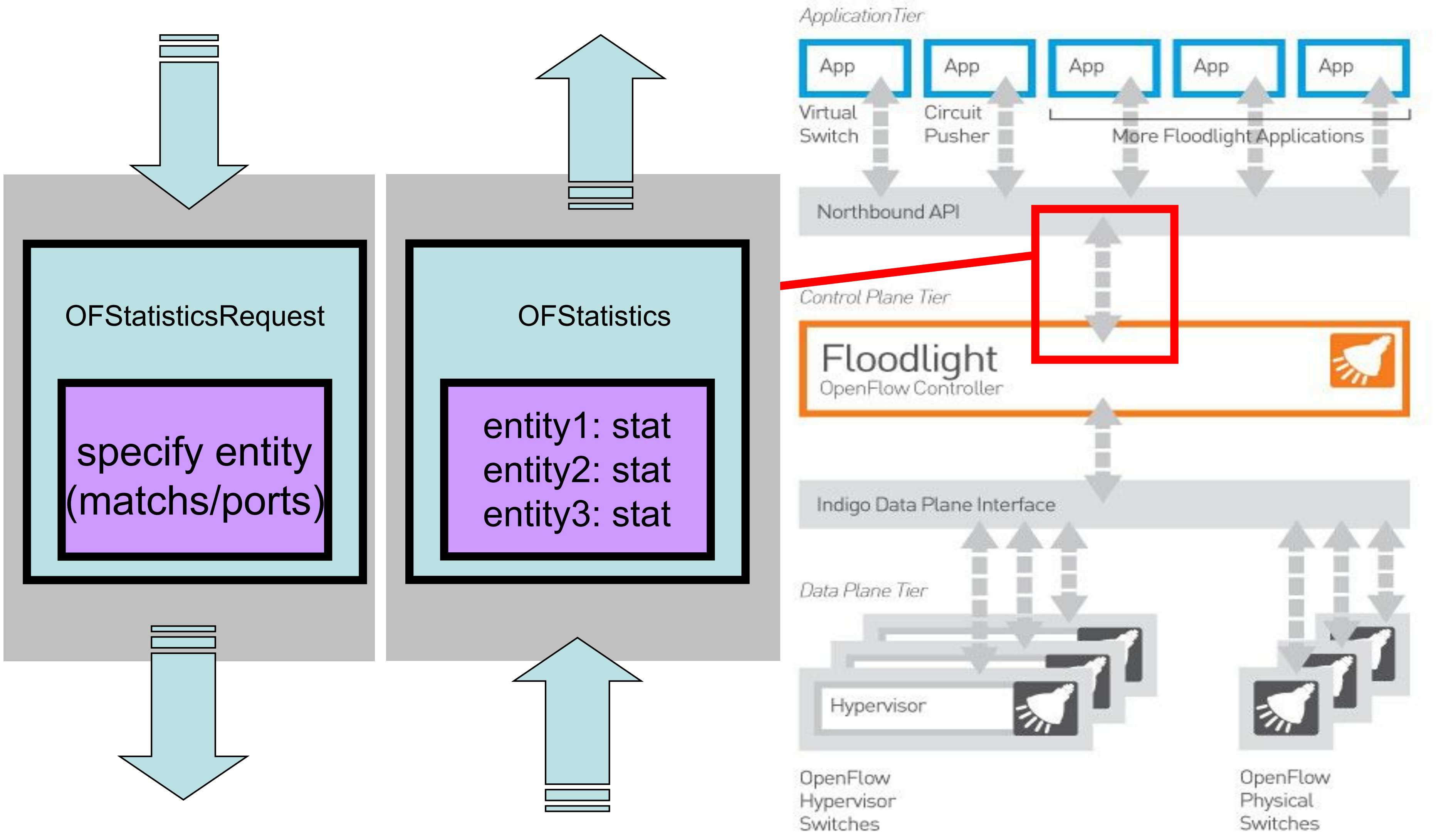


```
flowMod.setOutPort(
        (command == OFFlowMod.OFPFC_DELETE)?
            outPort : OFPort.OFPP_NONE.getValue());
```

Basic functionality

- Topology discovery
  - LLDP protocol
- Flow installation/deletion
  - install/modify/delete a flow on a switch
    - flow is definted as all packets with the same match
- Stats query
  - packet counts
  - flow counts
  - port stats query
  - etc.

Statistics query



OFStatisticsRequest

specify entity (matchs/ports)

OFStatistics

entity1: stat
entity2: stat
entity3: stat

Statistics query

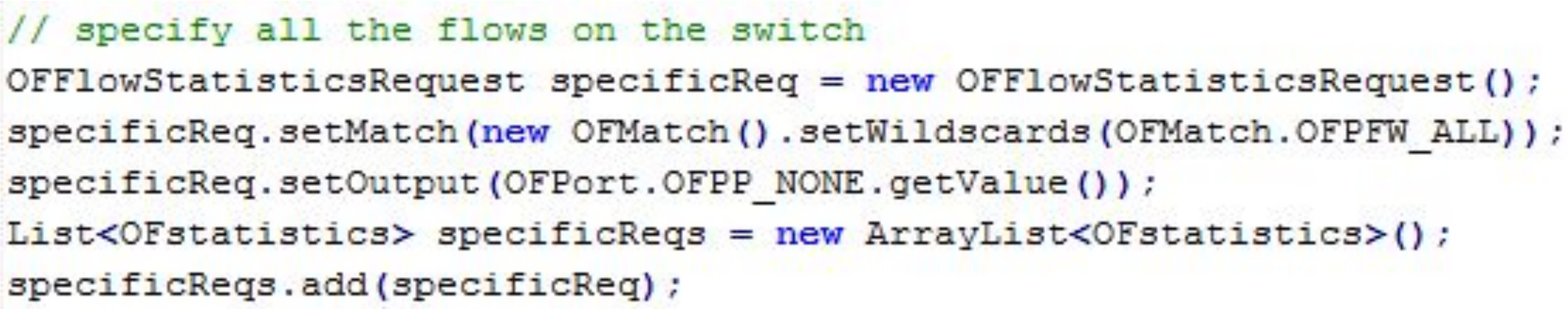
- Query
  - from controller to switch
  - through OFStatisticsRequest message
    - specify the entity
    - specify the type of statistics

- 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

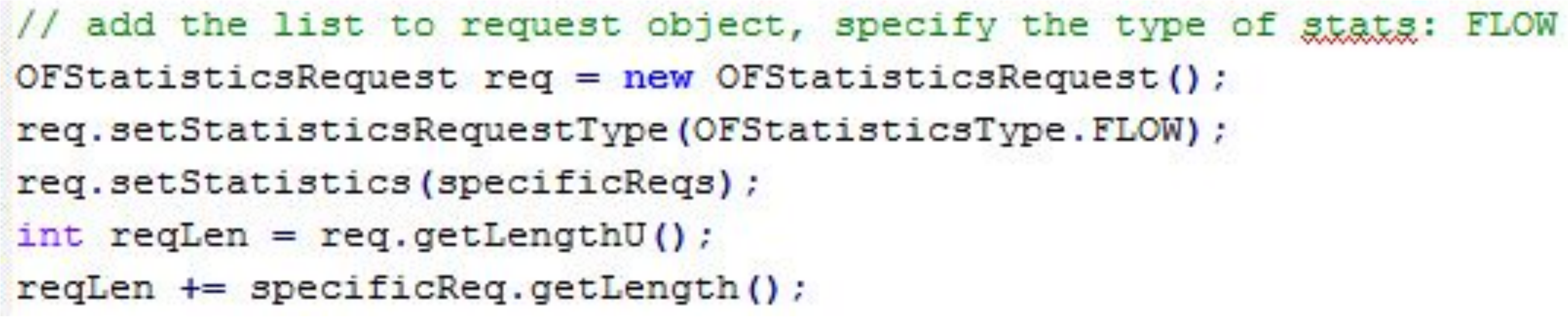


```
// specify all the flows on the switch
OFFlowStatisticsRequest specificReq = new OFFlowStatisticsRequest();
specificReq.setMatch(new OFMatch().setWildscards(OFMatch.OFPFW_ALL));
specificReq.setOutput(OFPort.OFPP_NONE.getValue());
List<OFstatistics> specificReqs = new ArrayList<OFstatistics>();
specificReqs.add(specificReq);
```

Example: byte counts of every flow

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

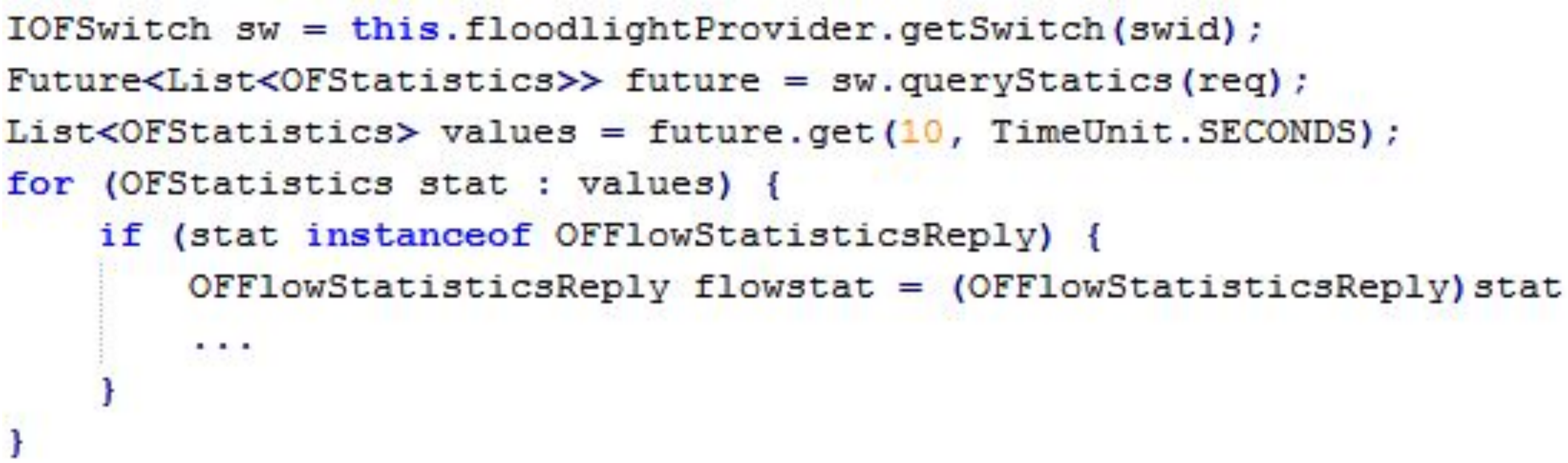
- In our example:
  - OFStatisticsType.Flow



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

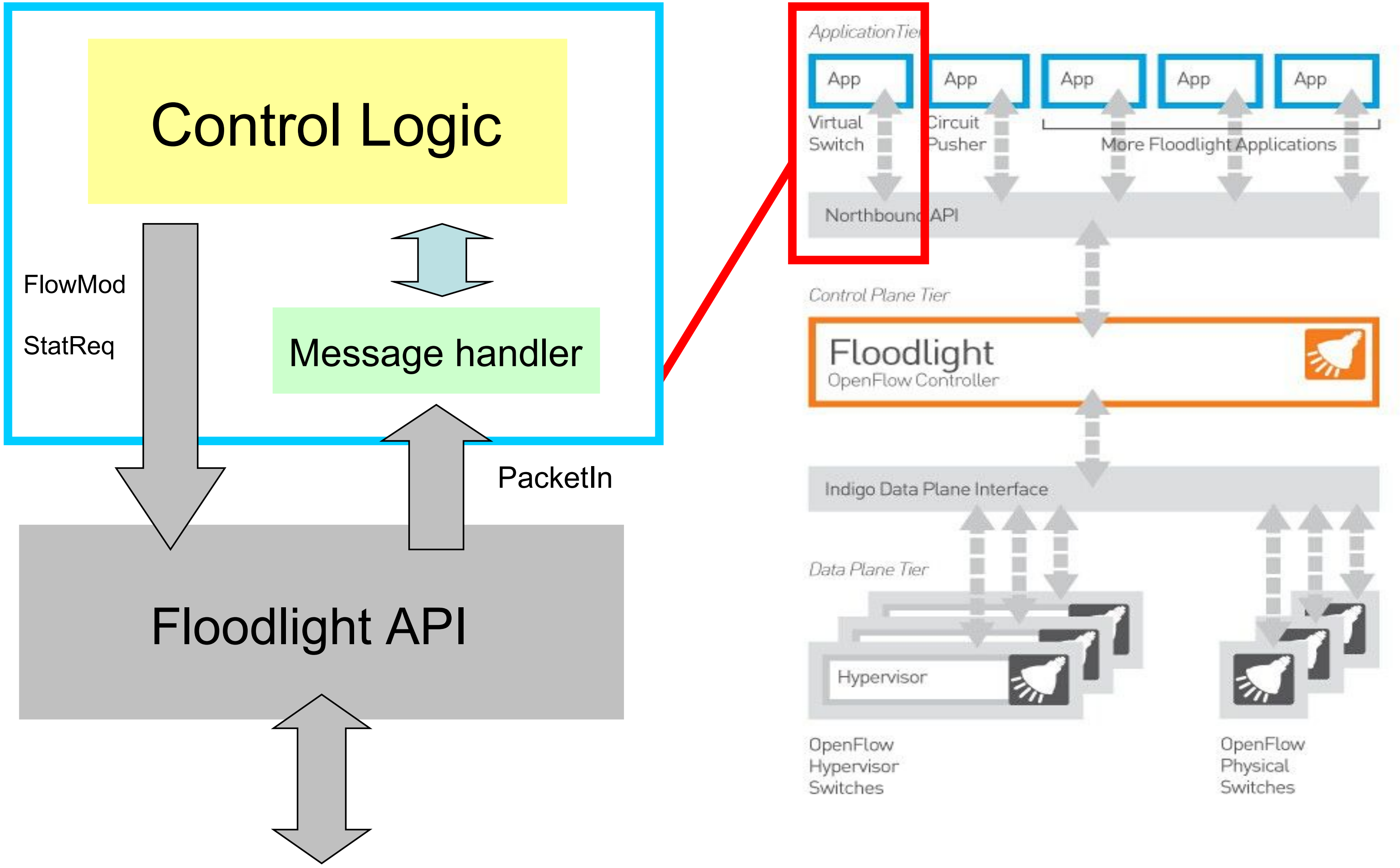
Example: byte counts of every flow

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



```
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

FlowMod

StatReq

Message handler

PacketIn

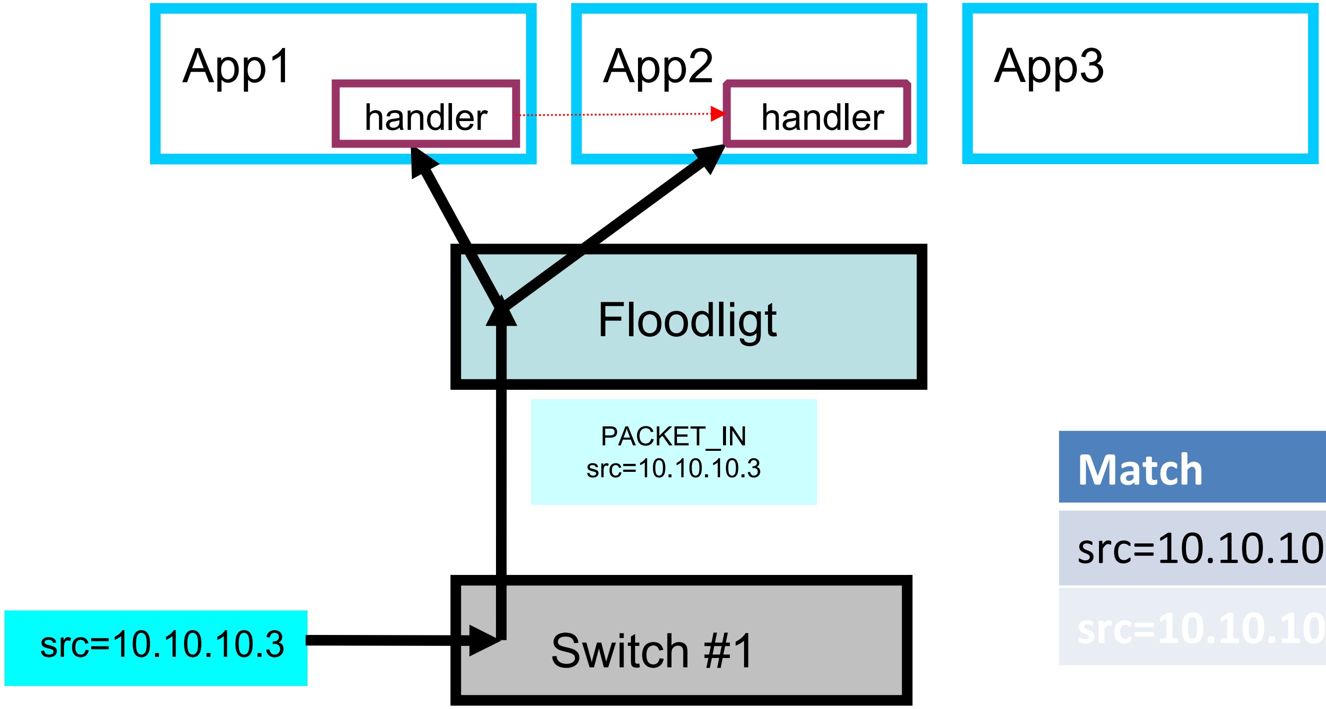
Floodlight API

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

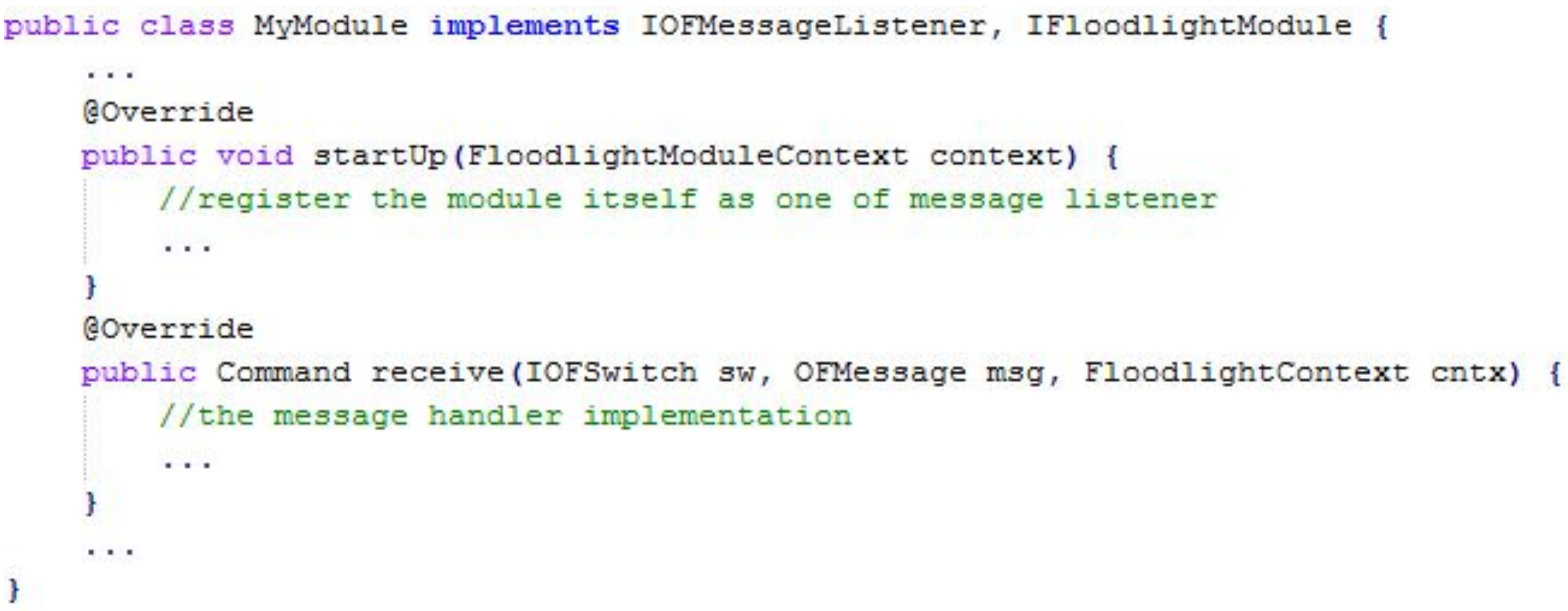
- Any packet received on a switch not matching any flow will trigger a packet_in message sent to the controller
- Controller triggers all the module registered on this message



Match	Action
src=10.10.10.1	out:2
src=10.10.10.2	out:3

Example: handling Packet_In messages

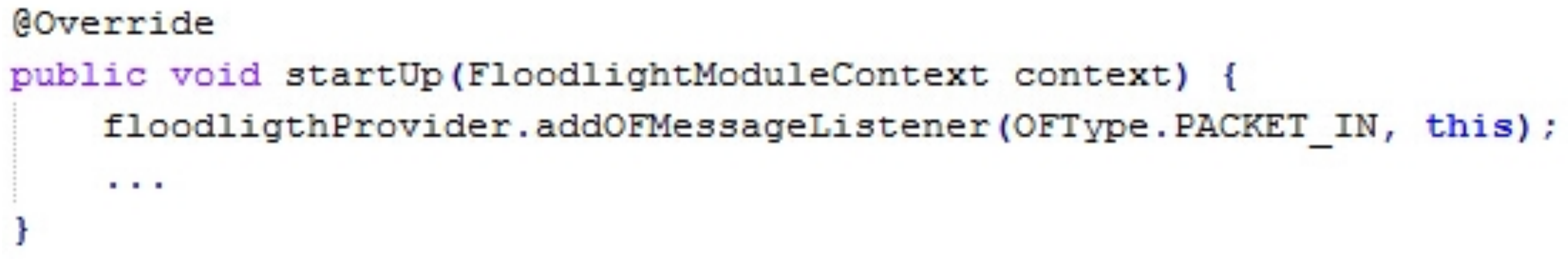
- Handling Packet_In in a prototype module:



```
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
        ...
    }
    ...
}
```

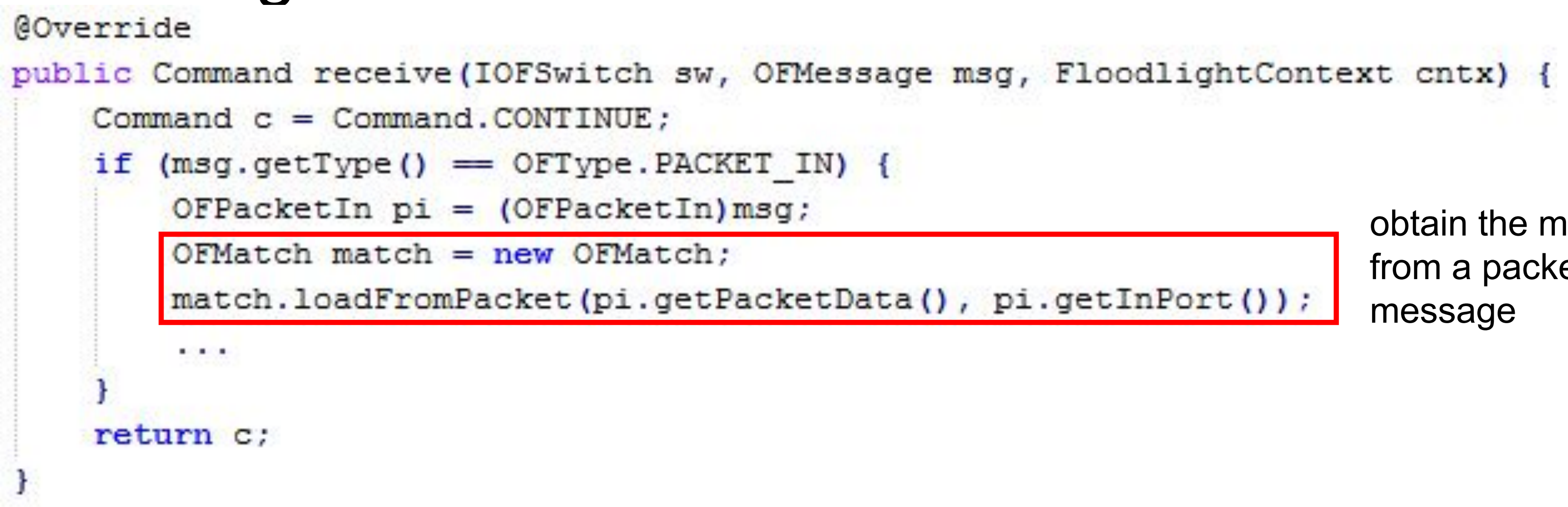
Example: handling Packet_In messages

- Message handler registering:



```
@Override
public void startUp(FloodlightModuleContext context) {
    floodligthProvider.addOFMessageListener(OFType.PACKET_IN, this);
    ...
}
```

- Message handler



```
@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 definted as all packets with the same match
- Stats query
  - packet counts
  - flow counts
  - port stats query
  - etc.

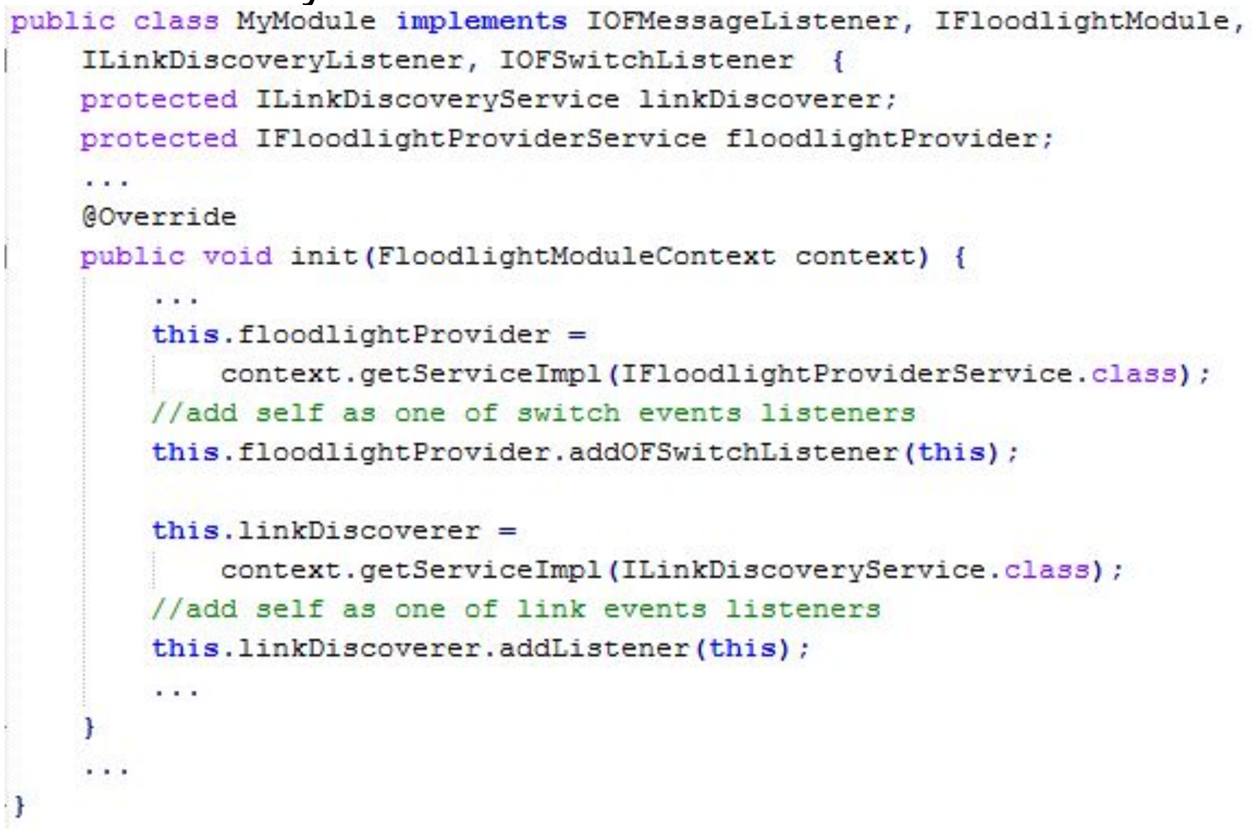
Topology Management

- 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

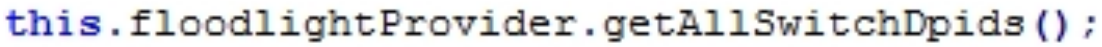


```
public class MyModule implements IOFMessageListener, IFloodlightModule,
        ILinkDiscoveryListener, IOFSwitchListener  {
    protected ILinkDiscoveryService linkDiscoverer;
    protected IFloodlightProviderService floodlightProvider;
    ...
    @Override
    public void init(FloodlightModuleContext 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);
        ...
    }
    ...
}
```

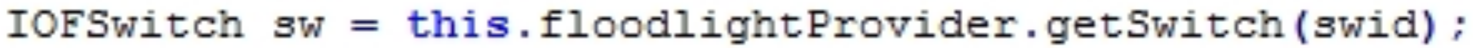
Topology Management

- Topology query: device status
  - get all switches (ids)



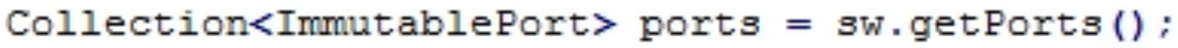
```
this.floodlightProvider.getAllSwitchDpids();
```

  - get a particular switch



```
IOFSwitch sw = this.floodlightProvider.getSwitch(swid);
```

  - get ports on a swith

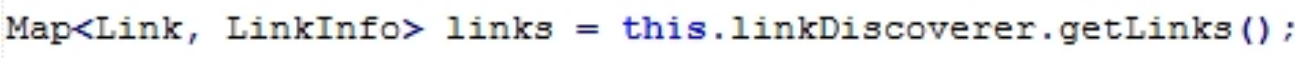


```
Collection<ImmutablePort> ports = sw.getPorts();
```

  - etc.

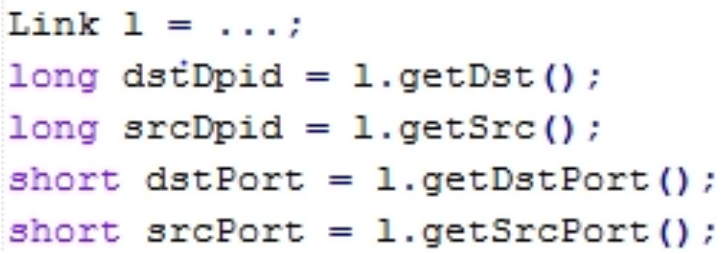
Topology Management

- Topology query: connetivity status
  - get all links:



```
Map<Link, LinkInfo> links = this.linkDiscoverer.getLinks();
```

  - get end points of a link

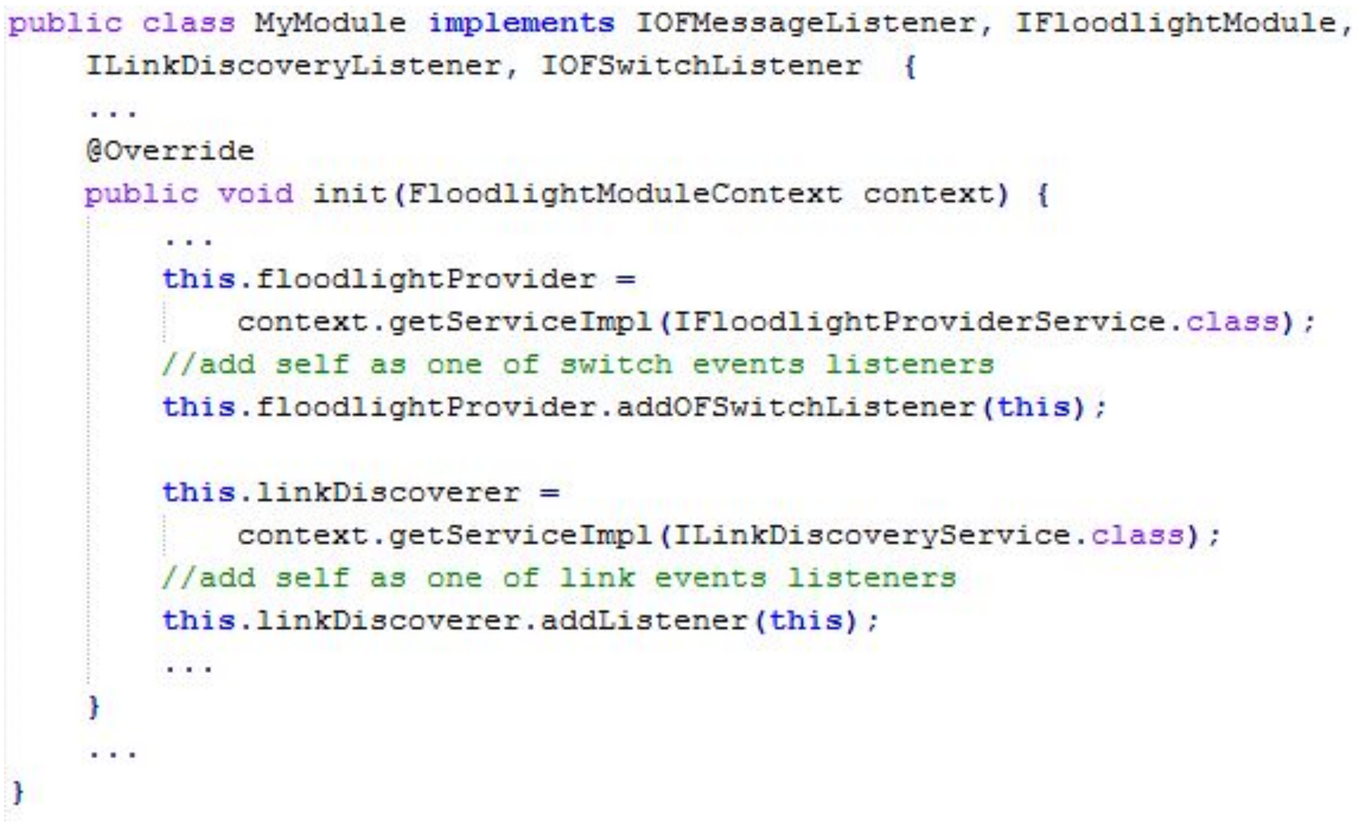


```
Link l = ...;
long dstDpid = l.getDst();
long srcDpid = l.getSrc();
short dstPort = l.getDstPort();
short srcPort = l.getSrcPort();
```

  - etc.

Topology Management

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

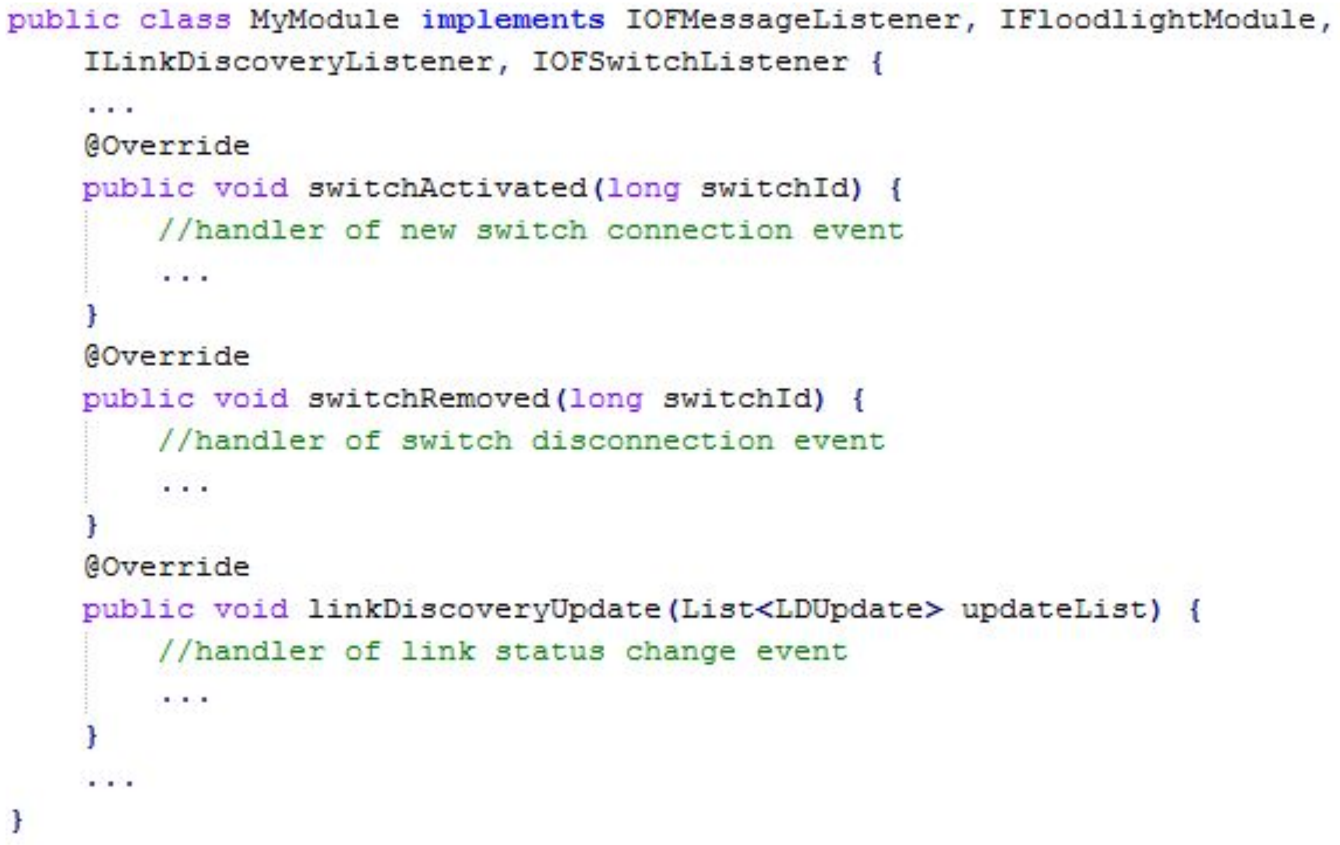


```
public class MyModule implements IOFMessageListener, IFloodlightModule,
    ILinkDiscoveryListener, IOFSwitchListener  {
    ...
    @Override
    public void init(FloodlightModuleContext context) {
        ...
        this.floodlightProvider =
            context.getServiceImpl(IFloodlightProviderService.class);
        //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 specifed.
  - 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 seting 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