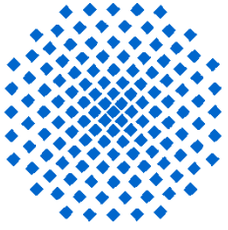


Universität Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Tutorial: Software-defined Networking

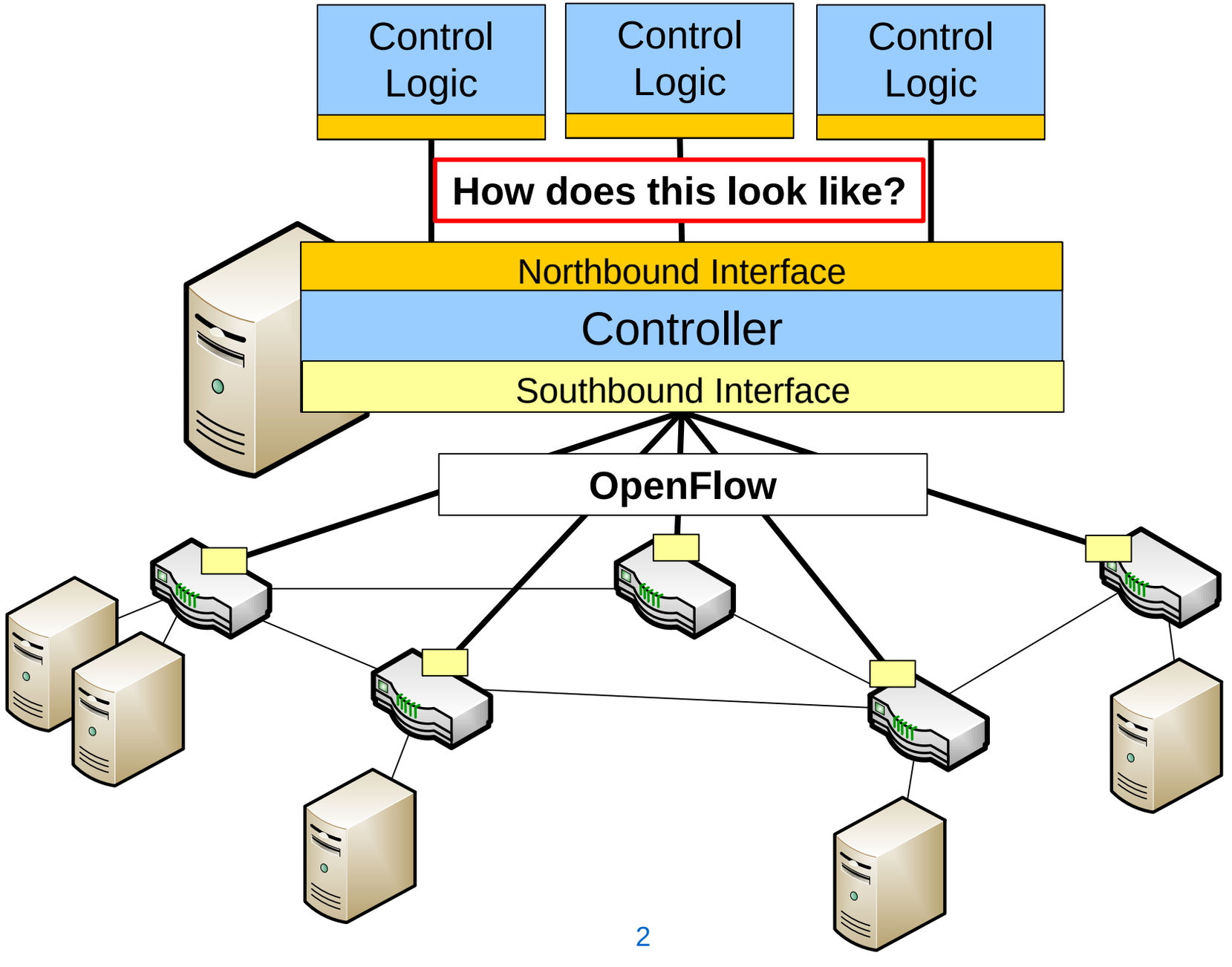
Part 4: Northbound Interfaces

RESTful Interfaces

Frank Dürr

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

Architecture of an SDN System



The Northbound Interface

- Interface between controller and application-specific control logic
  - "API" to program the network
- Controller …
  - … exposes information about the network to the application
    - Network topology
    - Traffic statistics
  - … translates application requests to OpenFlow requests
  - … sends OpenFlow events to application
    - `packet_in` events for reactive routing

No Single Standard Northbound Interface

- No standard for the northbound interface from the ONF
- Every controller can define its own northbound interfaces
  - Or even multiple as we will see
- Different requirements for reactive and proactive flow programming
  - Reactive programming requires event functionality for packet-in events

In this part, we only consider one exemplary **RESTful** interface for **proactive flow configuration** as implemented by the **Floodlight** SDN controller

REST Interfaces

- REST interfaces are popular for web services
  - Many programmers already know web technologies
  - Can expose network configuration as a web service
- Based on common web technologies
  - HTTP
    - No problems with firewalls
  - XML, JSON
    - Simple and intuitive markup languages
- Drawback: events not supported
  - HTTP based on request/response paradigm
  - **Restricted to proactive routing!**

*RE*presentational *S*tate *T*ransfer (REST)

REST is resource-oriented

- In contrast to service-oriented architectures (XML-RPC, WS-*)
- Unique identifiers to identify resources (URI)
  - Employees of a company: http://foo.bar/employees/
    One employee: http://foo.bar/employees/170974d
  - Relevant SDN resources: flows, topology, etc.
- Client/Server architecture
  - Server manages resources (➔ controller implements server)
  - Client manipulates resources (➔ control logic/applications)
- Uniform interface: well-defined methods to manipulate resources

*RE*presentational *S*tate *T*ransfer (REST)

HTTP methods to manipulate resources:

- GET: retrieve a resource
- POST: create resource
- PUT: update resource
- DELETE: remove resource
- HEAD: retrieve meta data on resource
- OPTIONS: methods that can be executed on resource

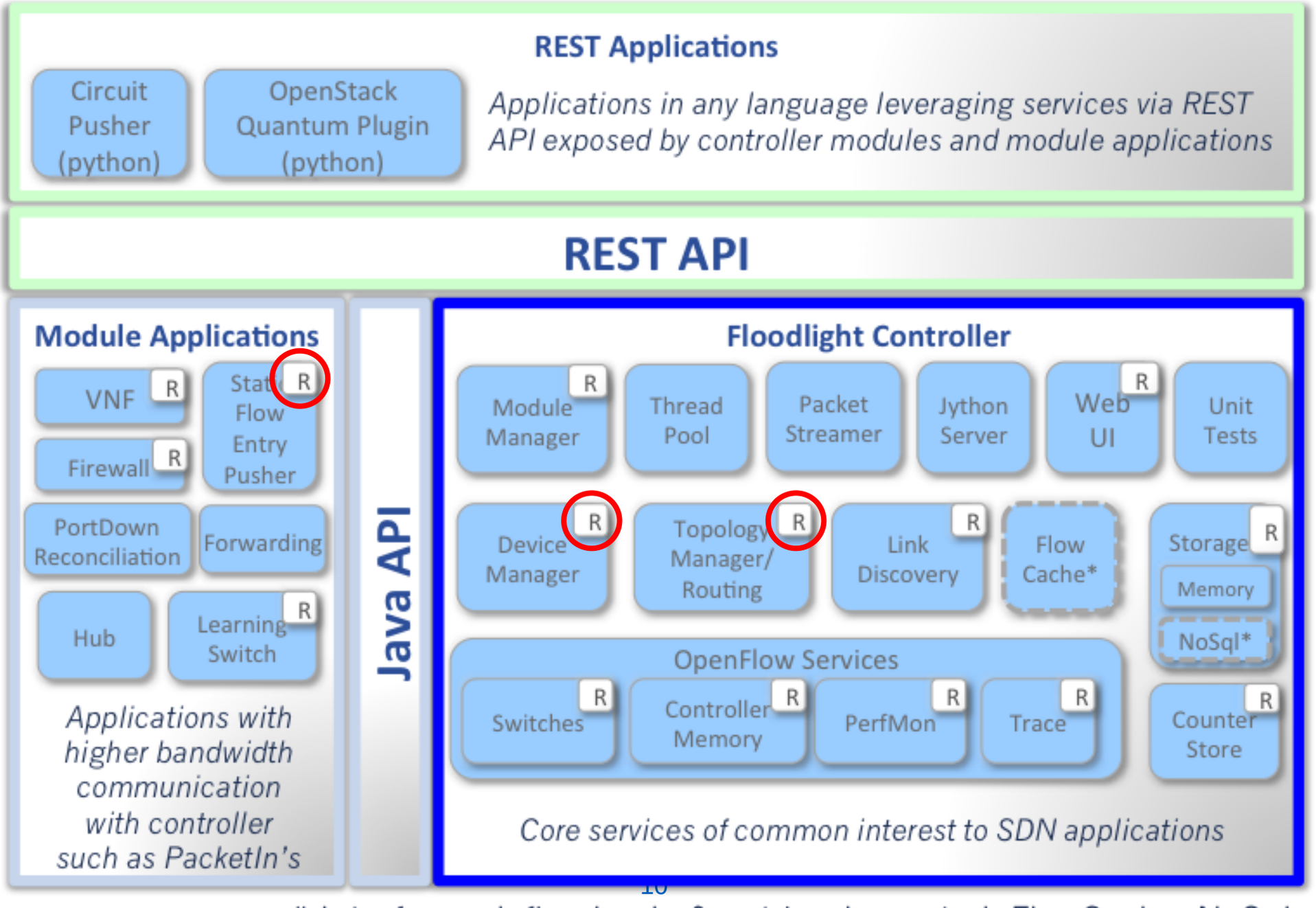
*RE*presentational *S*tate *T*ransfer (REST)

Stateless protocol between client and server

- Exchange of resource **representations** between client and server
  - XML,
  - JSON,
  - binary format,
  - etc.

Let’s look at a concrete example:
REST interface of Floodlight SDN controller

Floodlight REST APIs



* Interfaces defined only & not implemented: FlowCache, NoSql

Static Flow Pusher API: Adding a Flow

- POST request to URI
  `http://<controller_ip>:8080/wm/staticflowpusher/json`
- JSON payload:

```
{
    "switch": "00:00:00:00:00:00:00:01",
    "name":"flow-mod-1",
    "priority":"32768",
    "in_port":"1",
    "active":"true",
    "actions":"output=2"
}
```

- Floodlight assigns each switch a unique ID (**data path id**; DPID)
- Used to find the switch where to send the OpenFlow request
- Not part of OpenFlow!

- Flow id used by Floodlight
- Not part of OpenFlow!

- OpenFlow flow table entry attributes

Important Attributes of a Flow Table Entry

switch	<switch DPID>	ID of the switch (data path) that this rule should be added to xx:xx:xx:xx:xx:xx:xx:xx
name	<string>	Name of the flow entry, this is the primary key, **MUST be unique**
actions	<key>=<value>	See table of actions below Specify multiple actions using a comma-separated list Specifying no actions will cause the packets to be dropped
priority	<number>	default is 32767 maximum value is 32767
in_port	<number>	switch port on which the packet is received Can be hexadecimal (with leading 0x) or decimal
eth_src	<mac address>	xx:xx:xx:xx:xx:xx
eth_dst	<mac address>	xx:xx:xx:xx:xx:xx
eth_vlan_vid	<number>	VLAN ID, can be hexadecimal (with leading 0x) or decimal **Must include 'present' bit (bit 12) in match!** E.g., to match on tag 0x0064, use the match 0x1064
eth_vlan_pcp	<number>	Priority Code Point, hexadecimal (with leading 0x) or decimal
eth_type	<number>	Can be hexadecimal (with leading 0x) or decimal
ip_tos	<number>	Can be hexadecimal (with leading 0x) or decimal
ipv4_src	<ip address>	xx.xx.xx.xx
ipv4_dst	<ip address>	xx.xx.xx.xx
tp_src	<number>	Can be hexadecimal (with leading 0x) or decimal
tp_dst	<number>	Can be hexadecimal (with leading 0x) or decimal

(Most Important) Actions

output	<number> all controller local in-port normal flood	no "drop" option (instead, specify no action to drop packets)
push_vlan	<eth-type-number>	Add VLAN header, <eth-type-number> can be hexadecimal (with leading 0x) or decimal **0x8100 for VLAN tag** (must be followed by set_field=eth_vlan_id…)
pop_vlan		Remove 802.1Q header
set_field	<OXM->value>	examples: "set_field=eth_src->00:11:22:33:44:55" "set_field=eth_vlan_vid->1"

Static Flow Pusher API: Deleting a Flow

- DELETE request to URI
  `http://<controller_ip>:8080/wm/staticflowpusher/json`
- JSON payload

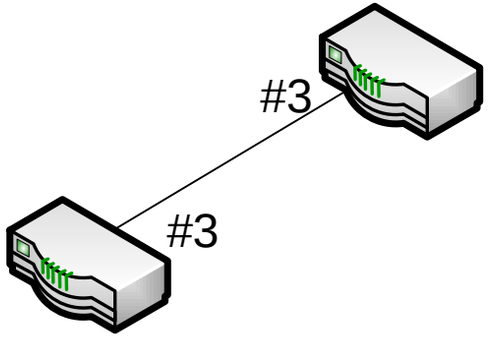
```
{
    "name":"flow-mod-1"
}
```

- Flow table of switch can be cleared using GET request to URI
  `http://<controller_ip>:8080/wm/staticflowpusher/clear/<switch-DPID>/json`
  - **`<switch-DPID>`** can be **`all`** to clear tables on all connected switches

Querying Topology Information

- GET request to URI
  `http://localhost:8080/wm/topology/links/json`
- Returns JSON document:

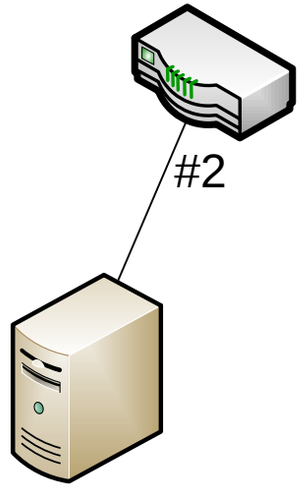
```
[{"src-switch":"00:00:00:25:90:94:6d:a8",
  "src-port":3,
  "src-port-state":0,
  "dst-switch":"00:00:00:25:90:93:97:9c",
  "dst-port":3,
  "dst-port-state":0,
  "type":"internal"},
 {"src-switch":"00:00:00:25:90:93:97:9c",
  "src-port":1,
  ...},
  ...
]
```



Query End Systems

- GET request to URI
  `http://localhost:8080/wm/device/`
- Returns JSON document:

```
[{"entityClass":"DefaultEntityClass",
  "mac":["52:54:00:97:11:33"],
  "ipv4":[],
  "vlan":[],
  "attachmentPoint":[
    {"errorStatus":null,
     "port":2,
     "switchDPID":"00:00:00:25:90:94:70:8c"}],
  "lastSeen":1365155842100},
  ...
]
```



Summary

- RESTful northbound interfaces support
  - Proactive routing
    - Flow pushing from controller to SDN switches
  - Queries about network state
    - Topology
    - Statistics
- RESTful northbound interfaces do not support:
  - Reactive routing
    - Events from SDN switches to network controller
  - Discussed in next part

Questions?