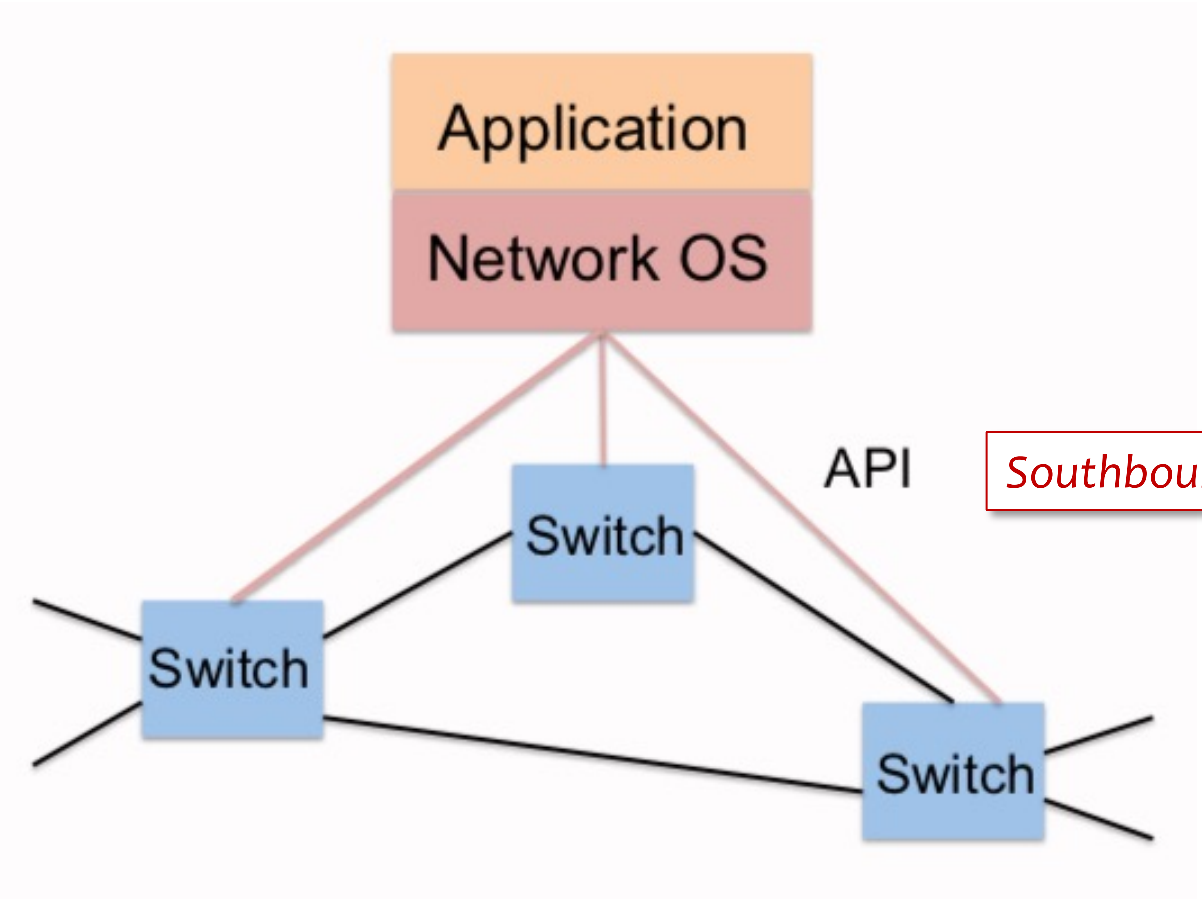
# Software-defined Networking (SDN) Part II

Prof. Anja Feldmann, Ph.D.

Balakrishnan Chandrasekaran, Ph.D.

Savvas Zannettou, Ph.D.

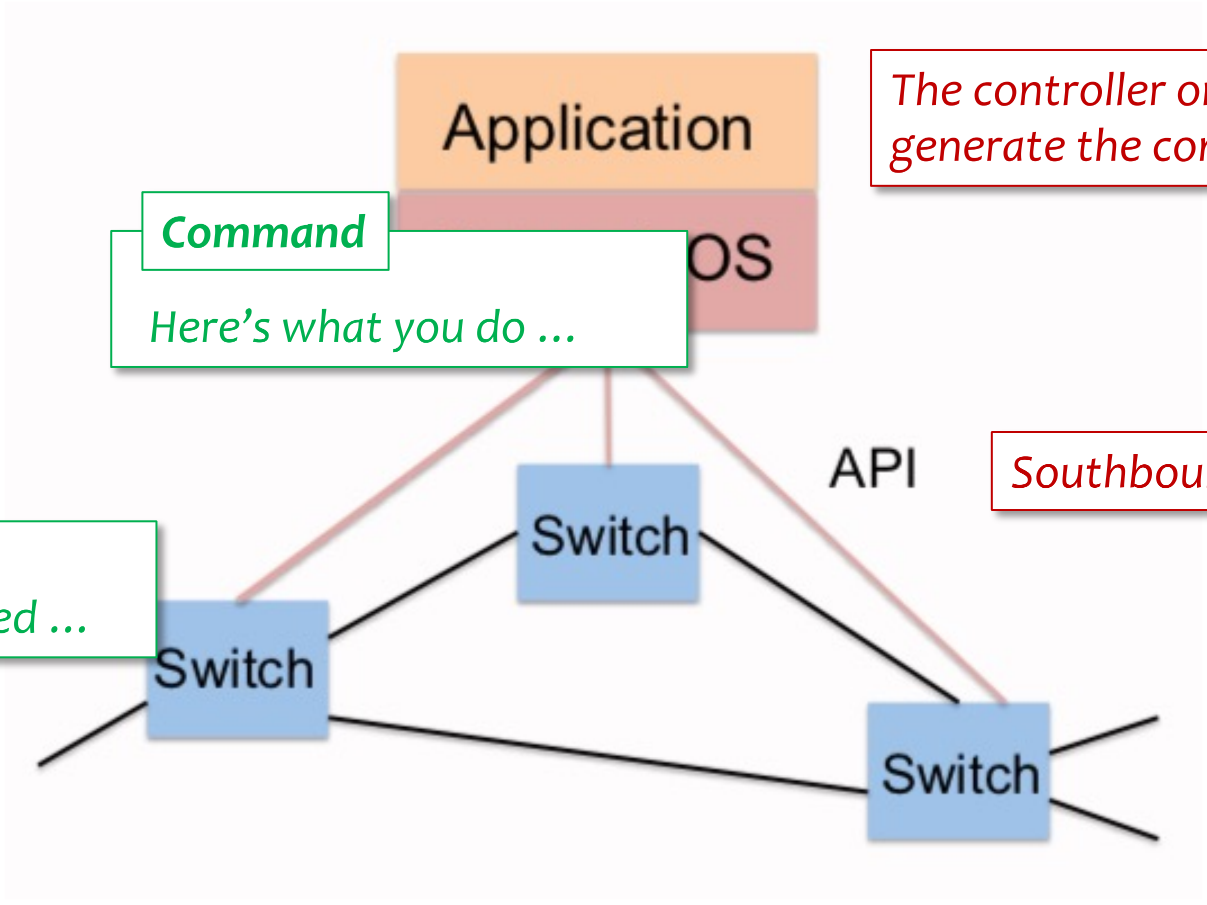
# Controller: Network OS



*Southbound API*

*[http://www.cs.princeton.edu/courses/archive/fall13/cos597E/docs/03API.pdf]*

# Controller: Network OS



*The controller or the application can generate the command.*

**Command**

*Here's what you do …*

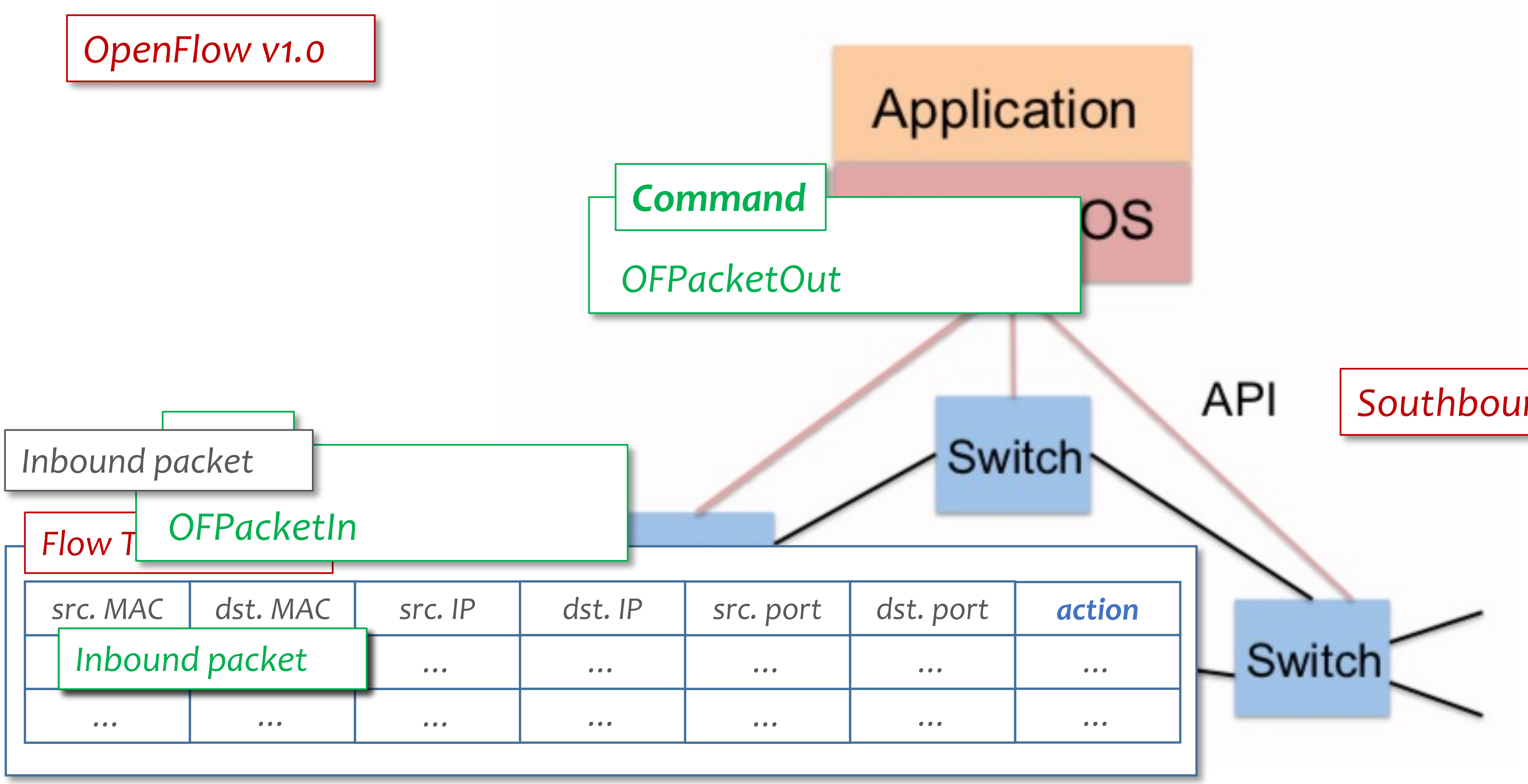
*Southbound API*

**Event**

*Something happened …*

*[http://www.cs.princeton.edu/courses/archive/fall13/cos597E/docs/03API.pdf]*

# Controller: Network OS



| src. MAC | dst. MAC | src. IP | dst. IP | src. port | dst. port | *action* |
|---|---|---|---|---|---|---|
| Inbound packet | | … | … | … | … | … |
| … | … | … | … | … | … | … |

[http://www.cs.princeton.edu/courses/archive/fall13/cos597E/docs/03API.pdf]

# OpenFlow

- Protocol that provides access to the forwarding plane of network switches
- Standardized by Open Networking Foundation
  - Current version: 1.5.1
- OpenFlow is considered an ”enabler” of SDN

# OpenFlow specification overview

- The OpenFlow protocol supports three message types:
  - **Controller-to-switch:** Initiated by the controller and are used to manage switches (e.g., read-state, modify-state, etc.)
  - **Asynchronous:** Initiated by the switch and are used to update the controller on events and changes on the switch state (packet-in, flow-removed, port-status, error)
  - **Symmetric:** Initiated by either the switch or the controller and sent without solicitation (e.g., hello, echo)

# OpenFlow messages

## *OFPacketIn*

- *Asynchronous message*
- *When first packet of a flow (with no matching rule in the flow table) arrives at an OpenFlow switch*
- Attributes: Details to help the controller/application decide
  - *Switch ID, incoming port, headers, …*

# OpenFlow messages

## *OFPacketOut*

- *Controller-to-Switch message*
- *Instruction for the switch on what to do with the packet*
    - *What to do?* ***Forward, Drop***
- Attributes: Details to help the switch carry out the action
    - *Action, buffer ID, …*

# Controller: Application example

## *OFPacketIn*

- Assume
  - *Switch ID: **A**, incoming port: **1**, headers, …*

## *OFPacketOut*

- Assume
  - *Switch ID: **A**, outgoing port: *** , headers, …*

## *Application?*

- ***Hub***

# SDN Application: It can’t be that simple …

Well, here’s an example implementation

```
# Create packet out message
msg = of.ofp_packet_out()
# Use the incoming packet as the data for the packet out
msg.buffer_id = event.ofp.buffer_id
# Set the in_port so that the switch knows
msg.in_port = packet_in.in_port
# Add an action to send to the specified port
msg.match = of.ofp_match.from_packet(packet)
action = of.ofp_action_output(port = of.OFPP_FLOOD)
msg.actions.append(action)
# Send message to switch
self.connection.send(msg)
```

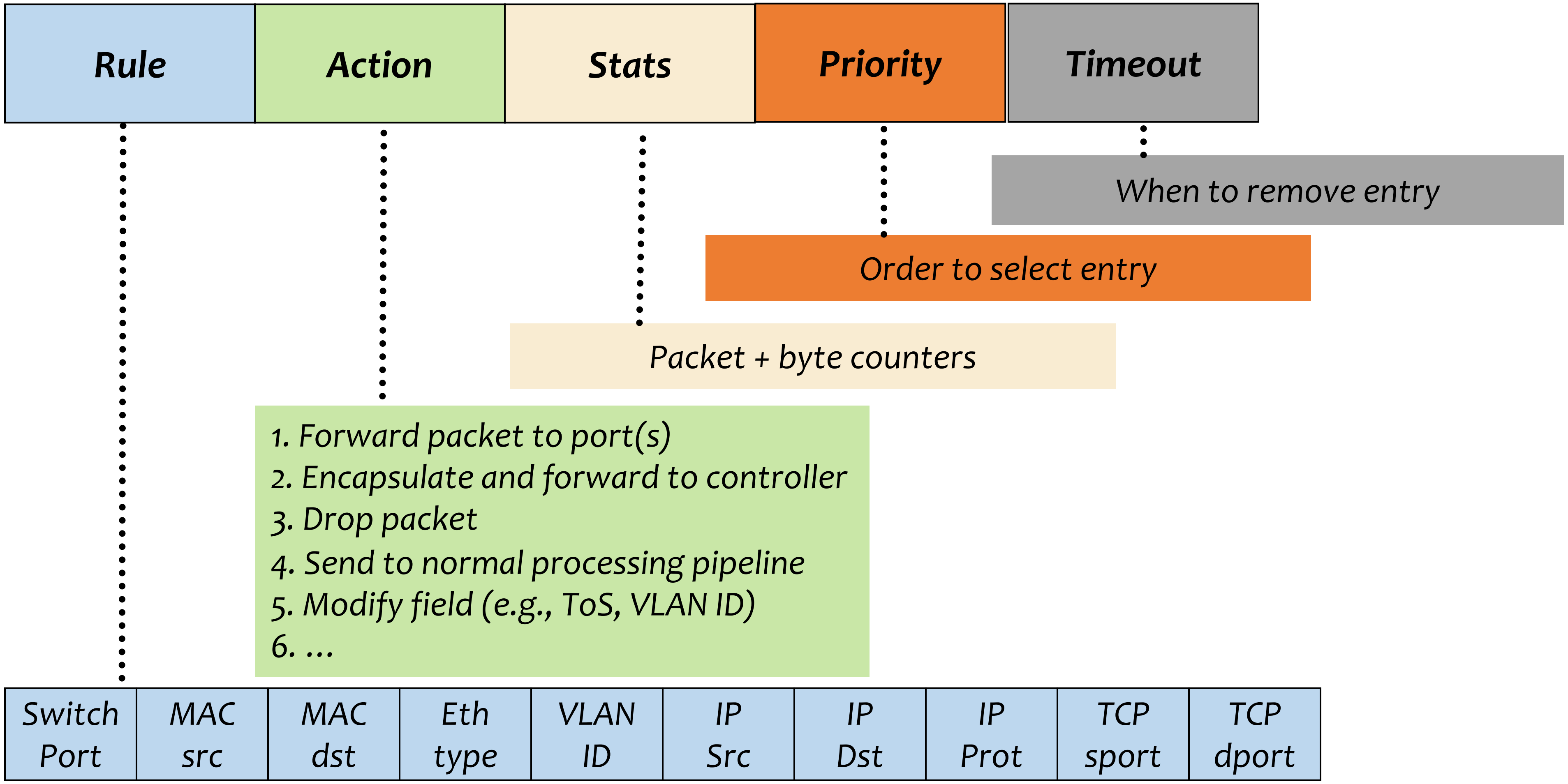
# Flow entries

## *Flow entries*

- Rather than *react* to every packet from the switch, we can install a flow entry that *matches* on certain packet headers

- Flow entries are *installed* by the *controller* and maintained in the *flow table*
  - Flow entries are generated by applications (either bundled with the controller or running on top of the controller)

# Anatomy of a flow table entry



| Rule | Action | Stats | Priority | Timeout |
|---|---|---|---|---|

When to remove entry

Order to select entry

Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. Modify field (e.g., ToS, VLAN ID)
6. …

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport |
|---|---|---|---|---|---|---|---|---|---|

+ mask

# Flow entry messages

## *OFFlowAdd*

- Add a new entry that matches on certain headers (or attributes)

  *e.g., all packets from source IP 1.2.3.4 and destination IP 4.3.2.1 (regardless of other attributes) to be dropped (Think, firewall!)*

## *OFFlowMod*

- Modify an existing flow entry in the switch (*e.g., change the above entry to not drop such packets*)

## *OFFlowDelete*

- Remove an existing flow entry in the switch

## *OFFlowRemoved*

- Tells a controller that a flow entry was removed (via timeout)

# OpenFlow statistics messages

## *OFStatisticsRequest*

- The controller instructs a switch to reply with an *OFStatisticsResponse* containing some statistics on traffic received by this switch
- *How? Switches have counters/meters to gather metrics!*

## *OFStatisticsResponse*

- Statistics on traffic observed by the switch (could be useful for traffic engineering)

# Flow entry expiration

- Each flow entry can have a soft (idle) and hard timeout
  - They define *when* a flow entry expires
- **Idle timeout**
  - Flow entry is removed, If no packet has matched the flow entry in the last "idle timeout" seconds
- **Hard timeout**
  - Flow entry is removed, if it has been "hard timeout" seconds since the flow entry was inserted
- If both timeouts are set to zero, the flow entry will never expire (the entry can still get removed by the controller!)

# SDN Dimensions: Flow insertion approaches

- Reactive flow insertion
  - First packet of the flow is sent to the controller
  - Controller installs flow entries on the switches
  - Subsequent packets (of the same flow) match the flow entry
  - Setup time overhead, loss of connection between controller/switch affects the network
- Proactive flow insertion
  - Controller pre-populates flow entries on the switches
  - Packets that do not match any flow entry are dropped
- Hybrid flow insertion
  - Controller pre-populates flow entries on the switches
  - Switches consult the controller for flows that do not match any entry

# SDN Dimensions: Granularity of flow rules

## Microflow

- One flow entry matches one flow
- Precision-oriented
  - Provides counters/metrics for individual flows
  - Allows/denies individual flows (access control)

## Aggregated rules (wildcards)

- One flow entry matches a group of flows
- Scalability-oriented
  - Minimizes overhead by grouping flows

Source: https://courses.cs.duke.edu/fall14/compsci590.4/lectures.html

# SDN Controller: Event-driven paradigm

- Switches *generate* events and send it to controller

- Application(s) *responds* to the events
  - Controller has (or interacts with) one or more applications

  - Application *subscribes* to (a subset of) events
  - When application receives an event, it responds with an output (command)
  - Controller sends this command to the switch

# SDN Controllers

- Many exist!
  - NOX (C++)
  - POX (Python)
  - Floodlight (Java)
  - OpenDaylight (Java)
  - Onix
  - ONOS
  - Pyretic

- *Different Implementations offer different services, applications, benefits, …*

# SDN application examples

- Different applications can install flow entries that perform different actions for different matches

*Flow Table*

| *src. MAC* | *dst. MAC* | *src. IP* | *dst. IP* | *src. port* | *dst. port* | ***action*** |
|---|---|---|---|---|---|---|
| … | … | … | … | … | … | … |
| … | … | … | … | … | … | … |

# SDN application examples

- Different applications can install flow entries that perform different actions for different matches

*Flow Table*

| src. MAC | dst. MAC | src. IP | dst. IP | src. port | dst. port | action | |
|---|---|---|---|---|---|---|---|
| * | H | … | … | … | … | port 8 | Switching |
| … | … | X | Y | … | 80 | port 2 | Routing |
| … | … | X | Y | * | 443 | port 4 | |
| … | … | X | Y | * | 22 | drop | Firewall |

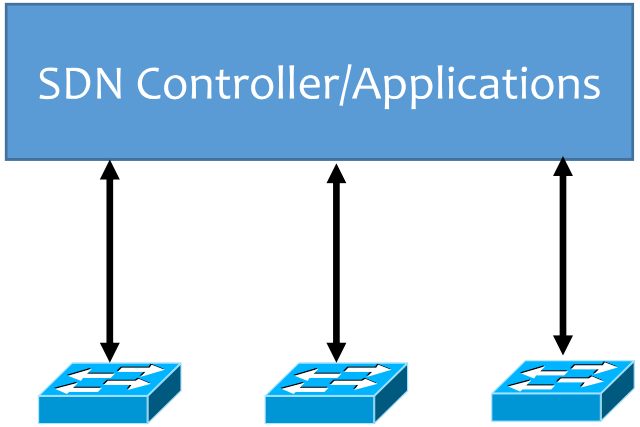
# SDN Challenges

# With great power comes …

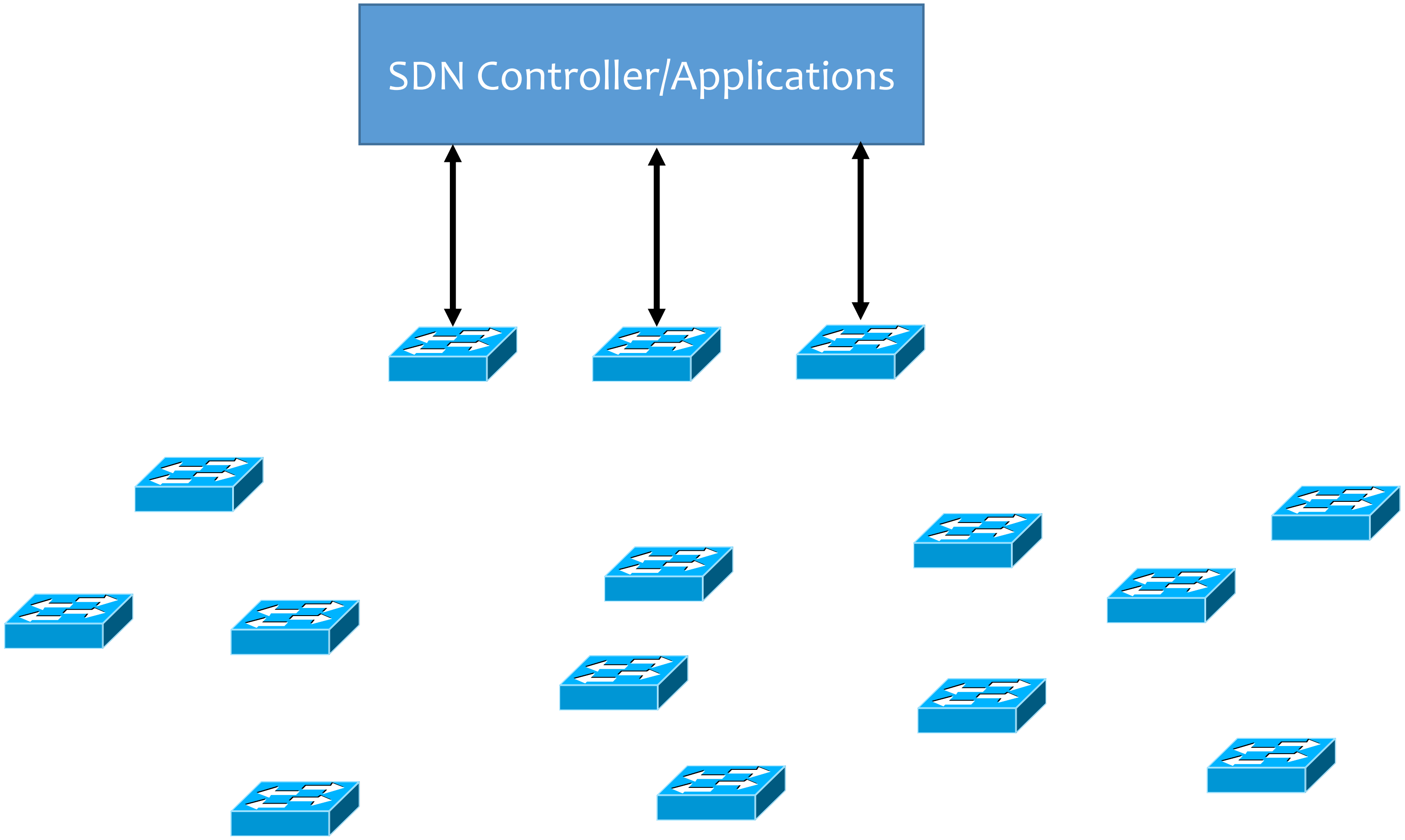
*… many great challenges!*

- SDN offers network-wide visibility, (programmable) control over switches, and a simple data-plane abstraction. Now, …
  - Ensure that SDN is available, fault-tolerant, and secure
  - *Need to map policies to the low-level API*
    - Ensure traffic from network A always is *screened/scrubbed*
    - Rules for identifying traffic from A, direct them to scrubber, ensure that this rule is always applied; how to ensure that the implementation matches policy?
  - *Need to compose modular applications, debug, and verify*
    - Diff. apps for load balancing, routing, traffic engineering, scrubbing (firewall)
    - How to compose or combine? How to debug and verify?

# SDN controller availability

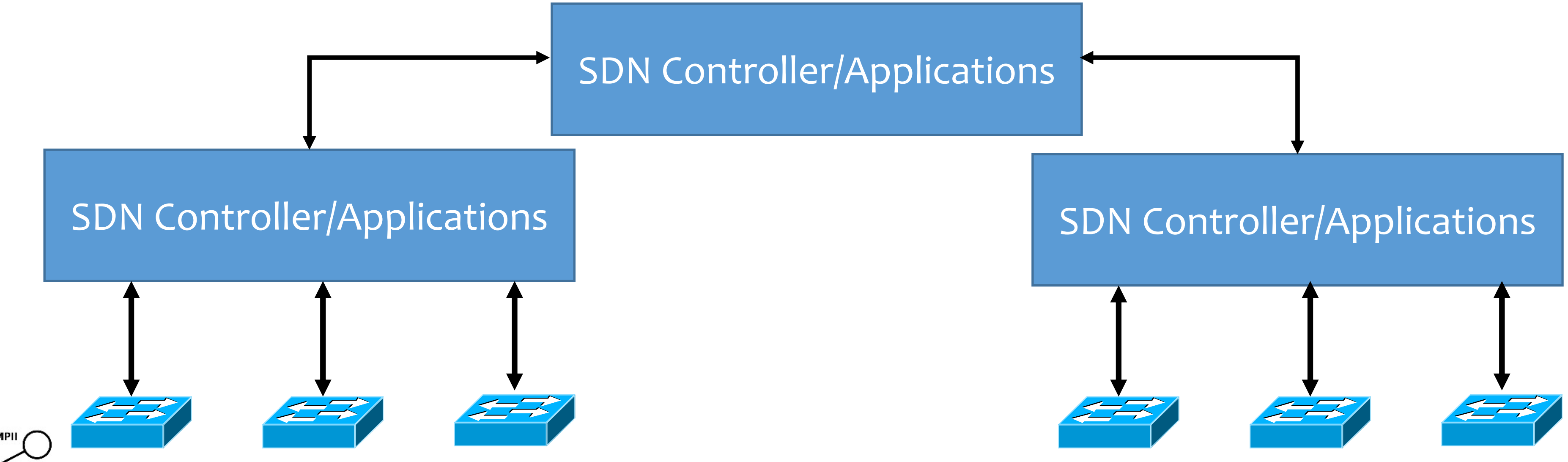


# SDN controller availability



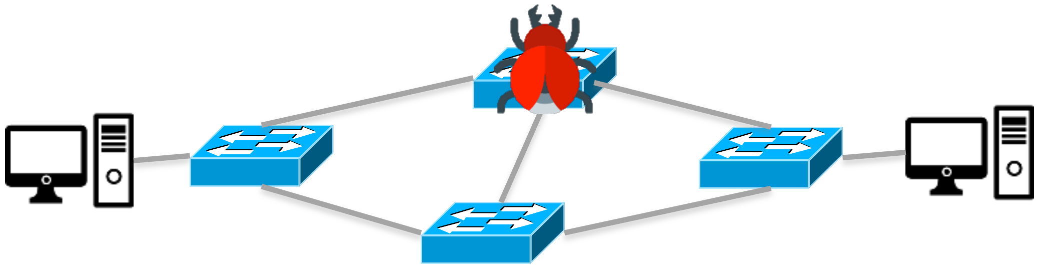
# SDN controller availability

- **"Divide and conquer" approach**
  - How many controllers?
  - How do you assign switches to controllers (e.g., reduce processing time)?
  - How to ensure consistency across controllers?

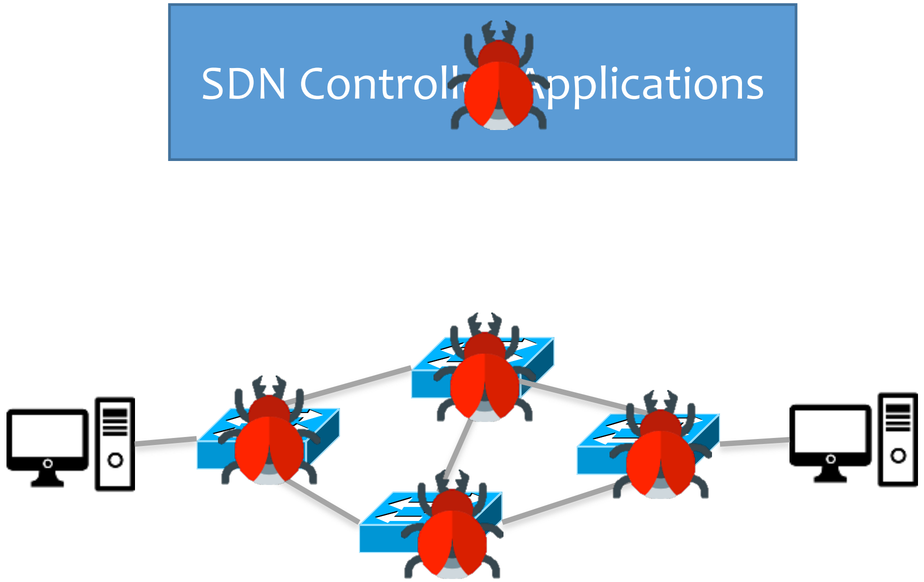

SDN Controller/Applications
SDN Controller/Applications
SDN Controller/Applications

# SDN fault tolerance

The network survives failures or bugs in code for network devices



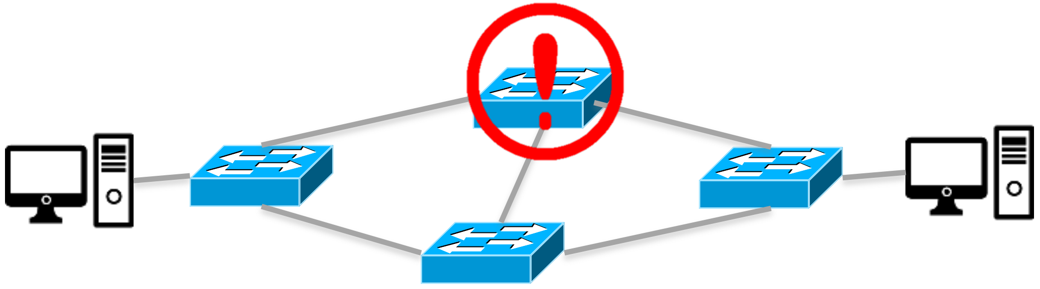
Bugs in the SDN controller or applications affect the entire network and can take the entire network down



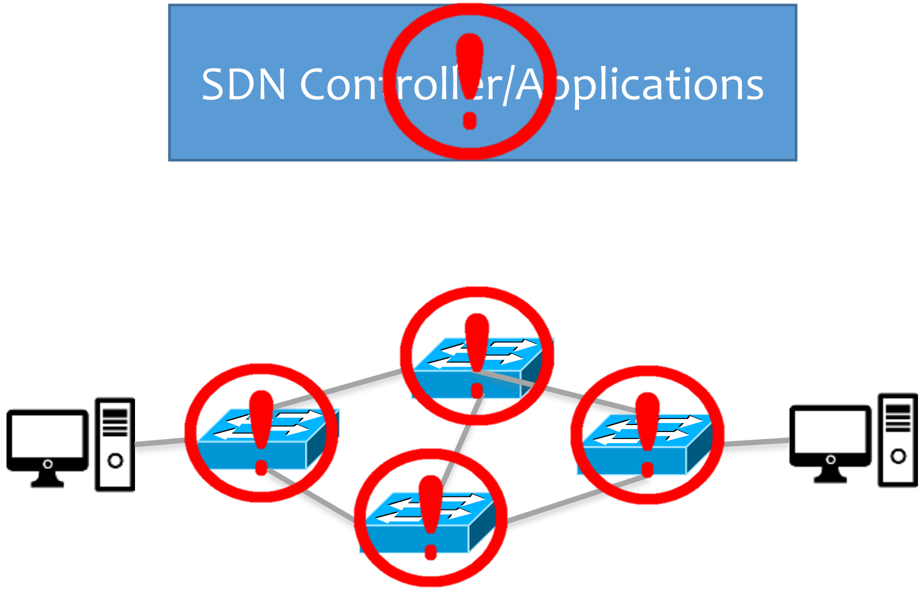
https://courses.cs.duke.edu/fall14/compsci590.4/notes/F14_Class1.pptx

# SDN security

If one device in traditional networking paradigm is compromised the network may still be safe



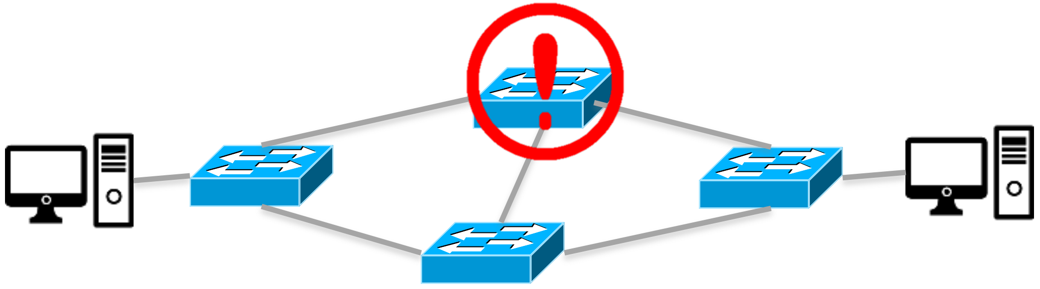
- If the SDN controller or applications get compromised, the network is not safe



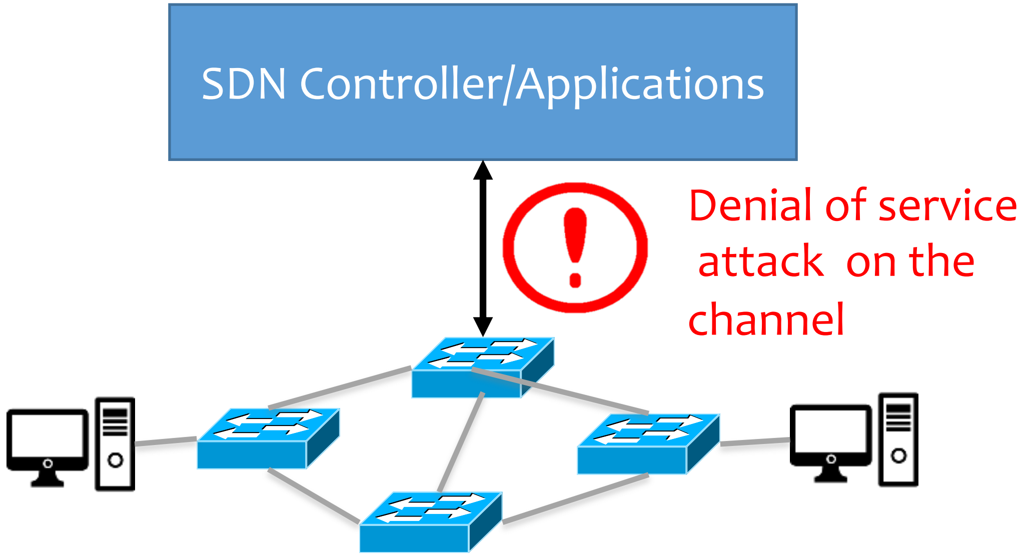
https://courses.cs.duke.edu/fall14/compsci590.4/notes/F14_Class1.pptx

# SDN security

If one device in traditional networking paradigm is compromised the network may still be safe



- If the SDN controller or applications get compromised, the network is not safe
- Communication channel between controller/switches can be attacked!


SDN Controller/Applications
Denial of service attack on the channel

https://courses.cs.duke.edu/fall14/compsci590.4/notes/F14_Class1.pptx

# Policies/Intents

## *How to specify policy/intent?*

- Policies may constitute rules across …
  - many switches
  - multiple applications

## *How to handle policy changes?*

- Applying policies without disrupting traffic in the network is hard
  - May have to change rules across many switches
  - … while handling traffic in the network (cannot stop traffic for changes: impractical!)

## *How to realize policies?*

- Hard for developers to handle all the complexities
- Need abstractions to simplify development

# SDN debugging

*When disaster strikes, how to debug?*

- Debugging network applications is hard!
  - Inputs to the SDN application are events/packets from the data plane
  - Outputs are policies spread across the entire network!

- Bugs can appear anywhere in the SDN stack
  - SDN controller, Applications, Switches (software and/or hardware)

# SDN verification

*How to verify that policies are implemented correctly?*

- Leverage *network invariants*
    - *Invariant?*
        - *Never changing; a property that always holds!*
        - *No route loops, no blackholes*

- Specify the invariants, for instance, to the controller
    - When the output of an application *violates* an invariant, flag it!
    - State of the art research efforts: *Header space analysis, Veriflow, …*

# SDN verification

*How to verify that rules installed are followed?*

- Network is a shared substrate
  - *Imagine an administrator manually entering/modifying/deleting a rule on a switch*
  - Verifying that data plane performs exactly as instructed by the control plane is hard!

  - Prof. Anja Feldmann’s group currently has an ongoing project in this space.

# Summary

- Data and control plane separation
- OpenFlow
- Controllers and applications
- Challenges