Learning Floodlight through examples

JavaDoc:
http://floodlight.github.io/floodlight/javadoc/floodlight/overview-summary.html

Tutorials:
https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343514/Tutorials

Antonio Virdis
Assistant Professor@ University of Pisa
antonio.virdis@unipi.it

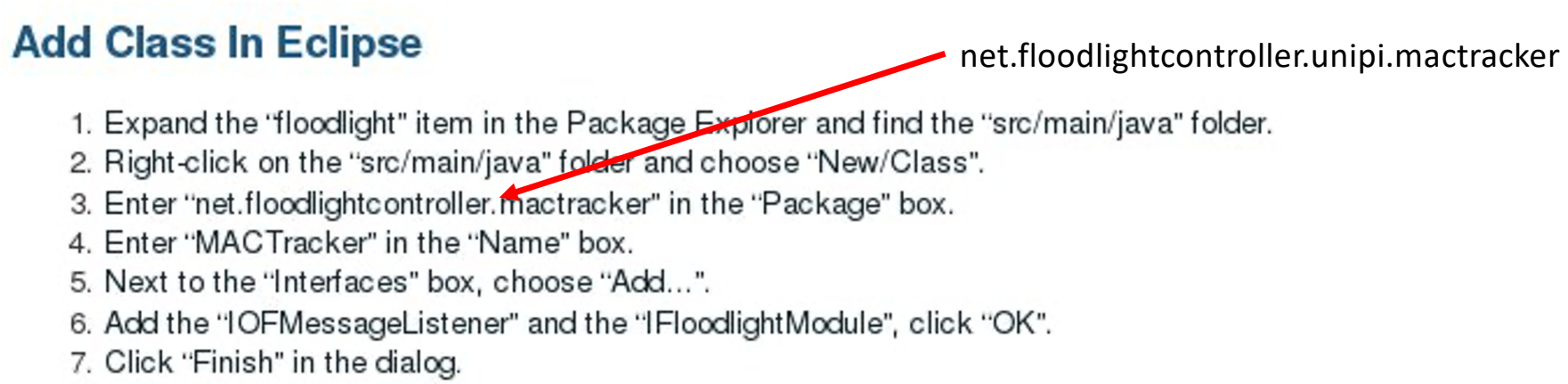
Architecture

- Floodlight is designed around a strongly modular architecture
- It mainly consist of
  - Controller modules
  - Application modules

- Modules can provide/exploit services through the use of Floodlight Services

New Floodlight Module

- To create a new module and add it to the pipeline you need to create a new Java class implementing the ***IOFMessageListener*** and ***IFloodlightModule*** interfaces
- Eclipse tools can be used to generate a skeleton:

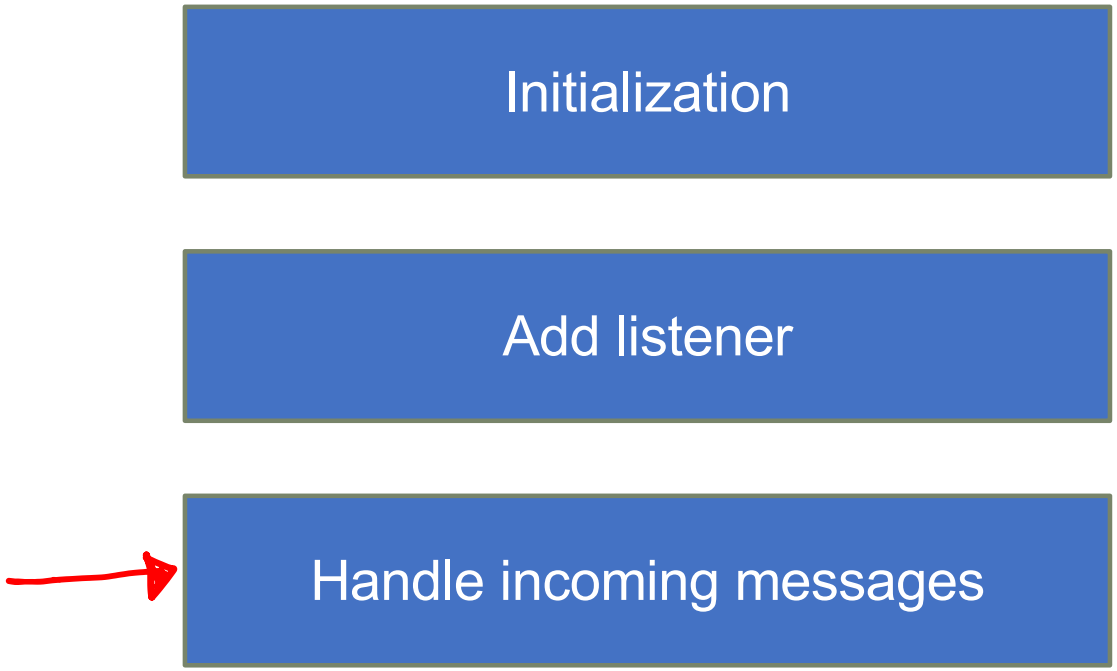


IFloodlightModule

[quoting the documentation]

- Defines an interface for loadable Floodlight modules. At a high level, these functions are called in the following order:
  - getServices() : what services does this module provide
  - getDependencies() : list the dependencies
  - init() : internal initializations (*doesn't* touch other modules)
  - startUp() : external initializations (*does* touch other modules)

Main Functions



Initialization and dependences

- Each module that wants to process OF packets need to connect with the ***FloodlightProvider*** which dispatches the messages
- Explicit dependency on its creation needs to be declared
- At initialization a reference to it needs to be gathered

```
protected IFloodlightProviderService floodlightProvider; // Reference to the provider
protected static Logger log;

// Called at initialization time. Retrieve reference to the provider
@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleException {
    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
    log = LoggerFactory.getLogger(MACTracker.class);
}

// Called to specify the dependences. Add dependency on the provider
@Override
public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
    Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
    l.add(IFloodlightProviderService.class);
    return l;
}
```

Initialization (2)

```
// Set module name
@Override
public String getName() {
    return ModuleExample.class.getSimpleName();
}

// Called at startup time (after all the modules have been initialized)
@Override
public void startUp(FloodlightModuleContext context) {
    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
}
```

Handle Packet-In Messages

- Each module that wants to process Packet-In messages needs to register and define a ***receive*** function (from ***IOFMessageListener***)
- **Parameters**:
  - sw - the OpenFlow switch that sent this message
  - msg - the message
  - cntx - a Floodlight message context object you can use to pass information between listeners
- **Returns**: the command to continue or stop the execution

```
     // Called every time a Packet-In is received
@Override
public net.floodlightcontroller.core.IListener.Command receive(IOFSwitch sw,     OFMessage
msg, FloodlightContext cntx) {

    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
      IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
    // Print the source MAC address
    Long sourceMACHash = Ethernet.toLong(eth.getSourceMACAddress().getBytes());
    System.out.printf("MAC Address: {%s} seen on switch: {%s}\n",
      HexString.toHexString(sourceMACHash),
           sw.getId());
    // Let other modules process the packet
    return Command.CONTINUE;
}
```

Register the new module

- Each needs to be registered in the pipeline

Append the name of the class in the file

```
src/main/resources/META-INF/services/net.floodlight.core.module.IFloodlightModule
net.floodlightcontroller.unipi.ModuleExample
```

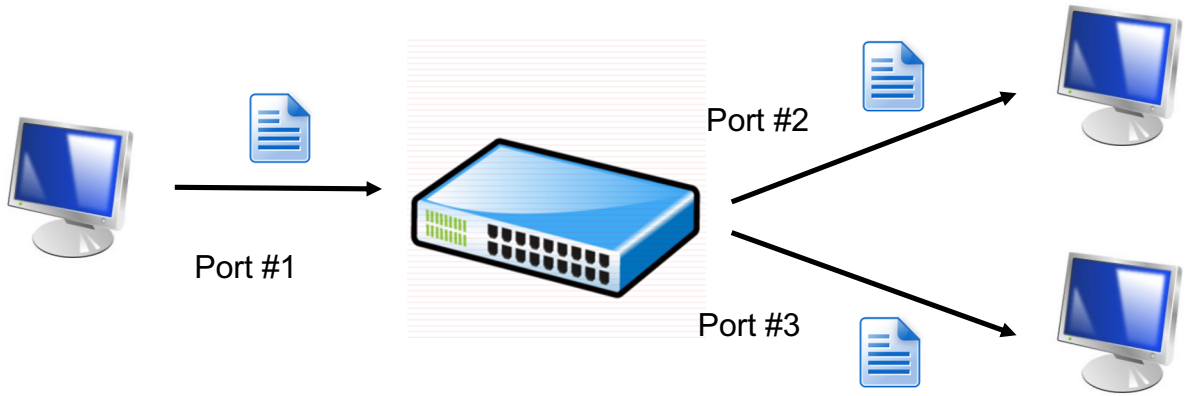
Add the module into the pipeline

```
src/main/resources/floodlightdefault.properties
floodlight.modules = <leave the default list of modules in place>,
net.floodlightcontroller.unipi.ModuleExample
```

- Test it!

Hub

- Retransmit the packet on all the ports of the switch
- Create a module that transforms the SDN switches into hubs that retransmit all the packets in all the ports



Hub

- All the packets are processed at the controller
- No new rules are installed

```
// within the receive function

// Cast to Packet-In
OFPacketIn pi = (OFPacketIn) msg;

// Create the Packet-Out and set basic fields
OFPacketOut.Builder pob = sw.getOFFactory().buildPacketOut();
pob.setBufferId(pi.getBufferId());

// Create action -> flood the packet on all the ports
OFActionOutput.Builder actionBuilder = sw.getOFFactory().actions().buildOutput();
actionBuilder.setPort(OFPort.FLOOD);

// Assign the action
pob.setActions(Collections.singletonList((OFAction) actionBuilder.build()));
```

Hub

- Packet waiting for being transmitted can be sent to the Controller encapsulated in the Packet-In message or can be buffered on the switch

```
// Packet might be buffered in the switch or encapsulated in Packet-In
// If the packet is encapsulated in Packet-In sent it back
if (pi.getBufferId() == OFBufferId.NO_BUFFER) {
     // Packet-In buffer-id is none, the packet is encapsulated -> send it back
     byte[] packetData = pi.getData();
     pob.setData(packetData);
}
// Send the Packet-Out
sw.write(pob.build());

// Interrupt the chain
return Command.STOP;
```

Generate Flow-Mods

- A Flow-Mod message can be generated to install new rules in the switches to flood all the packets in all the other ports, instead of processing all the packets at the controller

```
// Create a new rule to be added
OFFlowAdd.Builder fmb = sw.getOFFactory().buildFlowAdd();
fmb.setBufferId(pi.getBufferId()) // Link the new rule to the received OF PKT IN
   .setHardTimeout(20) // Set hard timeout
   .setIdleTimeout(10) // Set soft timeout
   .setPriority(32768) // Set priority
   .setXid(pi.getXid());

// Create a new action to be executed
OFActionOutput.Builder actionBuilder = sw.getOFFactory().actions().buildOutput();
actionBuilder.setPort(OFPort.FLOOD); // Set as action the flood of the packet
fmb.setActions(Collections.singletonList((OFAction) actionBuilder.build()));

// Send the Packet-Mod
sw.write(fmb.build());

// Interrupt the chain
return Command.STOP;
```

Play around with Flow-mods

- Disable the code for packet out
- Disable the forward module from the pipeline
- Try sending only one ping

  h1 ping –c 1 h2
- Check the flowtable on s1

  sudo ovs-ofctl dump-flows s1