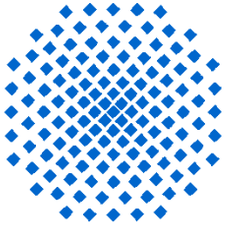


Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Tutorial: Software-defined Networking

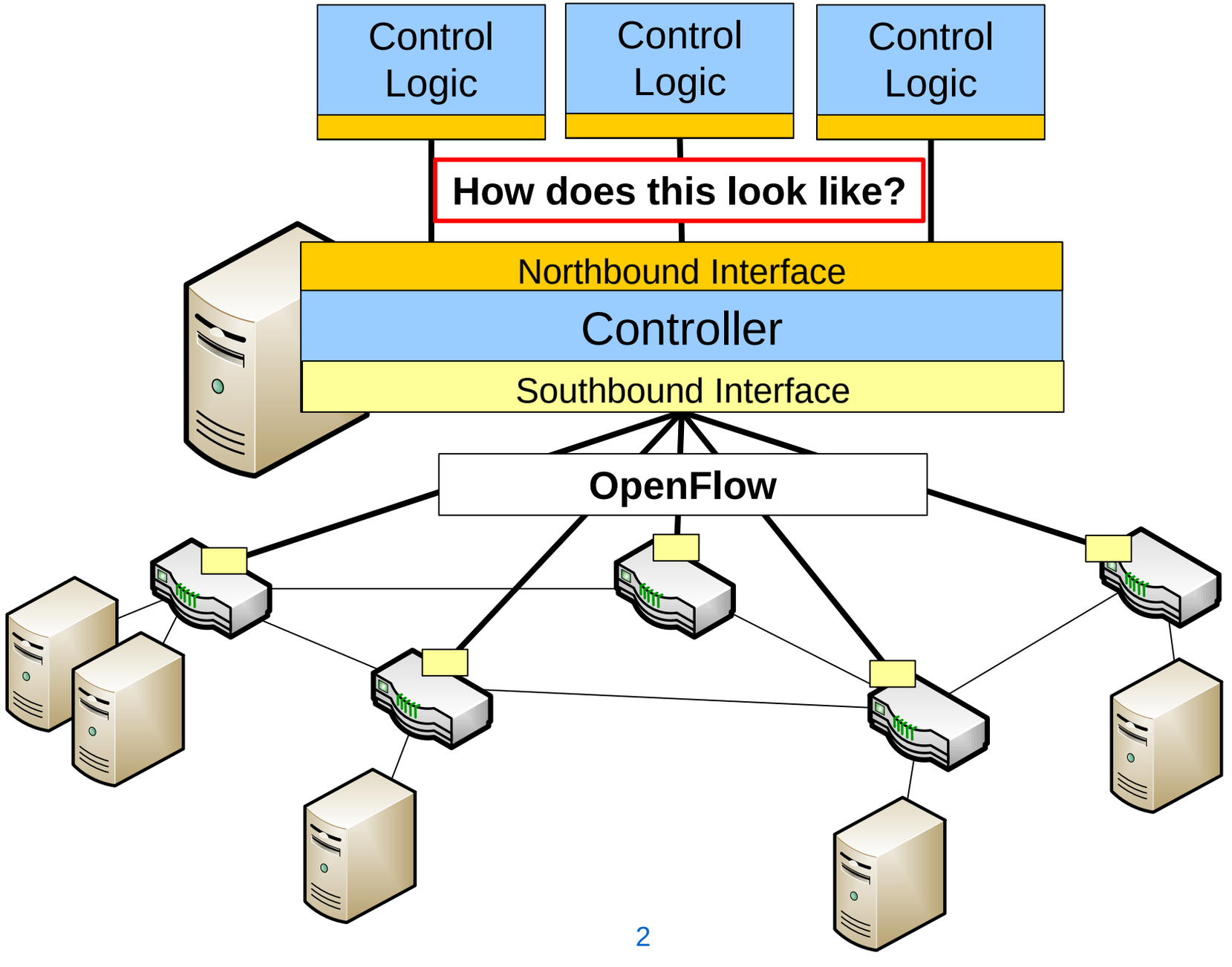
Part 5: Northbound Interfaces

Java API for Floodlight Module Applications

Frank Dürr

Acknowledgements / other contributors:
Sukanya Bhowmik, Ben Carabelli, Thomas Kohler

Architecture of an SDN System

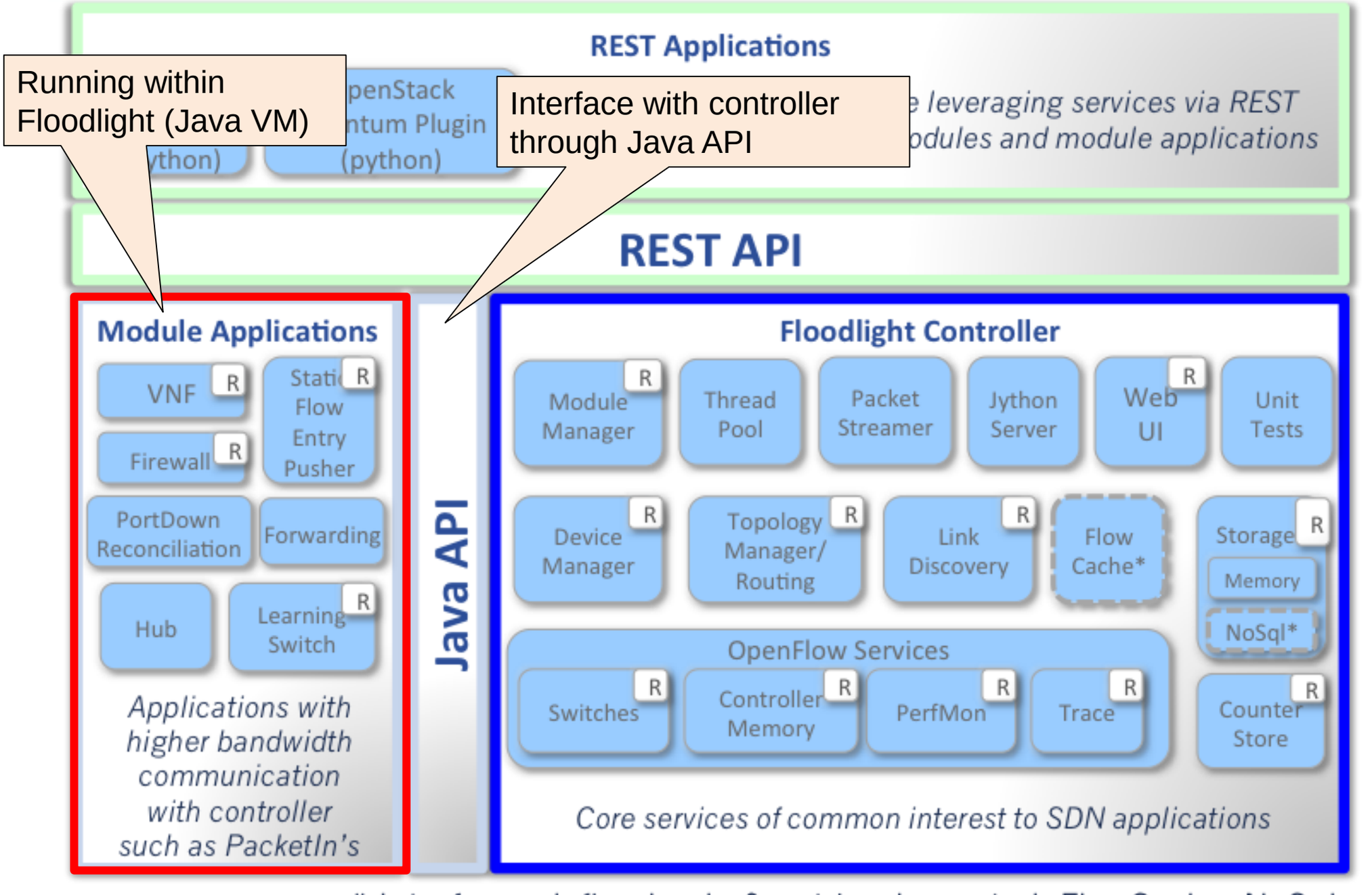


Examples of Northbound Interfaces

- RESTful Interface ➔ Tutorial, Part 4
  - HTTP only supports request/response invocation of controller by control app
  - Only proactive routing
  - No packet-in events from switch to controller

- Java Interface (Floodlight modules) ➔ next
  - Supports reactive routing
  - Packet-in events trigger callback functions in Java control app

Extending Floodlight Controller with Modules



* Interfaces defined only & not implemented: FlowCache, NoSql

Floodlight Modules – Features

- Sending OpenFlow messages to switch
  - Flow programming
    - Proactive *and* reactive (see below)
  - Pushing packets from controller out of a switch
- Receiving asynchronous OpenFlow messages (events)
  - E.g., packet-in events (reactive flow programming)

Creating a Floodlight Module

- Create Java class implementing interface `IFloodlightModule`
- Initialization of a module (callbacks):
  1. `getModuleServices()`:
     What services does this module provide?
  2. `getModuleDependencies()`:
     List service dependencies

  If module provides a service, e.g., via REST

  3. `init()`:
     Internal initializations (don't touch other modules)
  4. `startUp()`:
     External initializations (do touch other modules)

Reactive Flow Programming (1)

Set up module name and dependencies:

```
public Collection<Class<? extends IFloodlightService>>
    getModuleDependencies() {
        Collection<Class<? extends IFloodlightService>> l =
        new ArrayList<Class<? extends IFloodlightService>>();
        l.add(IFloodlightProviderService.class);
        return l;
}

public String getName() {
        return this.getClass().getSimpleName();
}
```

Service required to register for packet-in events

Reactive Flow Programming (2)

Register your module for receiving packet-in events:

```
public void init(FloodlightModuleContext context)
        throws FloodlightModuleException {
        floodlightProvider = context.getServiceImpl(
                IFloodlightProviderService.class);
}

public void startUp(FloodlightModuleContext context)
        throws FloodlightModuleException {
        floodlightProvider.addOFMessageListener(
                OFType.PACKET_IN,
                this);
}
```

Reactive Flow Programming (3)

Implement interface `IOFMessageListener` in your module class

Callback method:

```
public Command receive(IOFSwitch sw, OFMessage
msg, FloodlightContext cntx);
```

- `sw:` Switch (object) from which message was received
  - Can be used to send OpenFlow messages to switch
- `msg:` OpenFlow Message from switch to controller
- `cntx:` Floodlight context
- Return: `Command.STOP` to “consume” message; `Command.CONTINUE` to let next listener process msg

Inspecting Received Packet (1)

- Get packet from OpenFlow message:

```
Ethernet eth = IFloodlightProviderService.bcStore.get(
        cntx,
        IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
```

- Determine layer 3 packet type (frame payload):

`eth.getEtherType()` (returns, e.g., `Ethernet.TYPE_IPv4`)

- Get layer 3 packet from layer 2 frame payload

```
IPacket pkt = (IPacket) eth.getPayload();
```

Inspecting Received Packet (2)

- Cast to right packet type (can use `instanceof` type check):

```
IPv4 ipv4Pkt = (IPv4) pkt;
```

- Inspect header packet header fields:

```
int destAddr = ipv4Pkt.getSourceAddress();
```

Setting a Flow Table Entry (1)

- Every OpenFlow concept (OFObjects such as Match, OFAction, OFMessage, etc.) is constructed by means of a builder
- All builders are exposed through the OFFactory interface
- Each OpenFlow version has a specific factory

- Get a reference to the OpenFlow factory

```
OFFactory myFactory =
        OFFactories.getOFFactory(openflow_version);
```

e.g., `OFVersion.OF_13`

- The following examples we provide are based on Openflow 1.3

Setting a Flow Table Entry (2)

- Create match, e.g.

```
Match myMatch = myFactory.buildMatch()
  .setExact(MatchField.IN_PORT, OFPort.of(1))
  .setExact(MatchField.ETH_TYPE, EthType.IPv4)
  .setExact(MatchField.IPV4_SRC,IPv4Address.of(""))
  .setExact(MatchField.IP_PROTO, IpProtocol.TCP)
  .setExact(MatchField.TCP_DST, TransportPort.of(80))
  .build();
```

Setting a Flow Table Entry (3)

- Create actions, e.g.

```
ArrayList<OFAction> myActionList = new ArrayList<OFAction>();
OFActions actions = myFactory.actions();

OFActionOutput output = actions.buildOutput()
       .setMaxLen(0xFFffFFff)
       .setPort(OFPort.of(1))
       .build();

 myActionList.add(output);
```

Setting a Flow Table Entry (3)

- Create a flow-mod message

```
OFFlowAdd flowAdd = myFactory.buildFlowAdd()
        .setPriority(32768)
        .setMatch(myMatch)
        ...
        .setActions(myActionList)
        .build();
```

- Send flow-mod message
  - Via switch object sw (e.g., passed as parameter of receive callback, see previous slides)

```
sw.write(flowAdd);
```

Pushing a Packet to and out of a Switch (1)

- Create each layer of the data to be injected

```
Ethernet layer2 = new Ethernet();
layer2.setSourceMACAddress(MacAddress.of(""))
layer2.setDestinationMACAddress(MacAddress.of(""));
layer2.setEtherType(EthType.IPv4);

IPv4 layer3 = new IPv4();
layer3.setSourceAddress(IPv4Address.of(""));
layer3.setDestinationAddress(IPv4Address.of(""));
layer3.setTtl((byte) 64);
layer3.setProtocol(IpProtocol.UDP);

UDP layer4 = new UDP();
layer4.setSourcePort(TransportPort.of(port));
layer4.setDestinationPort(TransportPort.of(port));

Data data = new Data();
data.setData(new byte[1000]);
```

Layer 2

Layer 3

Layer 4

Data for Layer 4 payload

Pushing a Packet to and out of a Switch (1)

- Set the payload of each layer as the next highest layer

```
layer2.setPayload(layer3);
layer3.setPayload(layer4);
layer4.setPayload(data);
```

- Serialize

```
byte[] serializedData = layer2.serialize();
```

Pushing a Packet to and out of a Switch (2)

```
OFPacketOut po = sw.getOFFactory().buildPacketOut()
        .setData(serializedData)
```

- Set the payload of the packet-out

```
        .setActions(Collections.singletonList((OFAction)
              sw.getOFFactory().actions()
        .output(OFPort.FLOOD, 0xffffffff)))
```

- Specify the output port through a list actions

```
        .setInPort(OFPort.CONTROLLER)
```

- Set the input port

```
        .build();
```

- Build the packet-out object

Pushing a Packet to and out of a Switch (3)

- Send packet-out message to switch via switch object `sw`

```
sw.write(po);
```

More Information

- Floodlight tutorial:

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

Summary

Discussed another northbound interface:

- Java API (Module interface) of Floodlight
  - Full power of OpenFlow
    - Proactive & reactive flow programming
    - Pushing packets

More interfaces possible; no standard

Questions?