# OpenState demo

Hands-on activity

# Outline

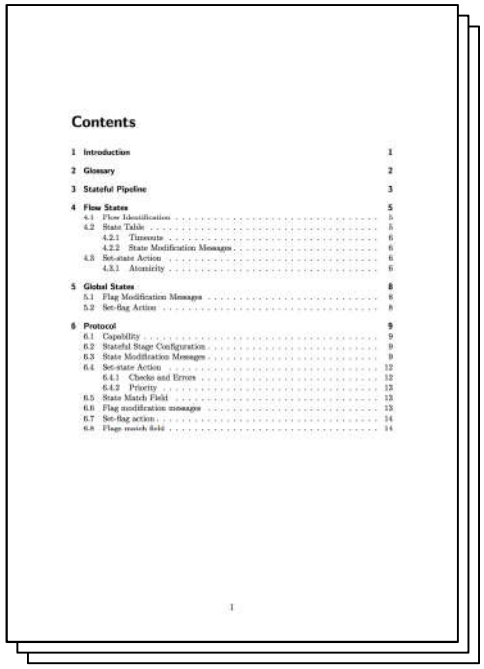
- **OpenState specification**
  - State table, key extractors, set-state action
- **Demo tools:**
  - Mininet, ofsoftswitch13, Ryu
- **Demo apps:**
  - Port Knocking, MAC Learning
  - Stateful composition

# OpenState specification

Overview

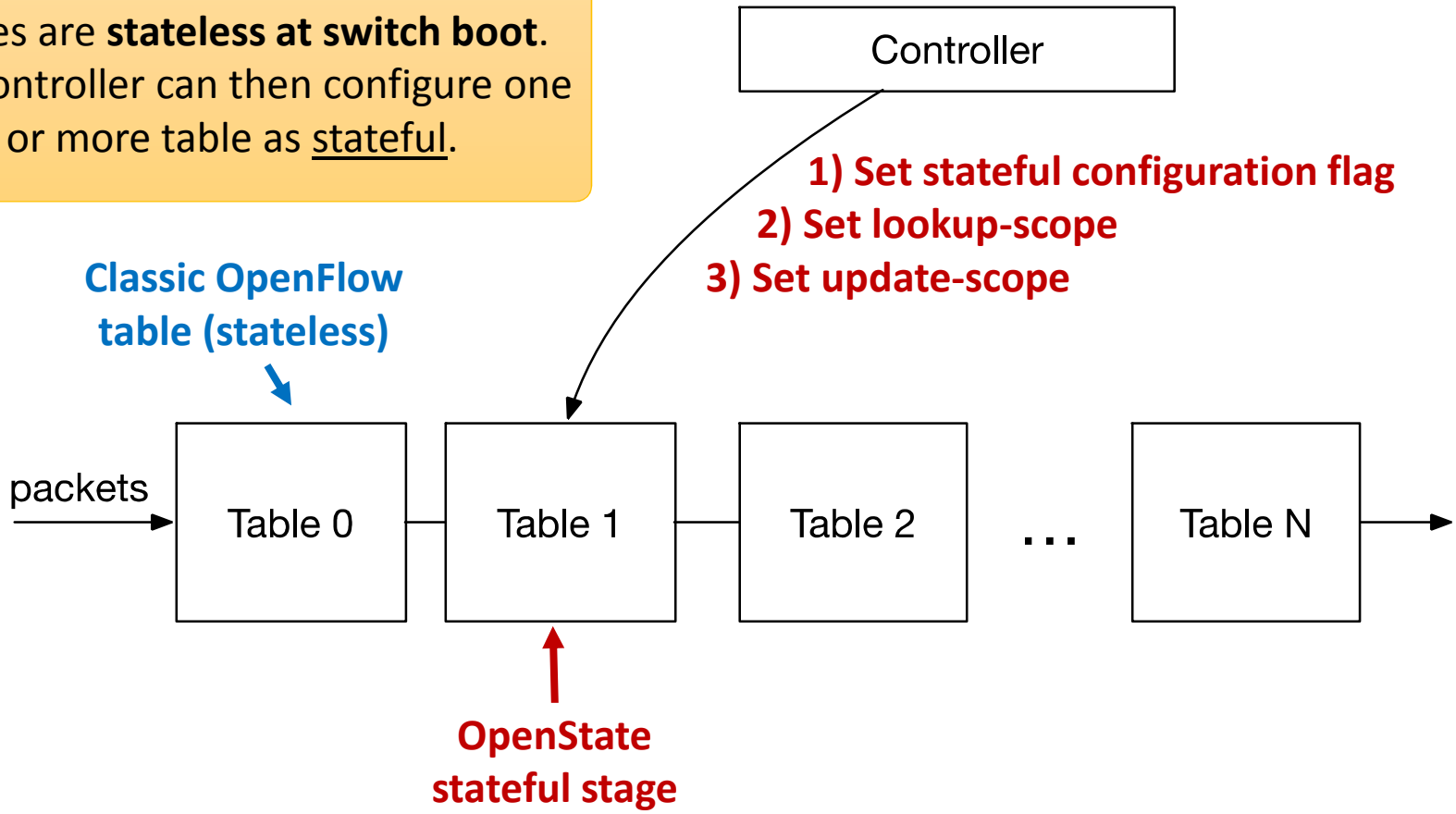
# OpenState specification

- **OpenFlow 1.3 extension**
- **Add new features:**
  - Stateful stage
  - Match on flow-states
  - Key extractors
  - Set-state action
  - Match on global-states
- **PDF available**
  - www.openstate-sdn.org

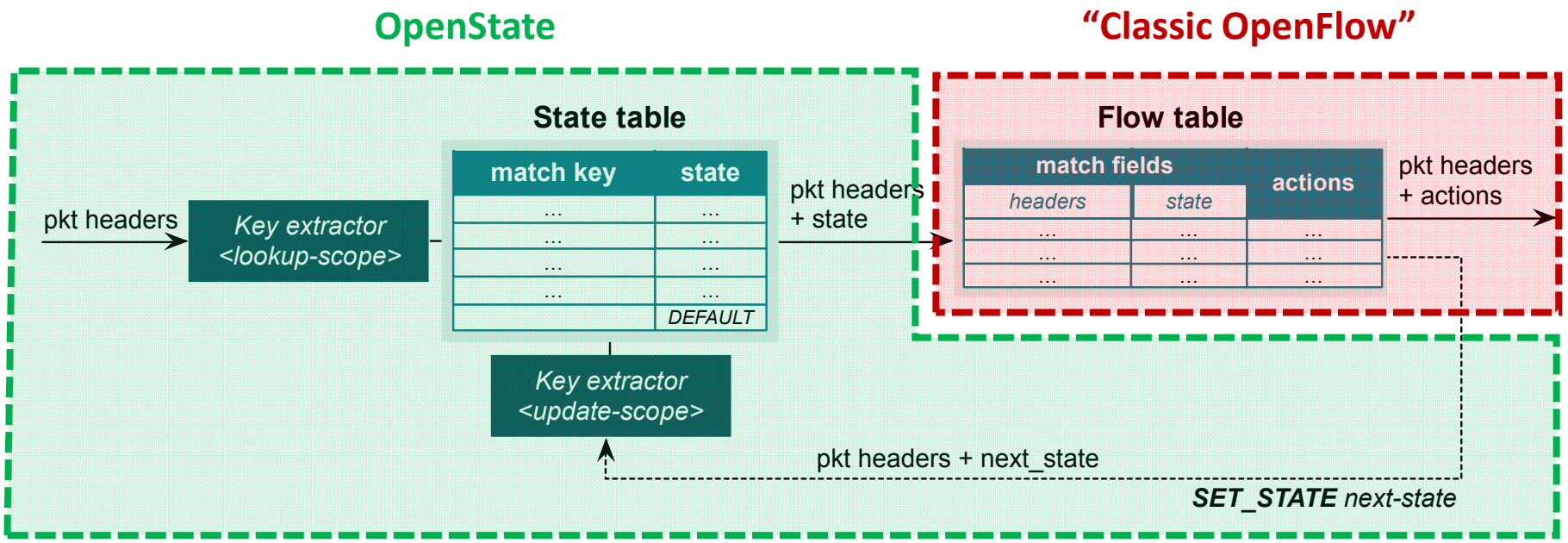

Contents

# Pipeline

Tables are **stateless at switch boot**. The controller can then configure one or more table as stateful.



# Stateful stage architecture



# Key extractors

- **Used to match the state table**
  - Lookup or update phase
- **Input: full packet headers**
- **Output: variable length bit sequence**
  - Concatenated header fields
- **Scope = ordered list of header fields**
  - E.g. {ip_src} → 32 bit key
  - E.g. {eth_src, eth_dst} → 96 bit key

# State table

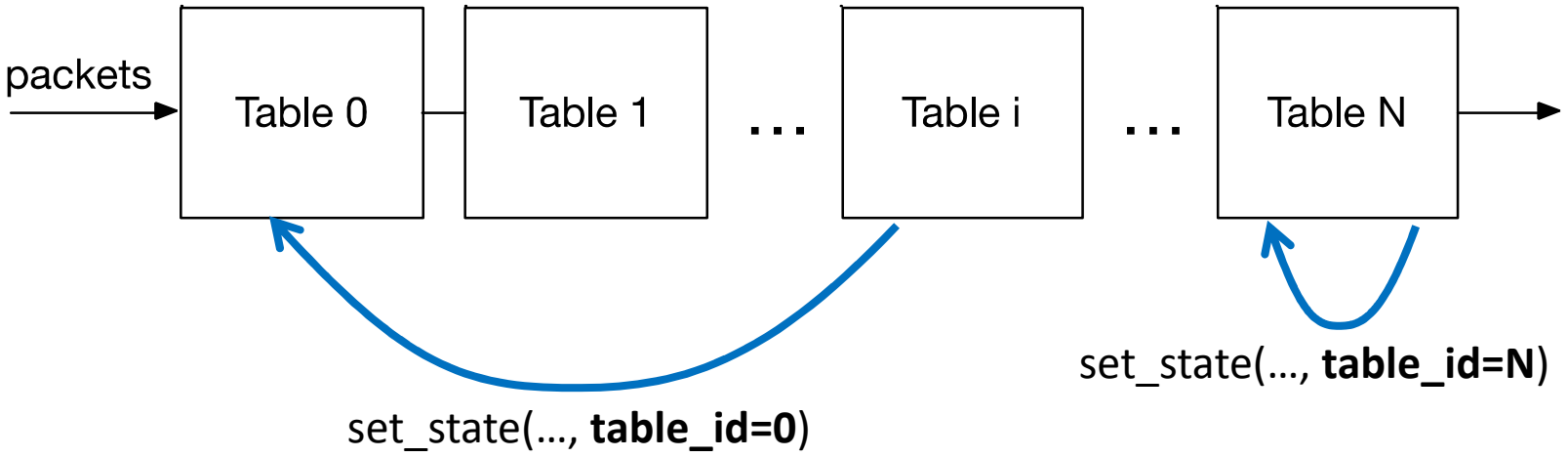
- **Exact match on packet key**
  - Efficient implementation in RAM (hash tables)
- **DEFAULT state if table miss**
- **NULL if invalid key**
  - Header fields not found by key extractors
- **Optional timeouts**
  - Idle or hard with rollback state

**State table**

| match key | state |
|---|---|
| … | … |
| … | … |
| … | … |
| … | … |
| ✳ | *DEFAULT* |

# Set-state action

- **set_state(*state*, *table_id*)**
  - **state:** 32bit integer
  - **table_id:** target stage to update
- **Inter-stage state updates**
  - Can update only previous state tables



# Other features

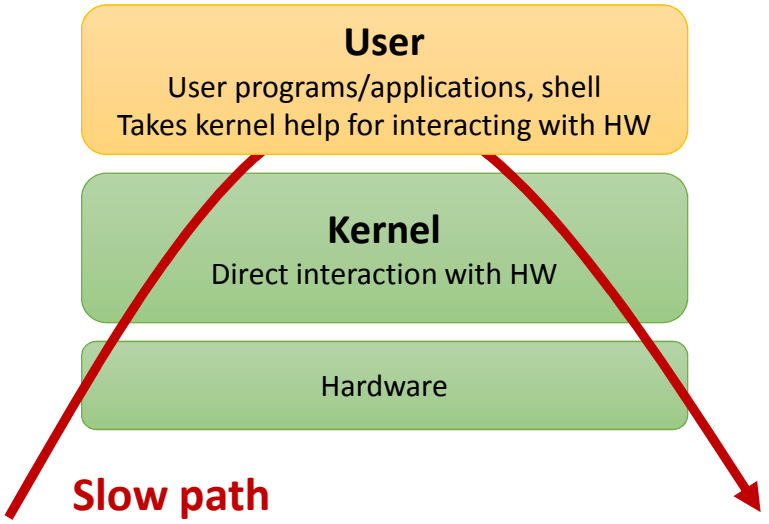
- **Maskable state match/write**
  - Match on state label can be masked
- **Idle/hard timeout on states**

# Emulation tools

Available at http://www.openstate-sdn.org

# ofsoftswitch13

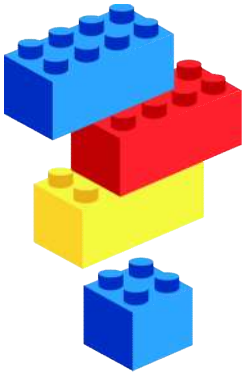
- **Open-source OF 1.3 Linux switch**
- **User-space implementation**
  - vs. fast kernel-space (e.g.: Open vSwitch)
- **Ease of development**
  - Fast experimentation
- **OpenState patched version**
  - www.openstate-sdn.org




http://cpqd.github.io/ofsoftswitch13/

# Ryu

- **Open-source OF 1.0-1.4 controller**
- **Component-based SDN framework**
  - Fast development of simple control application
  - vs. general purpose monolithic controller (e.g. OpenDaylight)
- **Written in Python**
- **OpenState patched version**
  - www.openstate-sdn.org

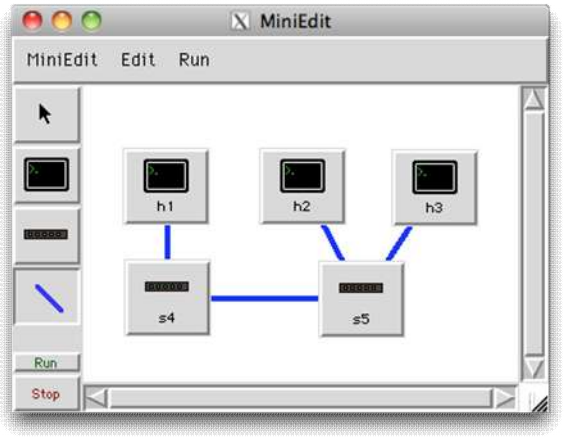


*Ryu means "flow" in Japanese.*
*Is pronounced "ree-yooh"*

http://osrg.github.io/ryu/

# Mininet

- **Open-source network emulator**
  - Virtual hosts, switches, controllers, and links
- **Arbitrary custom topologies**
- **Hosts run standard Linux software**
- **Switches support OpenFlow**
  - Open vSwitch, ofsoftswitch13
- **Python API**
  - Network creation, test automation
- **OpenState-enabled version**
  - www.openstate-sdn.org



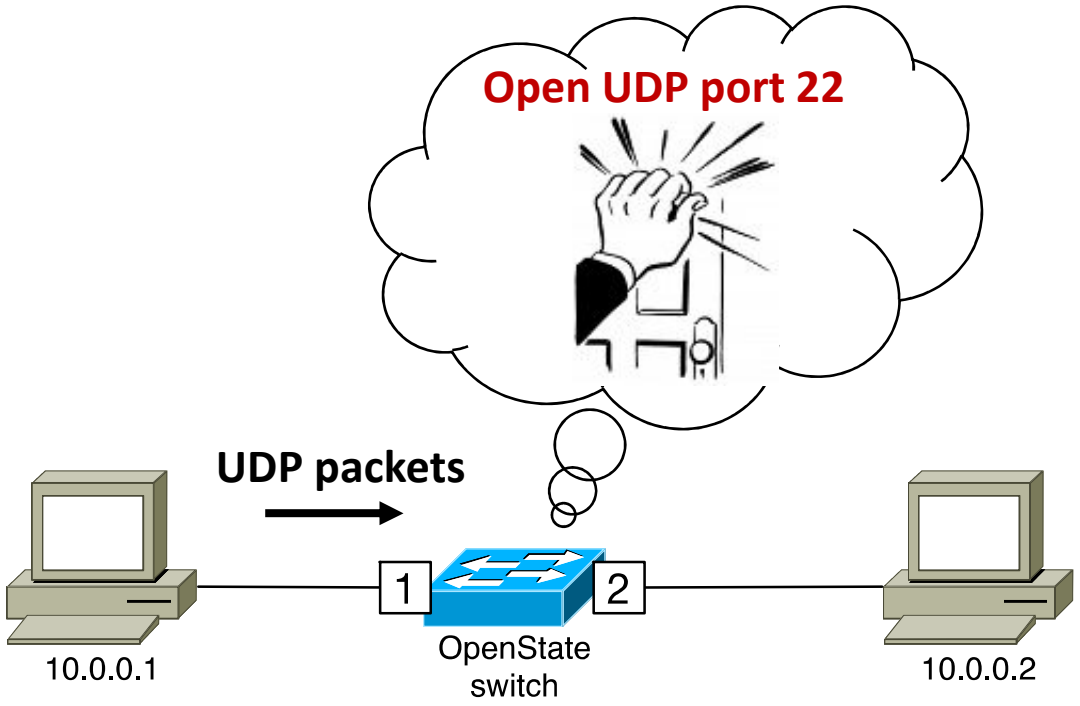

http://mininet.org/

# Applications

Implementation details & demo

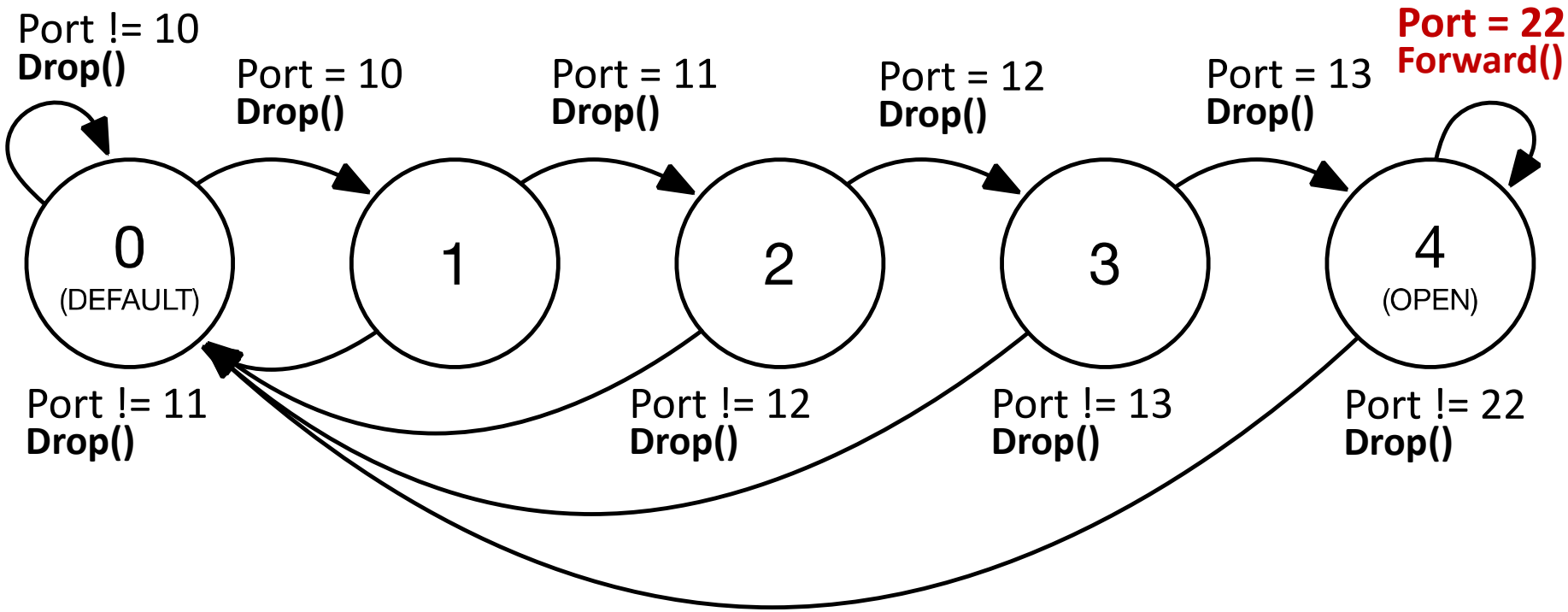
# Port knocking

**Secret knock sequence (UDP): 10, 11, 12, 13**



Port knocking

# Meally machine



Port knocking

# Table configuration

**Key extractors:**

Lookup-scope = {ip_src}

Update-scope = {ip_src}

Flows identified only by the IP source address

## Flow table (table_id = 0)

| | Priority | Match | Actions |
|---|---|---|---|
| *ARP packets (flood)* | 100 | eth_type=0x0806 | flood() |
| *1st knock* | 10 | eth_type=0x0800, ip_proto=17, state=0, udp_dest=10 | set_state(1, 0), drop() |
| *2nd knock* | 10 | eth_type=0x0800, ip_proto=17, state=1, udp_dest=11 | set_state(2, 0), drop() |
| *3rd knock* | 10 | eth_type=0x0800, ip_proto=17, state=2, udp_dest=12 | set_state(3, 0), drop() |
| *4th knock* | 10 | eth_type=0x0800, ip_proto=17, state=3, udp_dest=13 | set_state(4, 0), drop() |
| *Port 22 forward* | 10 | eth_type=0x0800, ip_proto=17, state=4, udp_dest=22 | output(2) |
| *Test port 1300* | 10 | eth_type=0x0800, ip_proto=17, state=4, udp_dest=1300 | output(2) |
| *Any other case, back to DEFAULT* | 0 | eth_type=0x0800, ip_proto=17 | set_state(0, 0), drop() |

Port knocking

# Code overview

## portknock.py

- **RYU app skeleton**
  - Switch features event handler
- **Set stateful stage**
  - Send stateful configuration flag
  - Set lookup key extractor
  - Set update key extractor
- **Populate flow table**

Port knocking

# Ryu app skeleton

## portknock.py

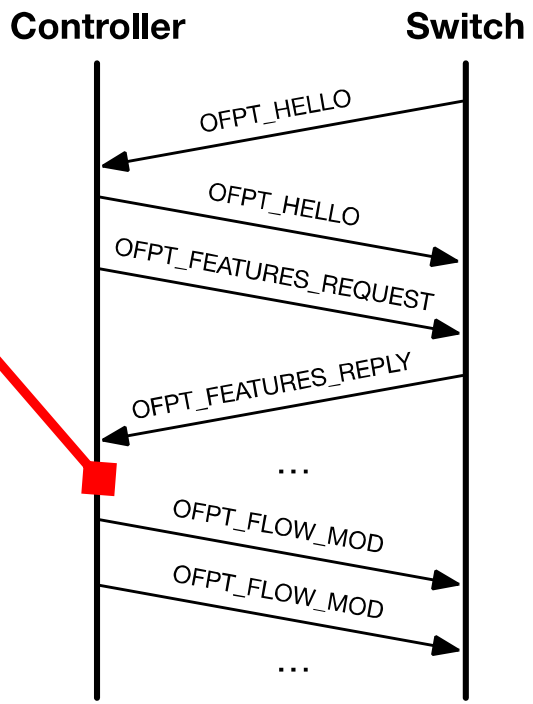
```python
class OSPortKnocking(app_manager.RyuApp):

    def __init__(self, *args, **kwargs):
        super(OSPortKnocking, self).__init__(*args, **kwargs)

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):

        msg = ev.msg
        datapath = msg.datapath
        # Check if OpenState is supported
        if not self.is_openstate_supported(msg.capabilities):
            LOG.error("ERROR: OpenState not supported")
            return
        ofp = datapath.ofproto
        parser = datapath.ofproto_parser

        LOG.info("Configuring switch %d..." % datapath.id)
```



Port knocking

# Set stateful stage

## portknock.py

```
""" Set table 0 as stateful """
req = parser.OFPTableMod(datapath=datapath,
                         table_id=0,
                         config=ofp.OFPTC_TABLE_STATEFUL)
datapath.send_msg(req)

""" Set lookup extractor = {ip_src} """
req = parser.OFPKeyExtract(datapath=datapath,
                           command=ofp.OFPSC_SET_L_EXTRACTOR,
                           field_count=1,
                           fields=[ofp.OXM_OF_IPV4_SRC],
                           table_id=0)
datapath.send_msg(req)

""" Set update extractor = {ip_src} (same as lookup) """
req = parser.OFPKeyExtract(datapath=datapath,
                           command=ofp.OFPSC_SET_U_EXTRACTOR,
                           field_count=1,
                           fields=[ofp.OXM_OF_IPV4_SRC],
                           table_id=0)
datapath.send_msg(req)
```

Port knocking

# Populate flow table

## portknock.py

```
""" ARP packets flooding """
match = parser.OFPMatch(eth_type=0x0806)
actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD)]
self.add_flow(datapath=datapath, table_id=0, priority=100,
              match=match, actions=actions)

""" Flow entries for port knocking """
for i in range(len(port_list)):
    match = parser.OFPMatch(eth_type=0x0800, ip_proto=17,
                            state=i, udp_dst=port_list[i])
    if port_list[i] != final_port:
        # If state not OPEN, set state and drop (implicit)
        actions = [parser.OFPActionSetState(state=i+1, table_id=0)]
    else:
        actions = [parser.OFPActionOutput(2)]
    self.add_flow(datapath=datapath, table_id=0, priority=10,
                  match=match, actions=actions)
```

Port knocking

# Populate flow table (2)

## portknock.py

```
""" Get back to DEFAULT if wrong knock (UDP match, lowest priority) """
match = parser.OFPMatch(eth_type=0x0800, ip_proto=17)
actions = [parser.OFPActionSetState(state=0, table_id=0)]
self.add_flow(datapath=datapath, table_id=0, priority=0,
              match=match, actions=actions)

""" Test port 1300, always forward on port 2 """
match = parser.OFPMatch(eth_type=0x0800, ip_proto=17, udp_dst=1300)
actions = [parser.OFPActionOutput(2)]
self.add_flow(datapath=datapath, table_id=0, priority=10,
              match=match, actions=actions)
```

Port knocking

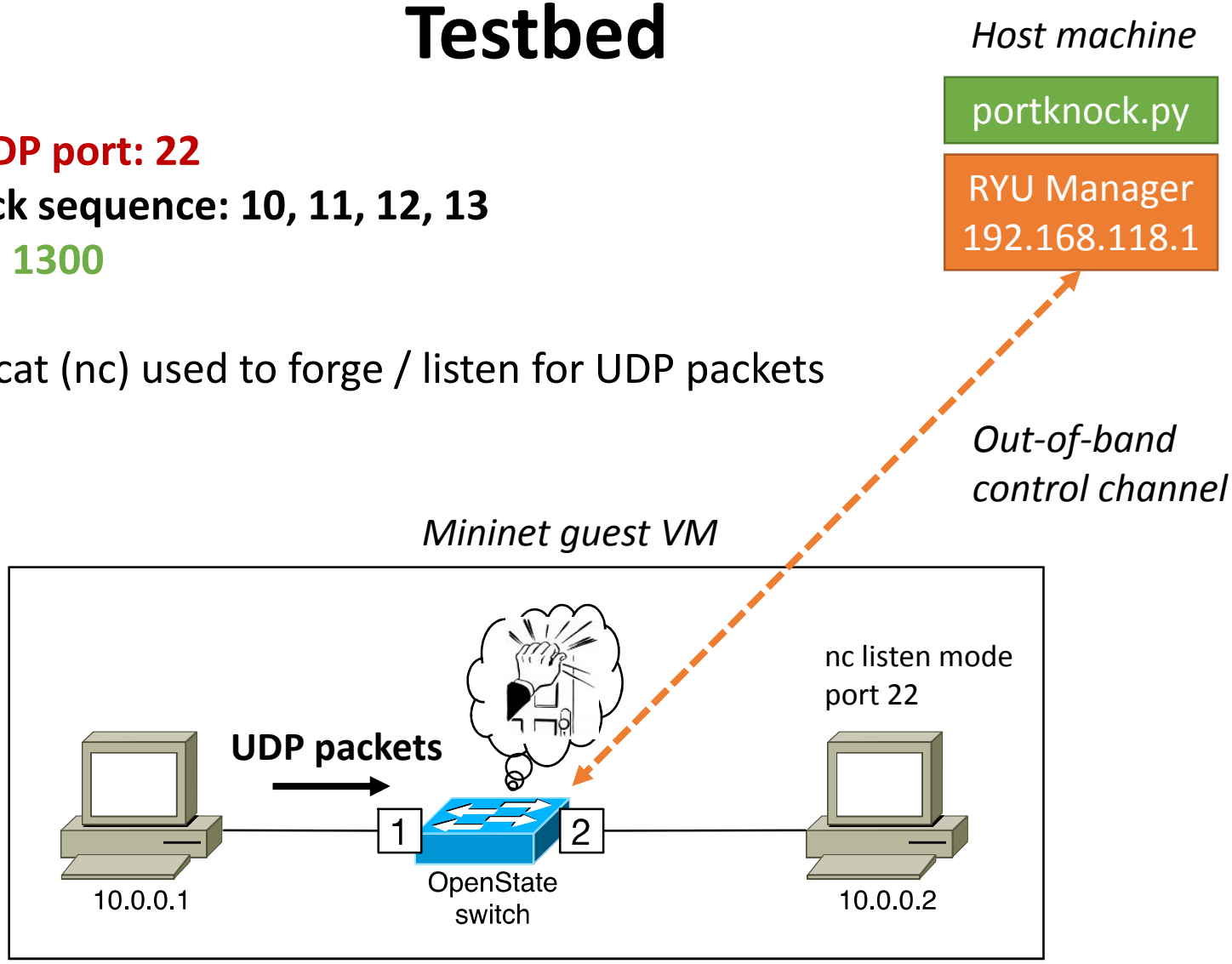
# Testbed

**Closed UDP port: 22**

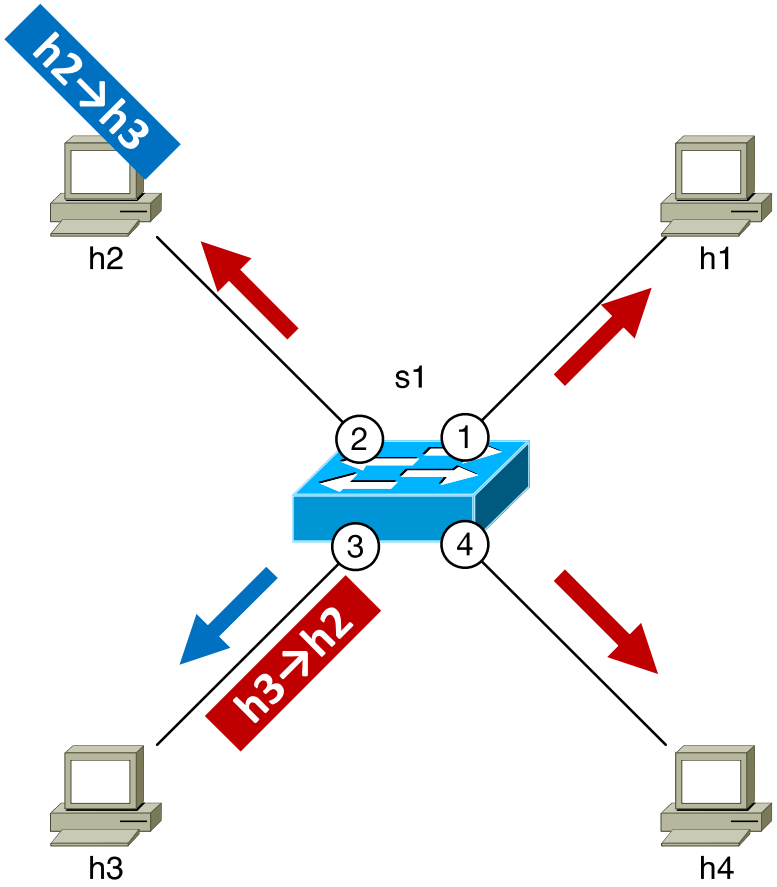
**UDP knock sequence: 10, 11, 12, 13**

**Test port: 1300**

UNIX Netcat (nc) used to forge / listen for UDP packets



# MAC learning




**MAC Table**

| MAC Addr | Port |
|---|---|
| h3 | 3 |
| h2 | 2 |

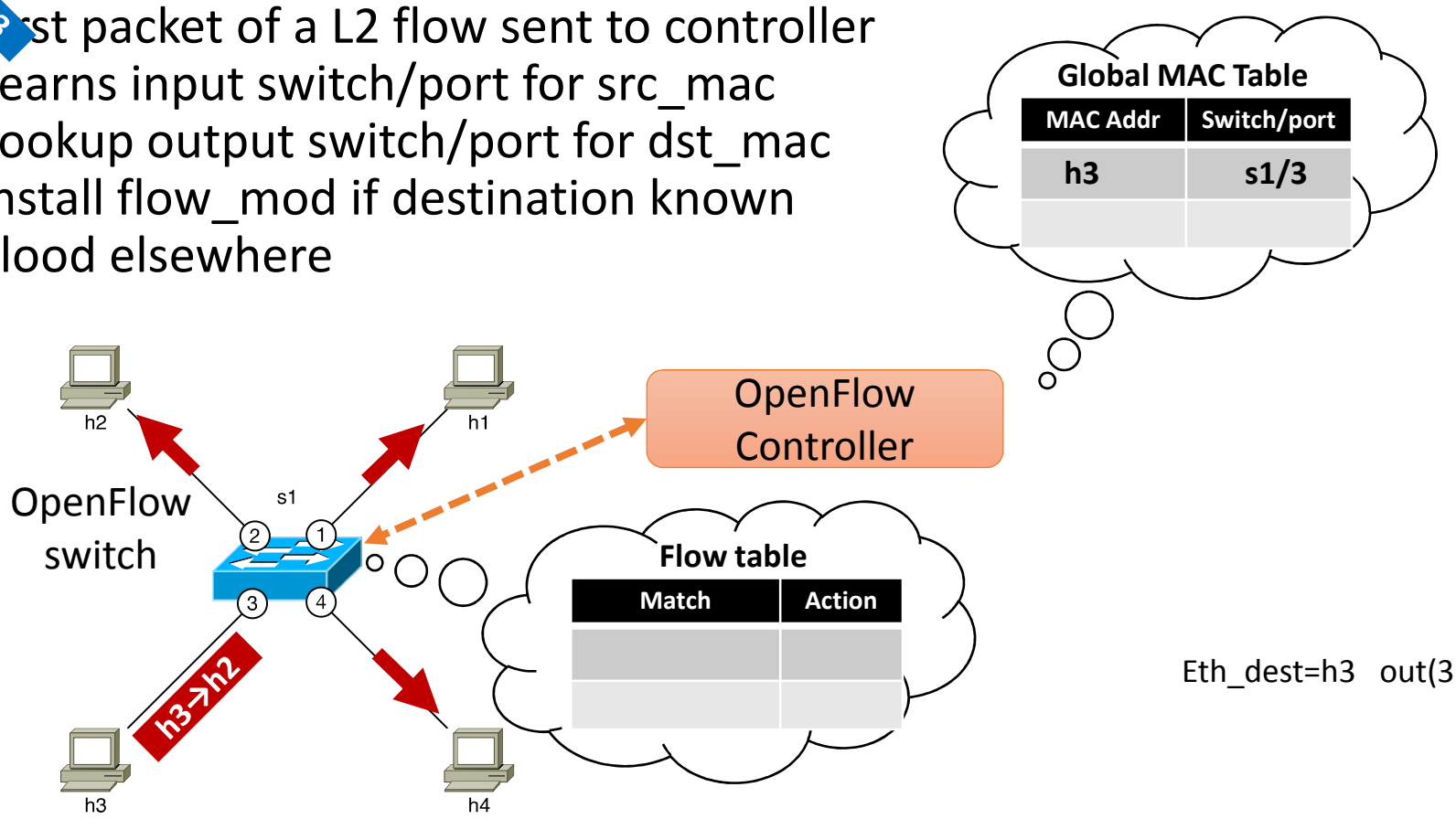
**Learn input port for each received frame**

**Flood when destination unknown**

MAC learning

# OpenFlow implementation

- **Reactive approach**
  - First packet of a L2 flow sent to controller
  - Learns input switch/port for src_mac
  - Lookup output switch/port for dst_mac
  - Install flow_mod if destination known
  - Flood elsewhere



MAC learning

# OpenFlow implementation

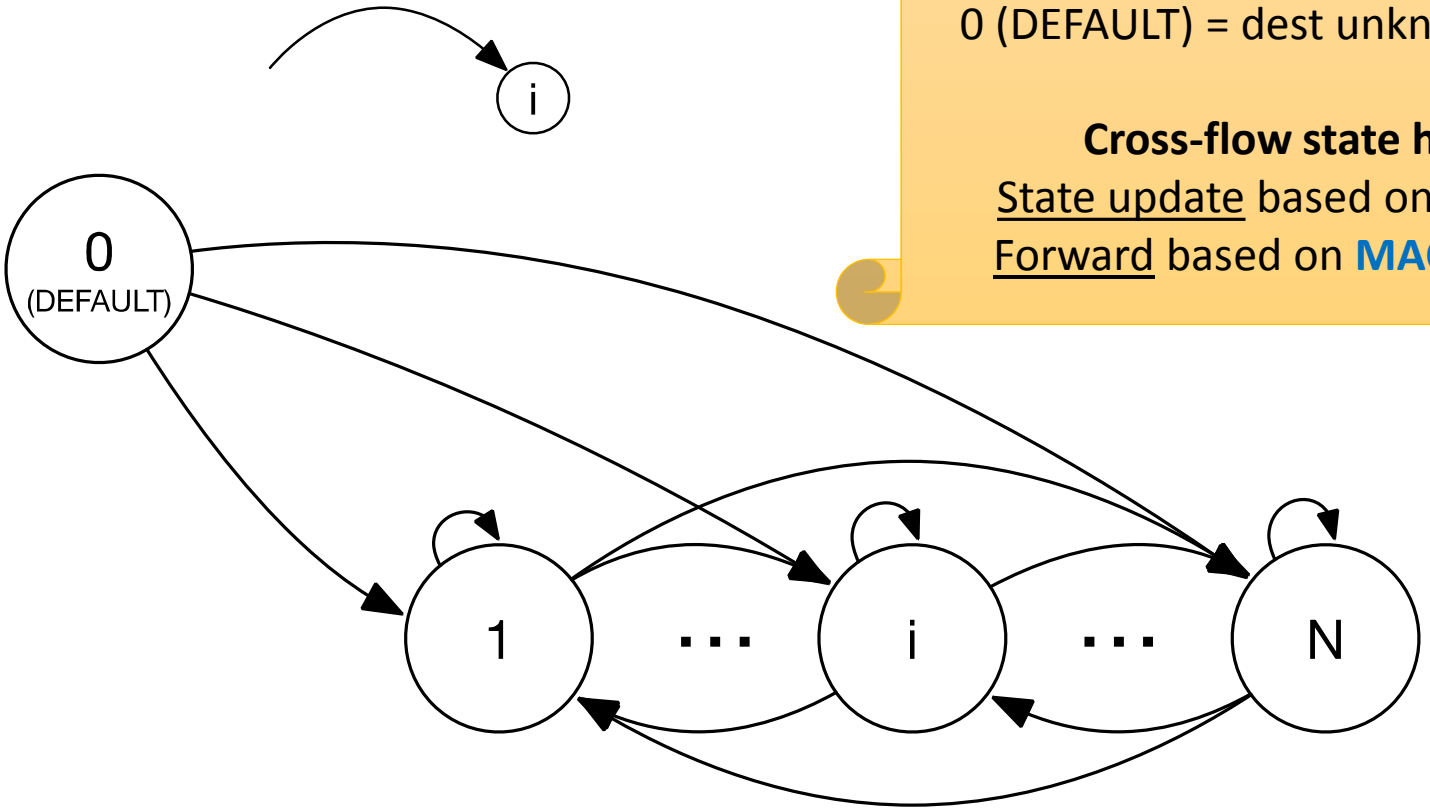
- **Full-state controller**
  - Store hosts location
- **Control messages required**
  - Learn hosts location (first packet)
  - Install flow-mod vs. flood packets (when dest unknown)
- **Different implementations/optimizations**
  - E.g.: topology-aware controller
    - learn at the edge + routed core
- **Flow table size depends on number of hosts**
  - MAC table explosion

MAC Learning

# Mealy machine

**in_port=i**

Forward(***state[eth_dst]***)




**State = Output port:**

N switch ports → N + 1 states

0 (DEFAULT) = dest unknown → Flood()

**Cross-flow state handling:**

State update based on **MAC source**

Forward based on **MAC destination**

MAC learning

# OpenState table configuration

**Key extractors:**

Lookup-scope = **{eth_dst}**

Update-scope = **{eth_src}**

Cross-flow state handling!

## Flow table (table_id = 0)

| Priority | Match | | Actions | |
|---|---|---|---|---|
| 0 | in_port=1, | state=0 | set_state(1, 0), | flood() |
| 0 | in_port=1, | state=1 | set_state(1, 0), | output(1) |
| 0 | in_port=1, | state=2 | set_state(1, 0), | output(2) |
| | | … | | |
| 0 | in_port=2, | state=0 | set_state(2, 0), | flood() |
| 0 | in_port=2, | state=1 | set_state(2, 0), | output(1) |
| | | … | | |
| 0 | in_port=N, | state=N | set_state(N, 0), | output(N) |

**N ports switch:**
**$N^2 + N$ entries**

MAC Learning

# Code Overview

## maclearning.py

- ~~RYU app skeleton~~
  - ~~Switch features event handler~~
    (same as portknocking.py)
- **Set stateful stage**
  - Send stateful configuration flag
  - Set lookup key extractor
  - Set update key extractor
- **Populate flow table**

MAC Learning – Code Overview

# Set Stateful Stage

## maclearning.py

```
""" Set table 0 as stateful """
req = parser.OFPTableMod(
        datapath=datapath,
        table_id=0,
        config=ofp.OFPTC_TABLE_STATEFUL)
datapath.send_msg(req)

""" Set lookup extractor = {eth_dst} """
req = parser.OFPKeyExtract(datapath=datapath,
        command=ofp.OFPSC_SET_L_EXTRACTOR,
        field_count=1,
        fields=[ofp.OXM_OF_ETH_DST],
        table_id=0)
datapath.send_msg(req)

""" Set update extractor = {eth_src}  """
req = parser.OFPKeyExtract(datapath=datapath,
        command=ofp.OFPSC_SET_U_EXTRACTOR,
        field_count=1,
        fields=[ofp.OXM_OF_ETH_SRC],
        table_id=0)
datapath.send_msg(req)
```

MAC Learning – Code Overview

# Populate Flow Table

## maclearning.py

```
# for each input port, for each state
for i in range(1, N+1):
    for s in range(N+1):
        match = parser.OFPMatch(in_port=i, state=s)
        if s == 0:
            out_port = ofp.OFPP_FLOOD
        else:
            out_port = s
        actions = [parser.OFPActionSetState(state=i, table_id=0),
                   parser.OFPActionOutput(out_port)]
        self.add_flow(datapath=datapath, table_id=0, priority=10,
                      match=match, actions=actions)
```

## MAC Learning

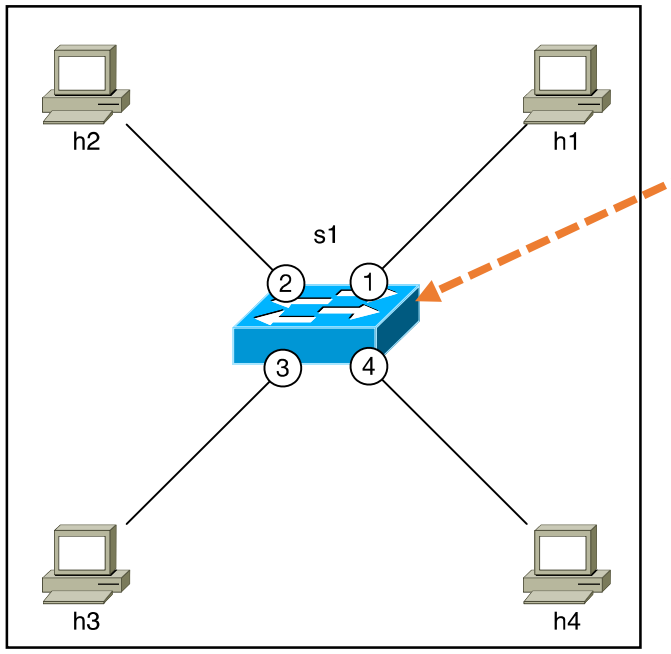
# Testbed

*Host machine*

maclearning.py

RYU Manager
192.168.118.1

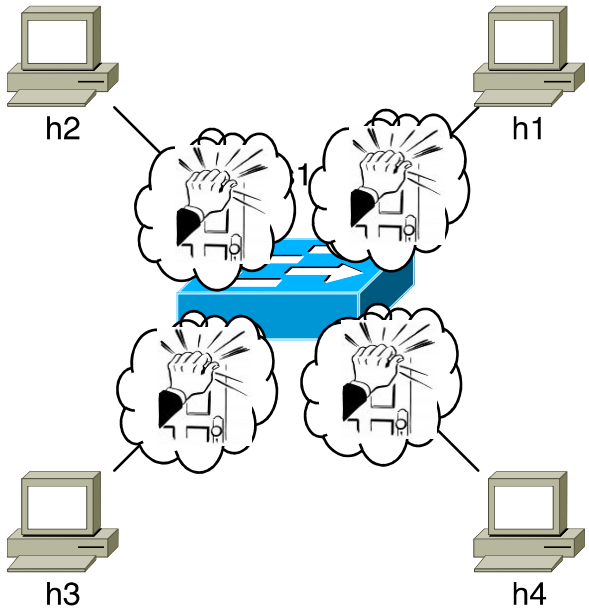
*Mininet guest VM*



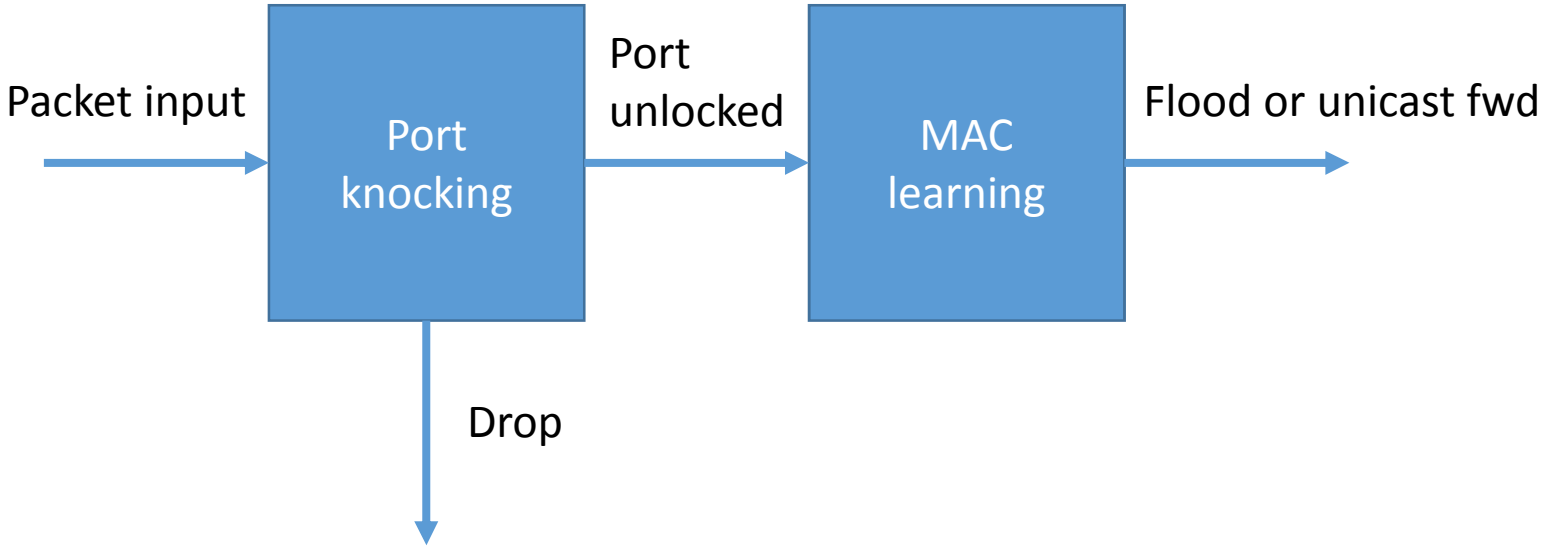

*Out-of-band control channel*

# Stateful composition: example

- **Port knocking + MAC learning**
  - Knock sequence required to enable port and forward packets



# 2 stateful stages



Stateful composition

# 1st table: port knocking

**Key extractors:**

Lookup-scope = {in_port, eth_src}

Update-scope = {in_port, eth_src}

Each host must unlock each port

## Flow table (table_id = 0)

| | Priority | Match | Actions |
|---|---|---|---|
| *Port 22 forward* | 100 | state=4 | Go_to(1) |
| *1st knock* | 10 | eth_type=0x0800, ip_proto=17, state=0, udp_dest=10 | set_state(1, 0), drop() |
| *2nd knock* | 10 | eth_type=0x0800, ip_proto=17, state=1, udp_dest=11 | set_state(2, 0), drop() |
| *3rd knock* | 10 | eth_type=0x0800, ip_proto=17, state=2, udp_dest=12 | set_state(3, 0), drop() |
| *4th knock* | 10 | eth_type=0x0800, ip_proto=17, state=3, udp_dest=13 | set_state(4, 0), drop() |

Stateful composition

# 1st table: MAC learning

**Key extractors:**

Lookup-scope = **{eth_dst}**

Update-scope = **{eth_src}**

## Flow table (table_id = 0)

| Priority | Match | Actions |
|---|---|---|
| 0 | in_port=1, state=0 | set_state(1, 0), flood() |
| 0 | in_port=1, state=1 | set_state(1, 0), output(1) |
| 0 | in_port=1, state=2 | set_state(1, 0), output(2) |
| | … | |
| 0 | in_port=2, state=0 | set_state(2, 0), flood() |
| 0 | in_port=2, state=1 | set_state(2, 0), output(1) |
| | … | |
| 0 | in_port=4, state=4 | set_state(4, 0), output(4) |

**4 ports switch: 20 entries**

# Thanks!