Advanced Networking Architectures and Wireless Systems

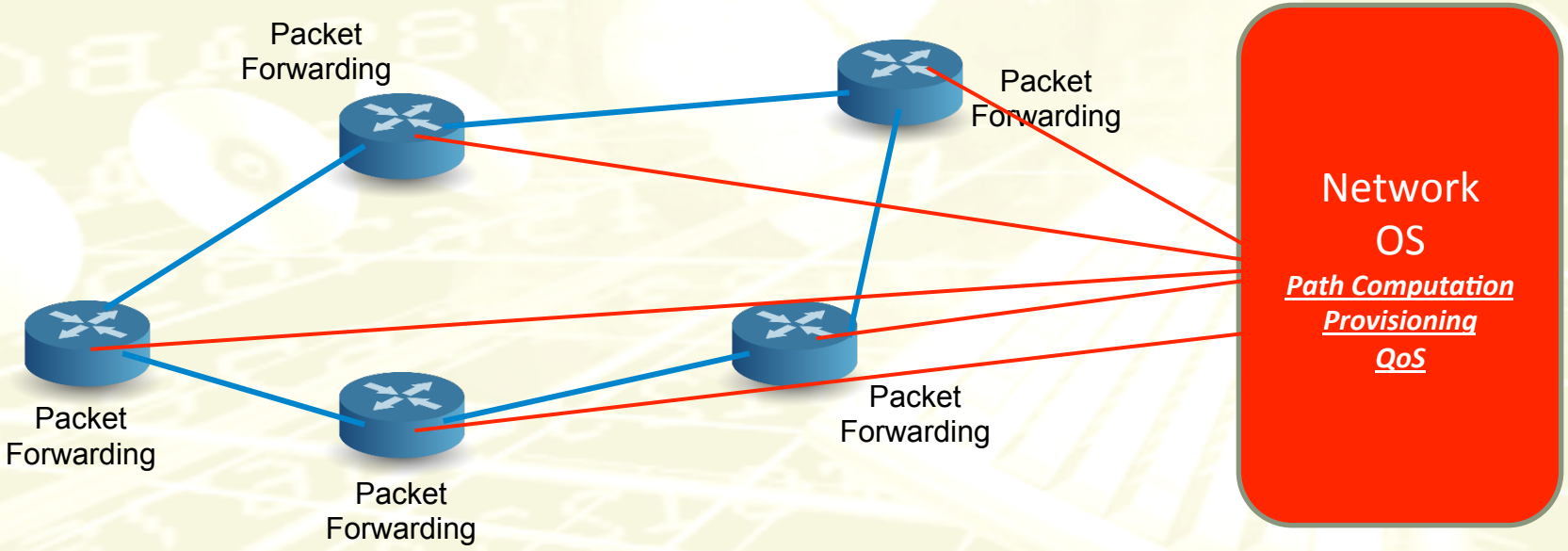
Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

Software Defined Networking

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

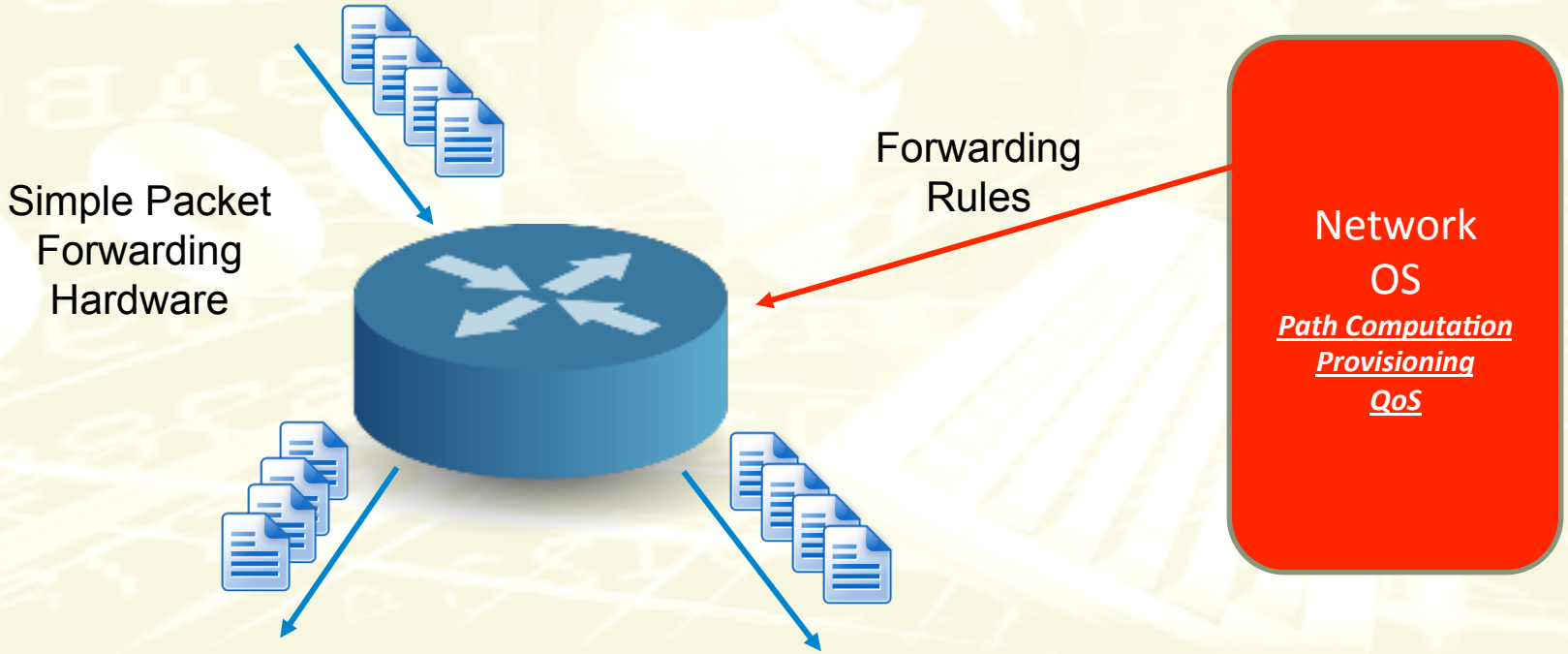
Concept

- ***Software defined networking***: Physical ***separation*** of the network **control** plane from the **forwarding** plane, where control plane controls several devices
- Centralization of control



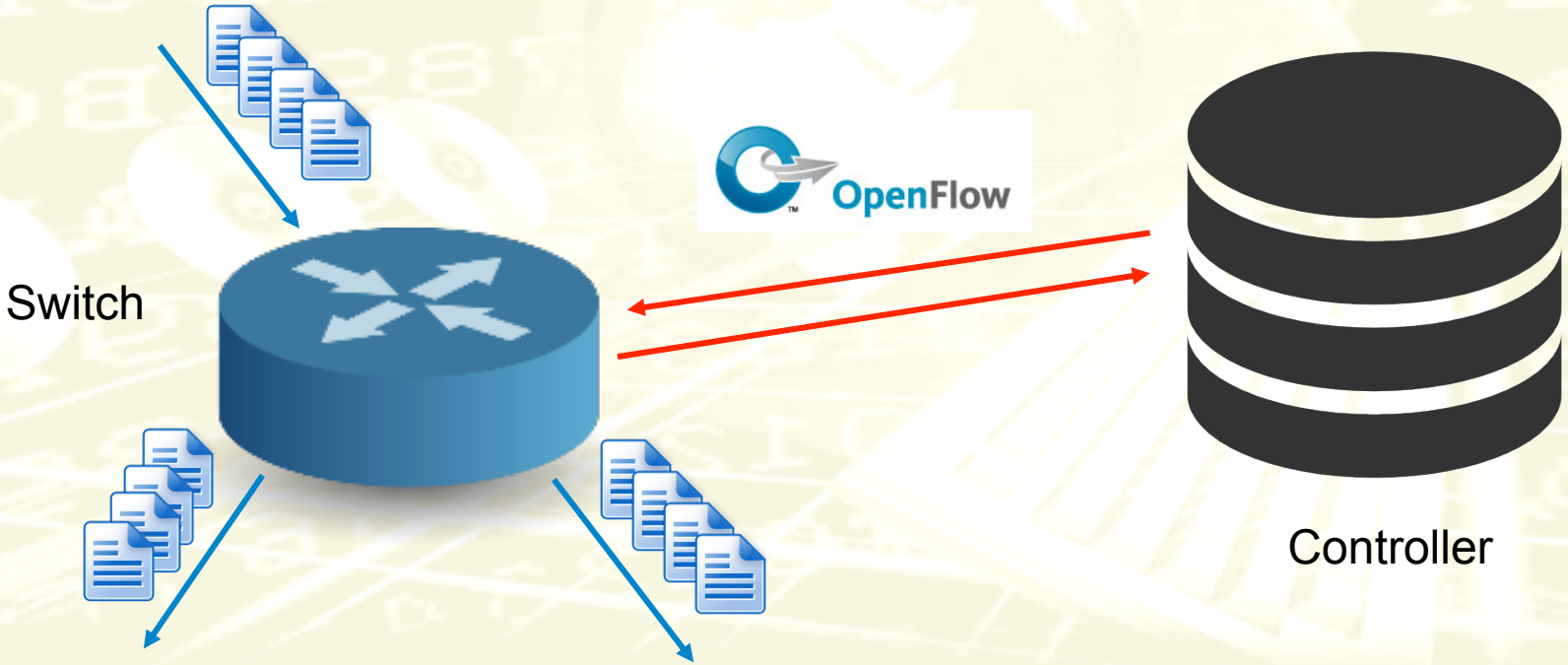
SDN Hardware

- Routers become ***simple hardware*** for packet forwarding (switch)
- A ***centralized controller*** is responsible for defining forwarding rules (controller)



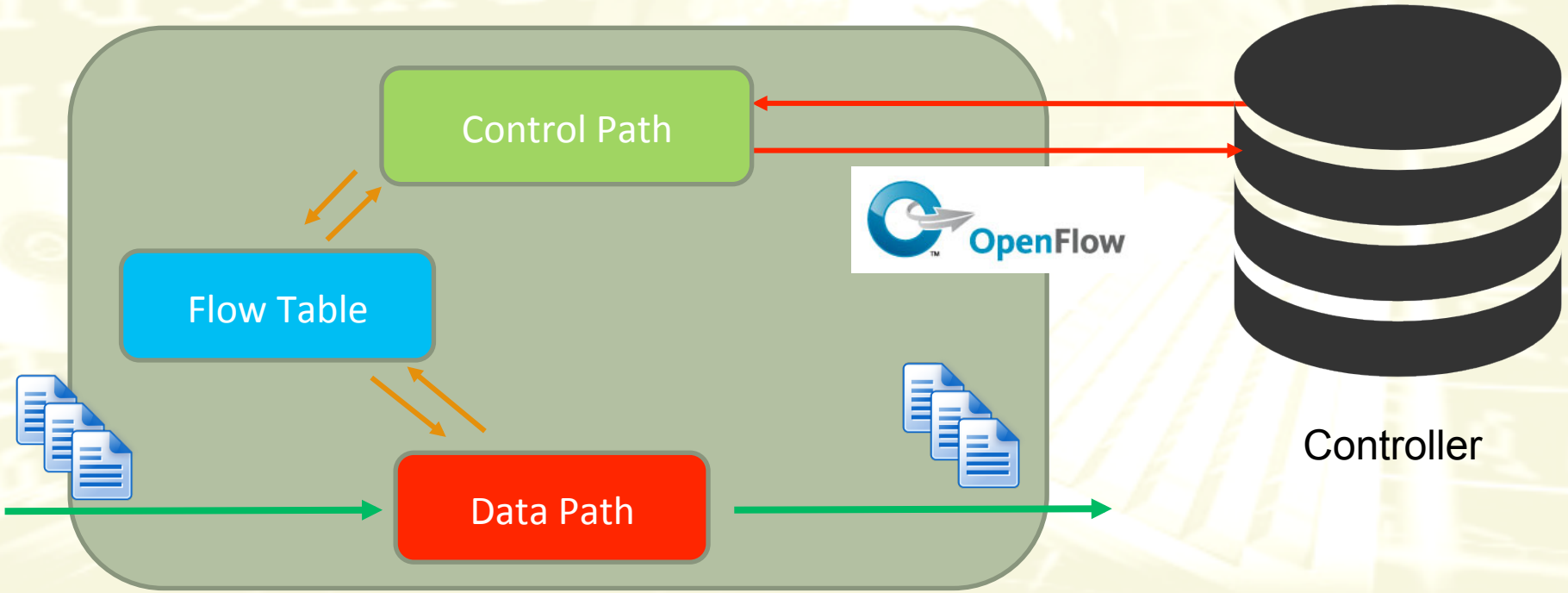
Open Flow

- Standard communication protocol that defines the interaction between switches and controllers
- It allows remote administration of packet forwarding table



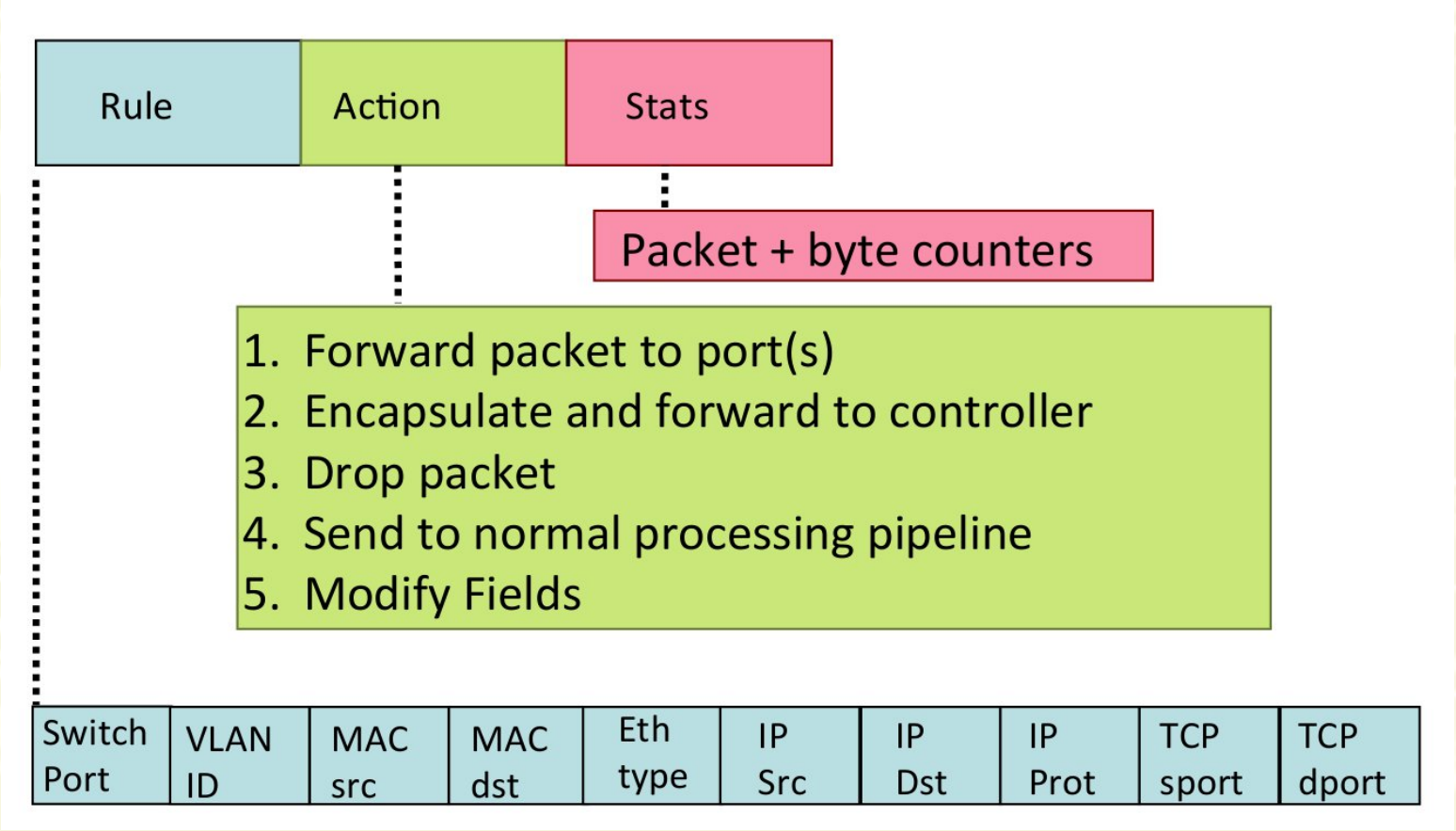
Open Flow Switch

- Packets are forwarded according to a simple ***flow table***
- Controller uses the Open Flow protocol to populate the Flow Table



Flow Table

- Set of rules (similar to cisco ACL) that determines the action to be performed for each packet



Rules Examples

- ***Cross-layer rules*** for packet classification
- Different ***functionalities*** can be implemented:
  - *Switching*
  - *Routing*
  - *Firewall*

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

Routing

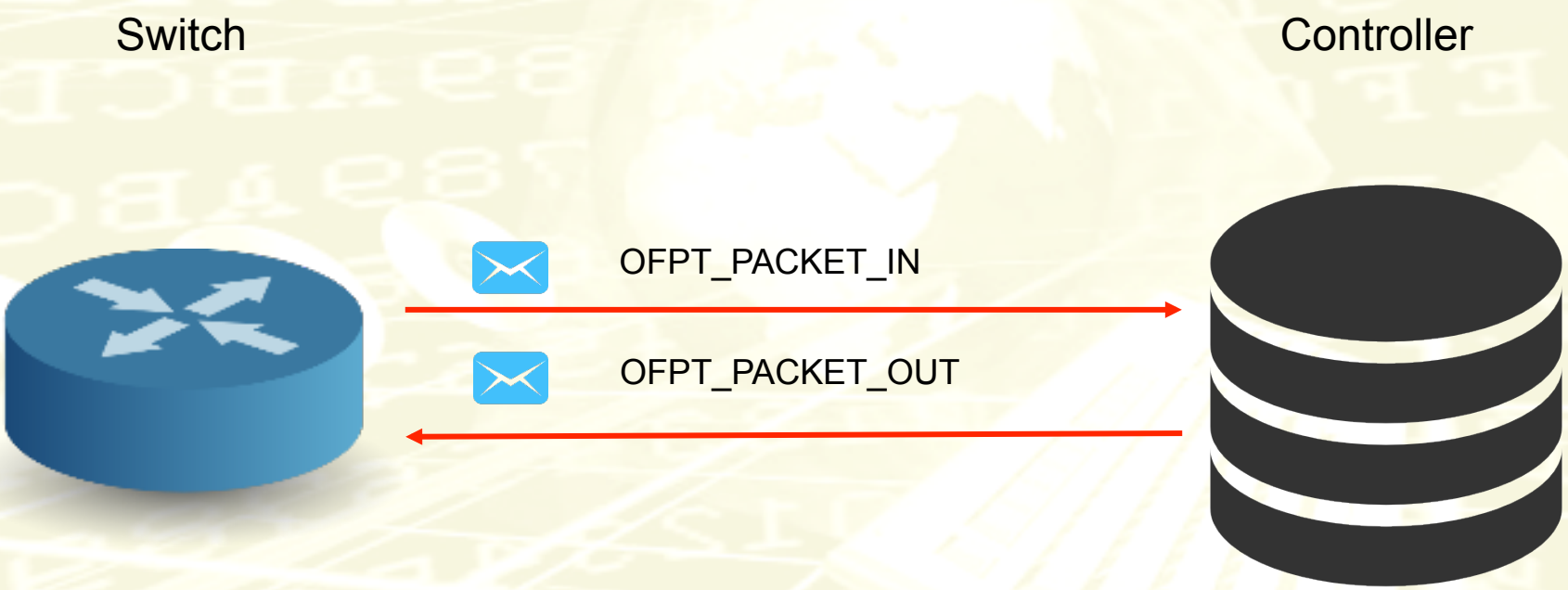
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

OF Messages - Startup

- At startup a set of startup messages is sent to allow the controller to discover the capabilities of the switch

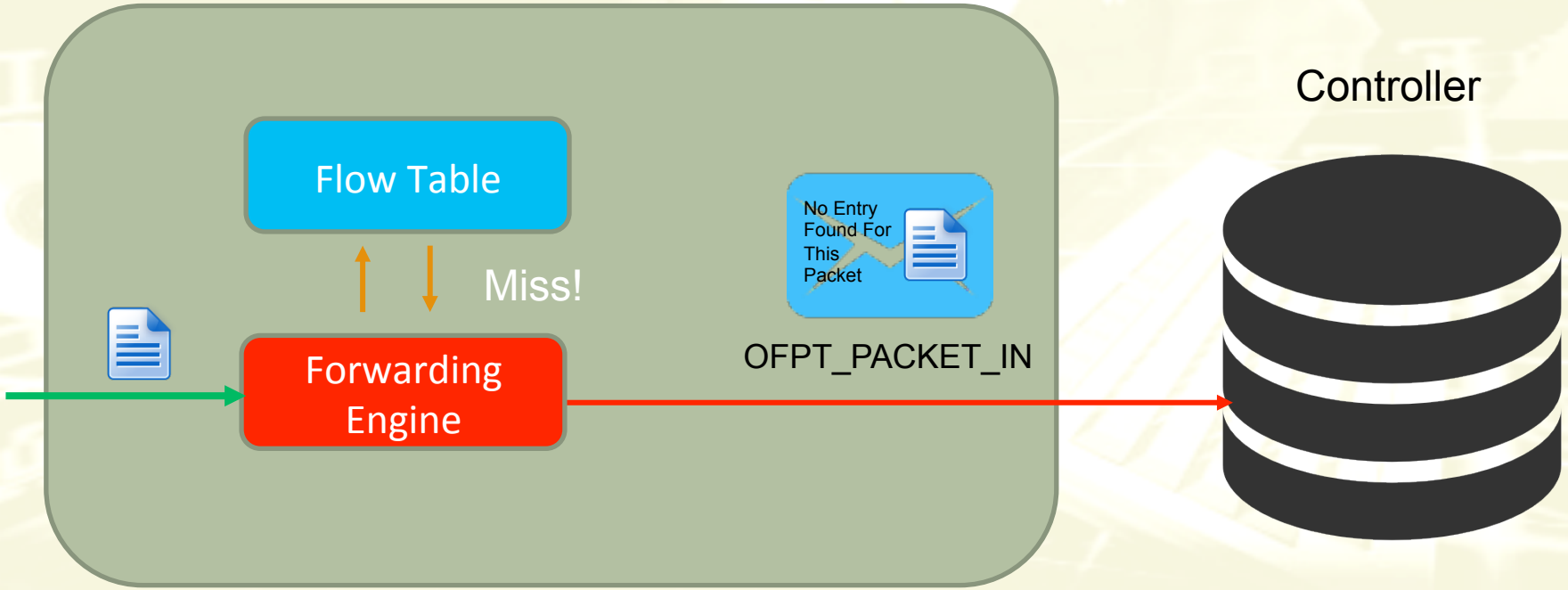
OF Messages – Normal Operations

- During normal operations switch and controller interacts with IN and OUT packets



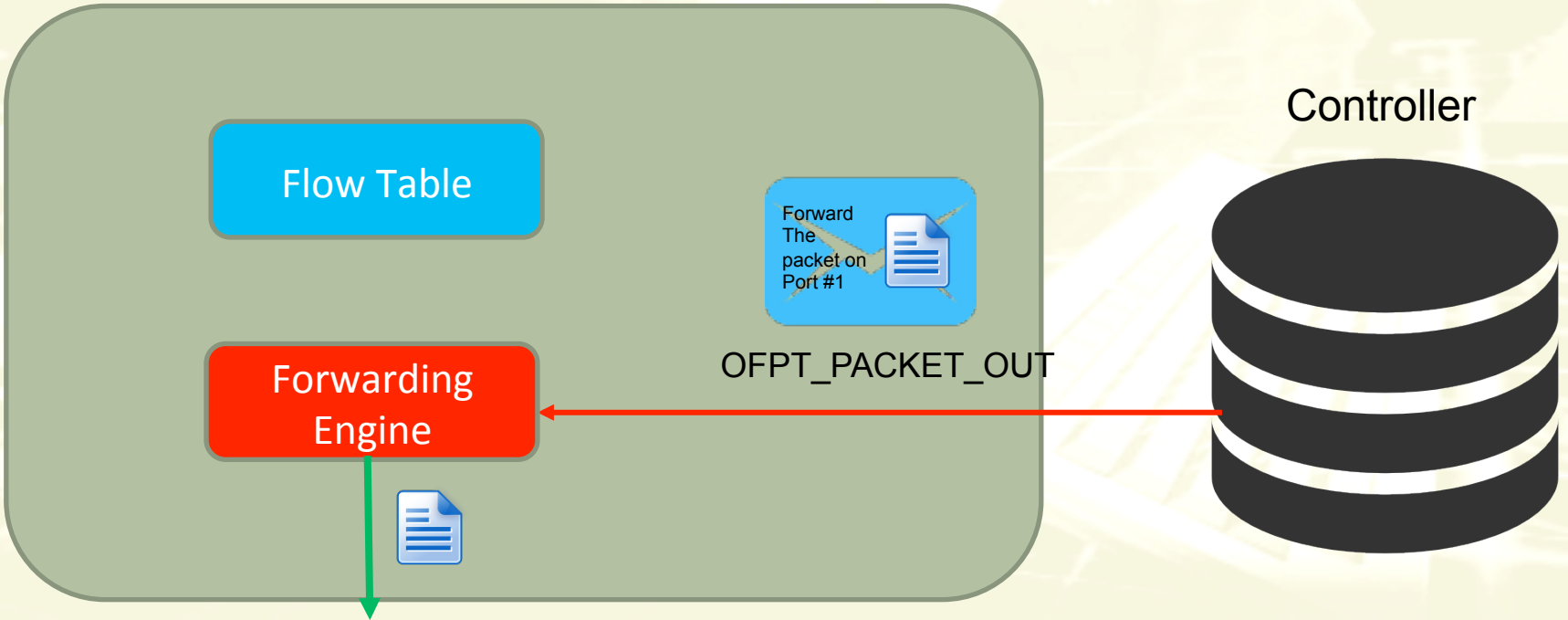
Switch Operations

- For each received packet the Flow Table is looked up
- If a match is ***found*** the action is executed, otherwise the ***packet is forwarded to the controller encapsulated into a Packet-In***



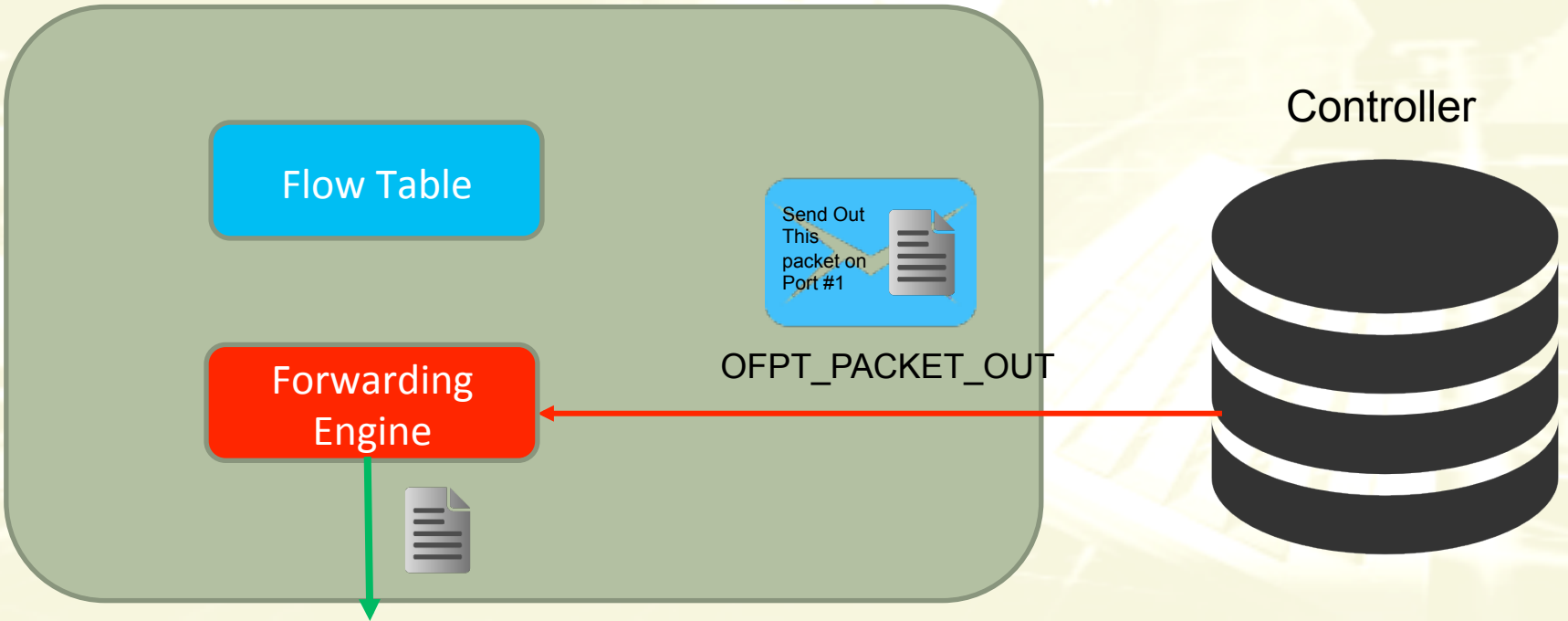
Switch Operations

- The controller can reply with a Packet-Out specifying the action to be performed (e.g. forward the packet on port #1)
- It will be executed only once (no modifications to the Flow Table)



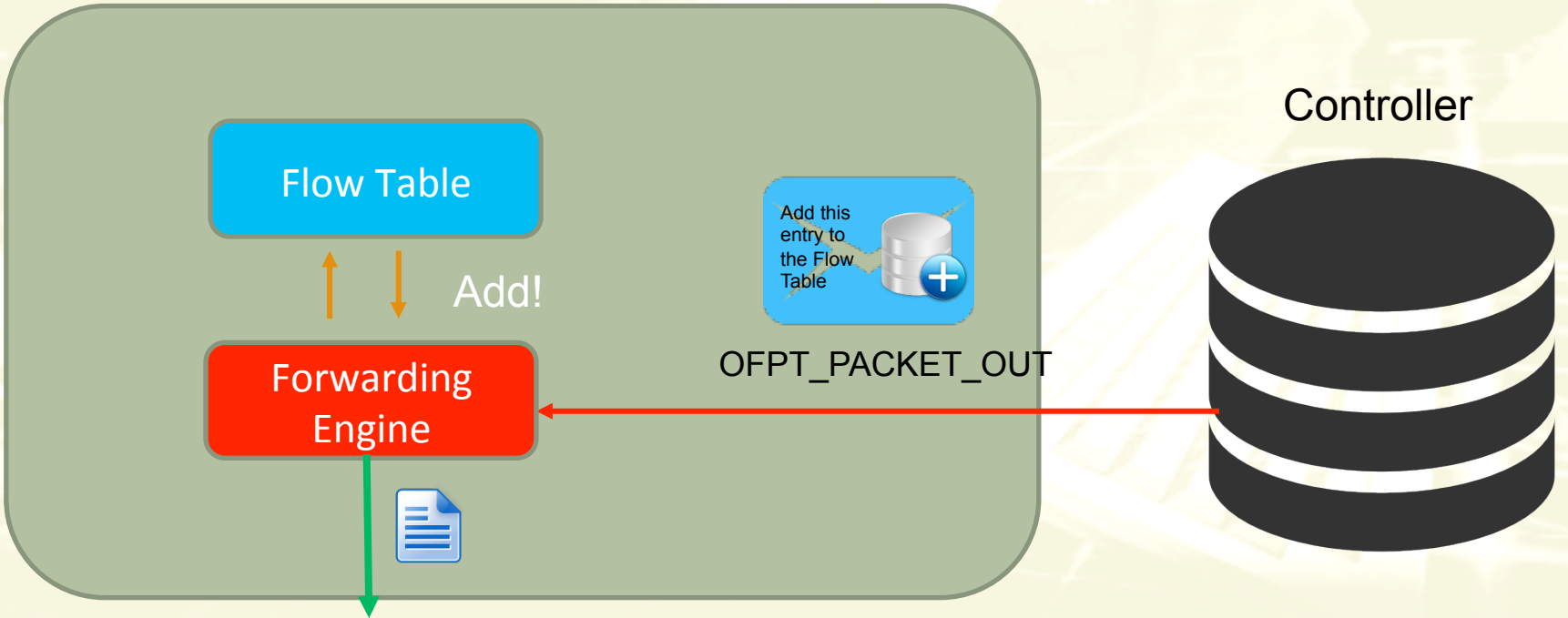
Switch Operations

- The controller can reply with a Packet-Out specifying a new packet to be sent out
- It will be executed only once (no modifications to the Flow Table)



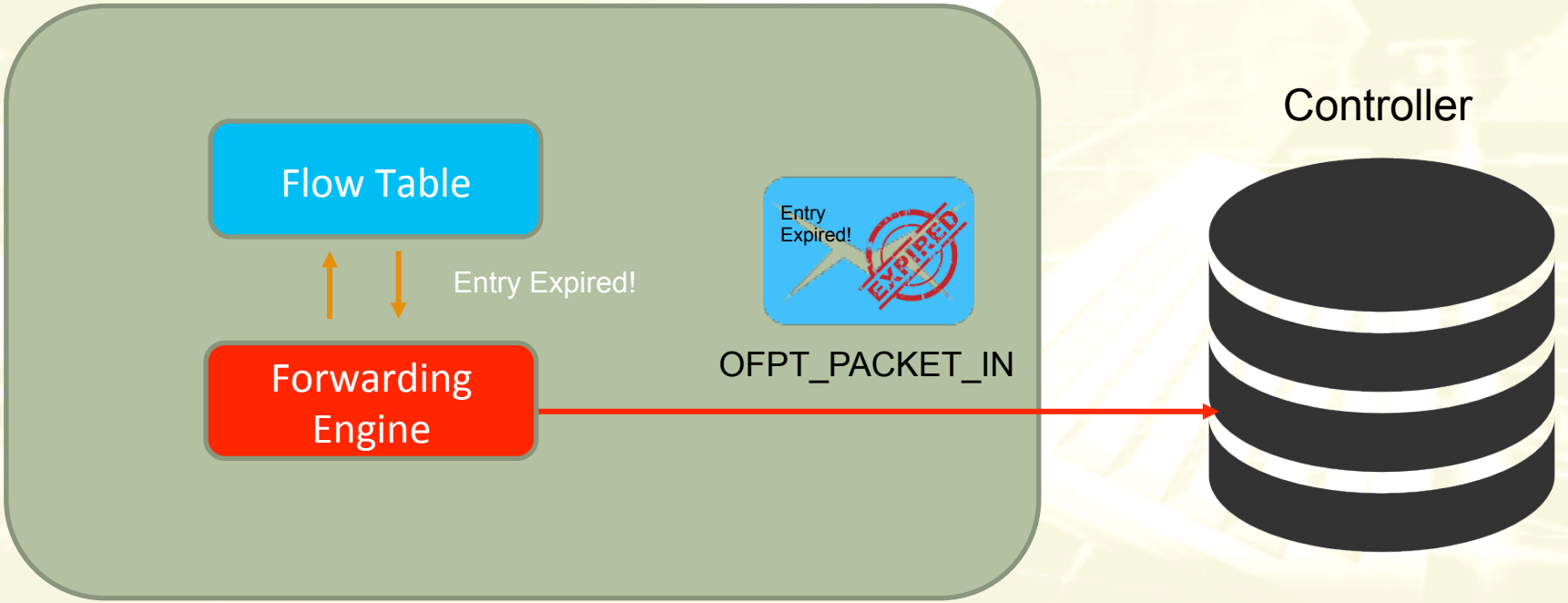
Switch Operations

- The controller can reply with a Packet-Out containing a ***Flow-Mod*** message that instructs the switch to add a new entry to its table
- The new entry will instruct the switch to perform a certain operation without contacting the controller
- The operation associated with the new entry is then executed



Entry Management

- Entries in the flow table expire
- As the entry is expired a Packet-In is sent to the Controller containing a ***Flow-Expired*** message
- Entries expire after a hard timeout (always) or after an idle timeout (if packets matching with the entry are not received)

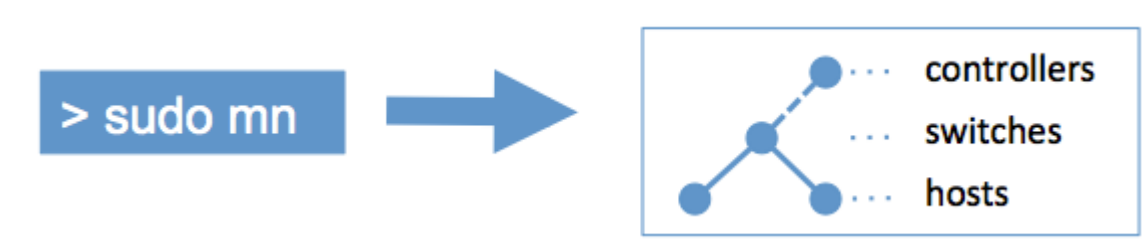


Mininet

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

Mininet

- Mininet is a virtual network emulator for testing of SDN deployments
- It allows in one program to emulate a network composed of OpenFlow switches and hosts which can generate traffic
- The network of OpenFlow switches can be connected to a real controller



Mininet

IP network subnet for simulated hosts

Type of the topology, topology with one single switch and three hosts

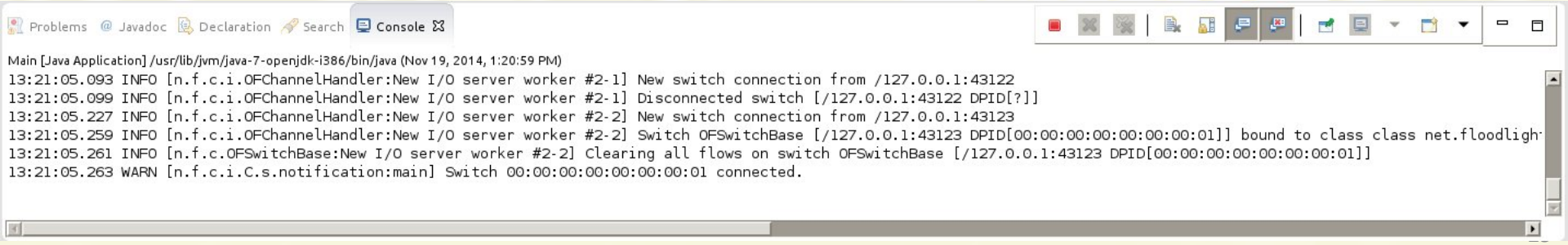
- Launch the simulator:

```
sudo mn --topo single,3
--mac --switch ovsk
--controller remote,ip=127.0.0.1,port=6653
--ipbase=10.0.0.0
```

We use a real OpenFlow controller which runs on localhost (port is 6653)

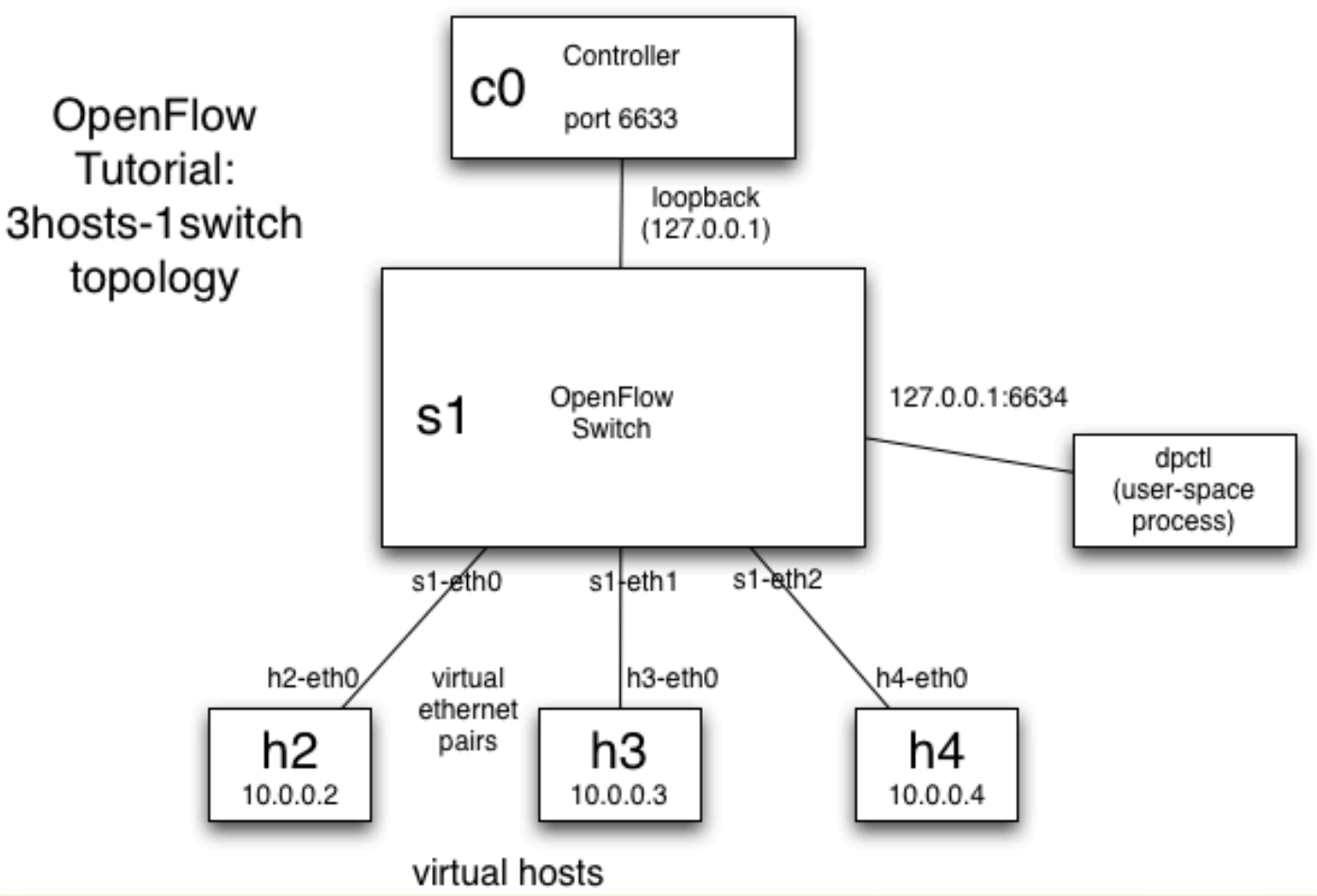
Controller

- If you launch the controller now you obtain the following message from the emulator:
  - `Unable to contact the remote controller at 127.0.0.1:6633`
- For our test we use ***Floodlight***, a framework to implement OpenFlow controller
- Run wireshark on the loopback interface to capture OpenFlow messages between controller and switch
- Compile and Execute the controller using the "*ant run*" command
- Or create a new ant configuration for running the floodlight directly into eclipse



Simulated Architecture

- The basic topology has the following architecture



Mininet Basics

- Run a program on a host
  - `host_name command`
    - `h1 ping 10.0.0.2`
    - `h1 ifconfig -a`
    - `h1 ifconfig h1-eth0 10.0.0.1`
- Open a separate terminal on a host
  - `xterm host_name`
    - `xterm h1`
    - `From the terminal for example you can run wireshark!`

Floodlight

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

Floodlight

- Floodlight is a java framework that allows the implementation of OpenFlow controllers
- It not only provides an implementation of the OpenFlow protocol but also an implementation of some basic operations

Floodlight

- Floodlight has a modular structure, each module implements one functionality
- Inbound packets are processed in cascade by each module, each module can interrupt the pipeline
- The modules included in the pipeline and their order are specified inside the file
  - `src/main/resources/floodlightdefault.properties`

```
floodlight.modules=\
net.floodlightcontroller.jython.JythonDebugInterface,\
net.floodlightcontroller.counter.CounterStore,\
net.floodlightcontroller.storage.memory.MemoryStorageSource,\
net.floodlightcontroller.core.internal.FloodlightProvider,\
net.floodlightcontroller.threadpool.ThreadPool,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
net.floodlightcontroller.devicemanager.internal.DefaultEntityClassifier,\
net.floodlightcontroller.staticflowentry.StaticFlowEntryPusher,\
net.floodlightcontroller.firewall.Firewall,\
net.floodlightcontroller.hub.Hub,\
net.floodlightcontroller.forwarding.Forwarding,\
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
net.floodlightcontroller.topology.TopologyManager,\
net.floodlightcontroller.flowcache.FlowReconcileManager,\
net.floodlightcontroller.debugcounter.DebugCounter,\
```

New 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:

Add Class In Eclipse

1. Expand the "floodlight" item in the Package Explorer and find the "src/main/java" folder.
2. Right-click on the "src/main/java" folder and choose "New/Class".
3. Enter "net.floodlightcontroller.mactracker" in the "Package" box.
4. Enter "MACTracker" in the "Name" box.
5. Next to the "Interfaces" box, choose "Add...".
6. Add the "IOFMessageListener" and the "IFloodlightModule", click "OK".
7. Click "Finish" in the dialog.

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

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

Handle Packet-In Messages

- Each module that wants to process Packet-In messages needs to register and define a ***receive*** function

```
// 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);
}

// 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
```

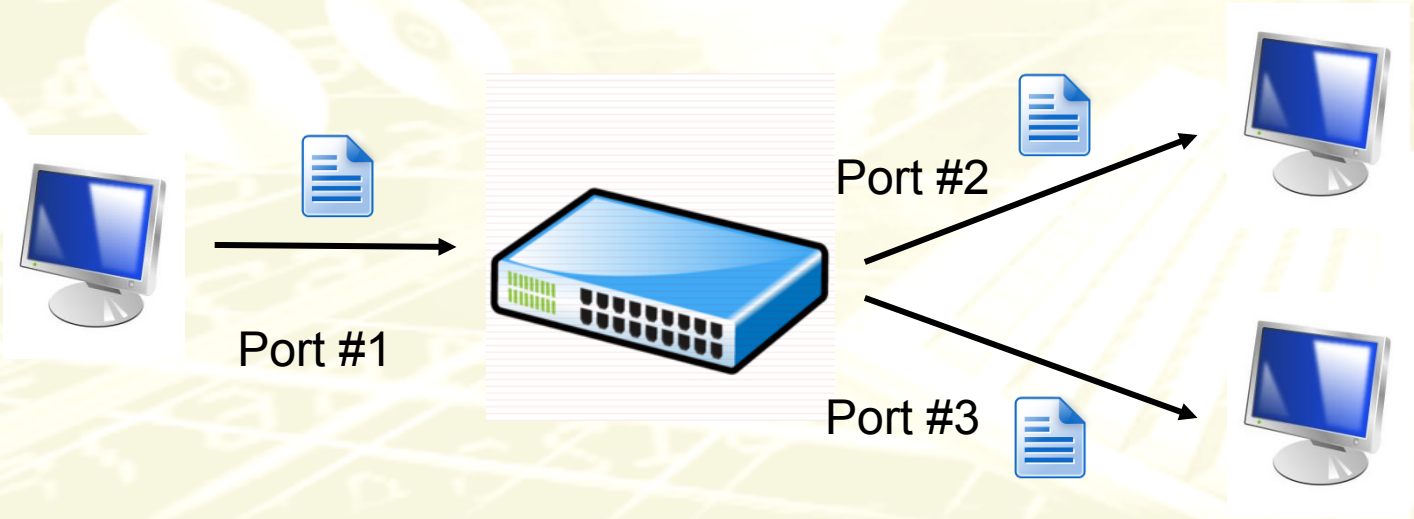
Add the module into the pipeline

```
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
- Do not install any Flow table entry but just send commands every time



Hub

- Create a Packet-Out

```
// 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());
pob.setInPort(pi.getInPort());

// 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

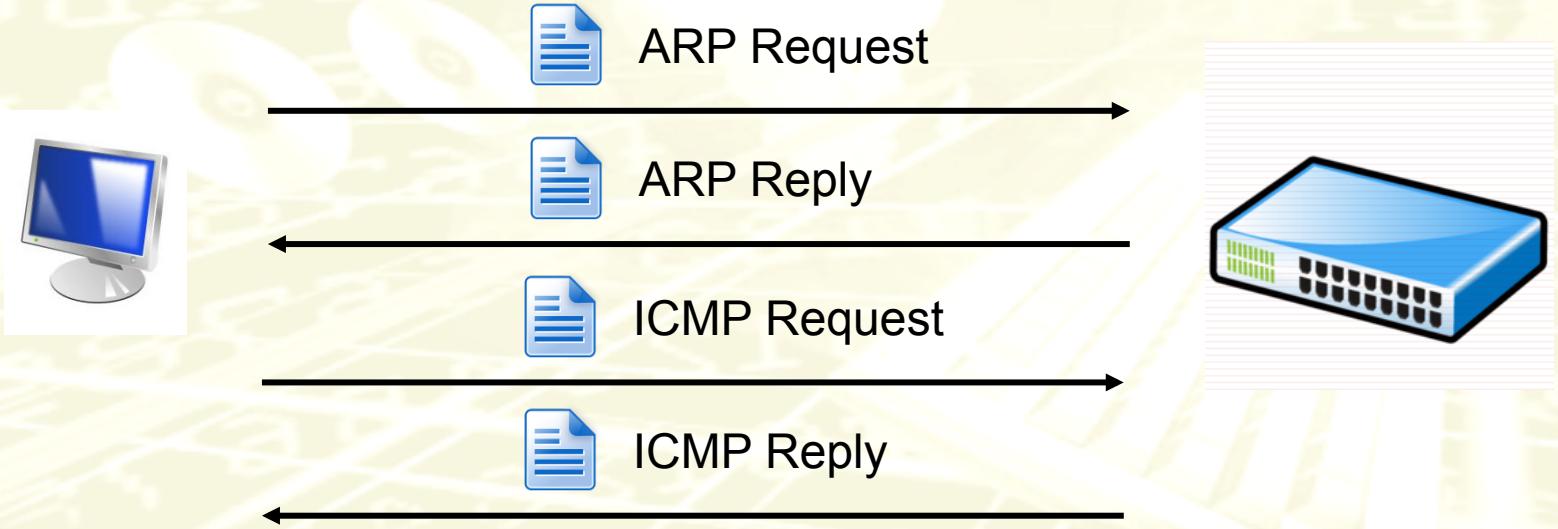
- Packet waiting for being transmitted can be sent to the Controller encapsulated in the Packet-In message or can be buffered on the switch

```
// 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(po, cntx);

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

Virtual address

- Program the controller in order to reply icmp ping request on a certain IP address (that does not exist)



Mininet

- Set each host to send out traffic for external networks on a given NIC card
  - h1 route add -net 0.0.0.0/32 dev h1-eth0

Virtual address

- Receiver function:

```
// Cast packet
OFPacketIn pi = (OFPacketIn) msg;
// Dissect Packet included in Packet-In

IPacket pkt = eth.getPayload();

if (eth.isBroadcast() || eth.isMulticast()) {
            if (pkt instanceof ARP) {
                        ...
            }
}
// We only care about packets which are sent to the logical IP address
IPv4 ip_pkt = (IPv4) pkt;

if(ip_pkt.getDestinationAddress().compareTo(SERVER_IP) == 0){
            // Allow the next module to also process this OpenFlow message
            return Command.CONTINUE;
}

// Do not continue processing this OpenFlow message
return Command.STOP;
```

Virtual address - handleARPRequest

- Create ARP reply

```
// Double check that the payload is ARP
Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
            IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
if (! (eth.getPayload() instanceof ARP))
            return;

// Cast the ARP request
ARP arpRequest = (ARP) eth.getPayload();
// Generate ARP reply
IPacket arpReply = new Ethernet()
            .setSourceMACAddress(SERVER_MAC)
            .setDestinationMACAddress(eth.getSourceMACAddress())
            .setEtherType(EthType.ARP)
            .setPriorityCode(eth.getPriorityCode())
            .setPayload(
                        new ARP()
                        .setHardwareType(ARP.HW_TYPE_ETHERNET)
                        .setProtocolType(ARP.PROTO_TYPE_IP)
                        .setHardwareAddressLength((byte) 6)
                        .setProtocolAddressLength((byte) 4)
                        .setOpCode(ARP.OP_REPLY)
                        .setSenderHardwareAddress(SERVER_MAC) // Set my MAC address
                        .setSenderProtocolAddress(SERVER_IP) // Set my IP address
                        .setTargetHardwareAddress(arpRequest.getSenderHardwareAddress())
                        .setTargetProtocolAddress(arpRequest.getSenderProtocolAddress())
            );
```

Virtual address - handleARPRequest

- Create Packet-Out
- Create list of actions
- Set the ARP reply as payload of Packet-Out
- Send it out

```
// Initialize a packet out
OFPacketOut.Builder pob = sw.getOFFactory().buildPacketOut();
pob.setBufferId(OFBufferId.NO_BUFFER);
pob.setInPort(OFPort.ANY);

// Set the output action
OFActionOutput.Builder actionBuilder = sw.getOFFactory().actions().buildOutput();
actionBuilder.setPort(pi.getInPort());
pob.setActions(Collections.singletonList((OFAction) actionBuilder.build()));

// Set the ARP reply as packet data
byte[] packetData = arpReply.serialize();
pob.setData(packetData);

// Send packet
sw.write(po, null);
```

Virtual address - handleIPPacket

- Check Packet-In
- Create the ICMP reply

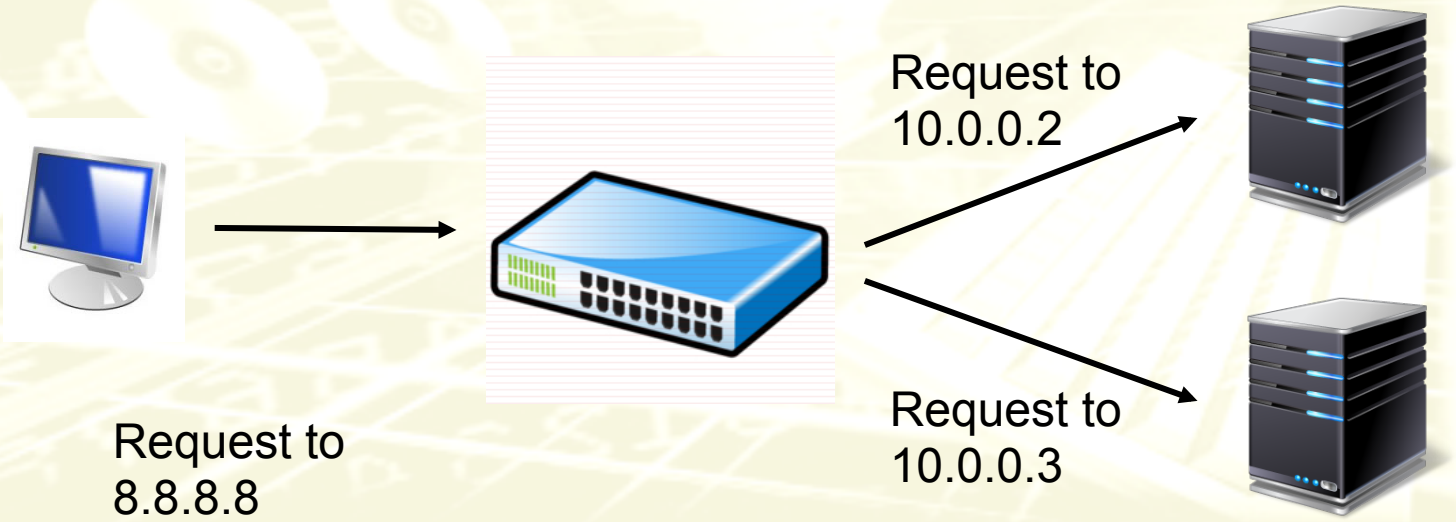
```
// Cast the IP packet
IPv4 ipv4 = (IPv4) eth.getPayload();

// Check that the IP is actually an ICMP request
if (! (ipv4.getPayload() instanceof ICMP))
            return;

// Cast to ICMP packet
ICMP icmpRequest = (ICMP) ipv4.getPayload();
// Generate ICMP reply
IPacket icmpReply = new Ethernet()
            .setSourceMACAddress(SERVER_MAC)
            .setDestinationMACAddress(eth.getSourceMACAddress())
            .setEtherType(Ethernet.TYPE_IPv4                    )
            .setPriorityCode(eth.getPriorityCode())
            .setPayload(
                        new IPv4()
                        .setProtocol(IpProtocol.ICMP)
                        .setDestinationAddress(ipv4.getSourceAddress())
                        .setSourceAddress(SERVER_IP)
                        .setTtl((byte)64)
                        .setProtocol((byte)4)
                        // Set the same payload included in the request
                        .setPayload(
                                    new ICMP()
                                    .setIcmpType(ICMP.ECHO_REPLY)
                                    .setIcmpCode(icmpRequest.getIcmpCode())
                                    .setPayload(icmpRequest.getPayload())
                        )
            );
```

Load Balancer

- Program the controller in order to act as a load balancer among a pool of servers
- A virtual server corresponds to a public IP address (e.g. 8.8.8.8) not assigned to any physical server
- The switch dispatches request directed to the IP of the virtual server among a pool of servers that reply to the request transparently changing the allocation every 20 seconds in order to balance the load
- The ***switch has to reply to ARP requests*** and c***hange destination and source addresses of IP packets (like a NAT)*** in order to hide which server actually process the request



Load Balancer

```
// Create a flow table modification message to add a rule
        OFFlowMod.Builder fmb = sw.getOFFactory().buildFlowAdd();

        fmb.setIdleTimeout(IDLE_TIMEOUT);
        fmb.setHardTimeout(HARD_TIMEOUT);
        fmb.setBufferId(OFBufferId.NO_BUFFER);
        fmb.setOutPort(OFPort.ANY);
        fmb.setBufferId(pi.getBufferId());
        fmb.setCookie(U64.of(0));
        fmb.setPriority(FlowModUtils.PRIORITY_MAX);

// Create match
        Match.Builder mbRev = sw.getOFFactory().buildMatch();
        mb.setExact(MatchField.ETH_TYPE, EthType.IPv4)
        .setExact(MatchField.IP_PROTO, IpProtocol.IPv4)
        .setExact(MatchField.IPV4_SRC, IPv4Address.of(SERVERS_IP[last_server]))
        .setExact(MatchField.ETH_SRC, MacAddress.of(SERVERS_MAC[last_server]));

// Create Actions
        ArrayList<OFAction> actionsRev = new ArrayList<OFAction>();
        actions.add(sw.getOFFactory().actions().setDlSrc(LOAD_BALANCER_MAC));
        actions.add(sw.getOFFactory().actions().setNwSrc(LOAD_BALANCER_IP));
        actions.add(sw.getOFFactory().actions().output(OFPort.of(1), Integer.MAX_VALUE ));

// Send out mod message
        fmb.setActions(actions);
        fmb.setMatch(mb.build());

        sw.write(fmb.build());
```

References

- *Google, SDN experience:*
  http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf
- *Mininet, walkthrough:*
  http://mininet.org/walkthrough/
- *Floodlight, How to write a module:*
  http://docs.projectfloodlight.org/display/floodlightcontroller/How+to+Write+a+Module
- *OpenFlow tutorial:*
  http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

References

- *Floodlight advanced tutorial:* http://www.openflowhub.org/display/floodlightcontroller/Advanced+Tutorial/
- *Mininet, how to apply dynamic topology changes:* http://www.kiranvemuri.info/computer-networks/sdn/mininet-advanced-users-dynamic-topology-changes/