Software-defined Networking Session II

Advanced Computer Networks

David Koll

Exam Information

- July 16th, 12.00-14.00 (not 10-12am!!!)
- Room: MN08
- Written exam
  - Bring a non-erasable blue/black pen (no pencils!)
  - Bring your student ID
  - We provide paper
  - No additional tools allowed (e.g., no calculator)

Exam Information

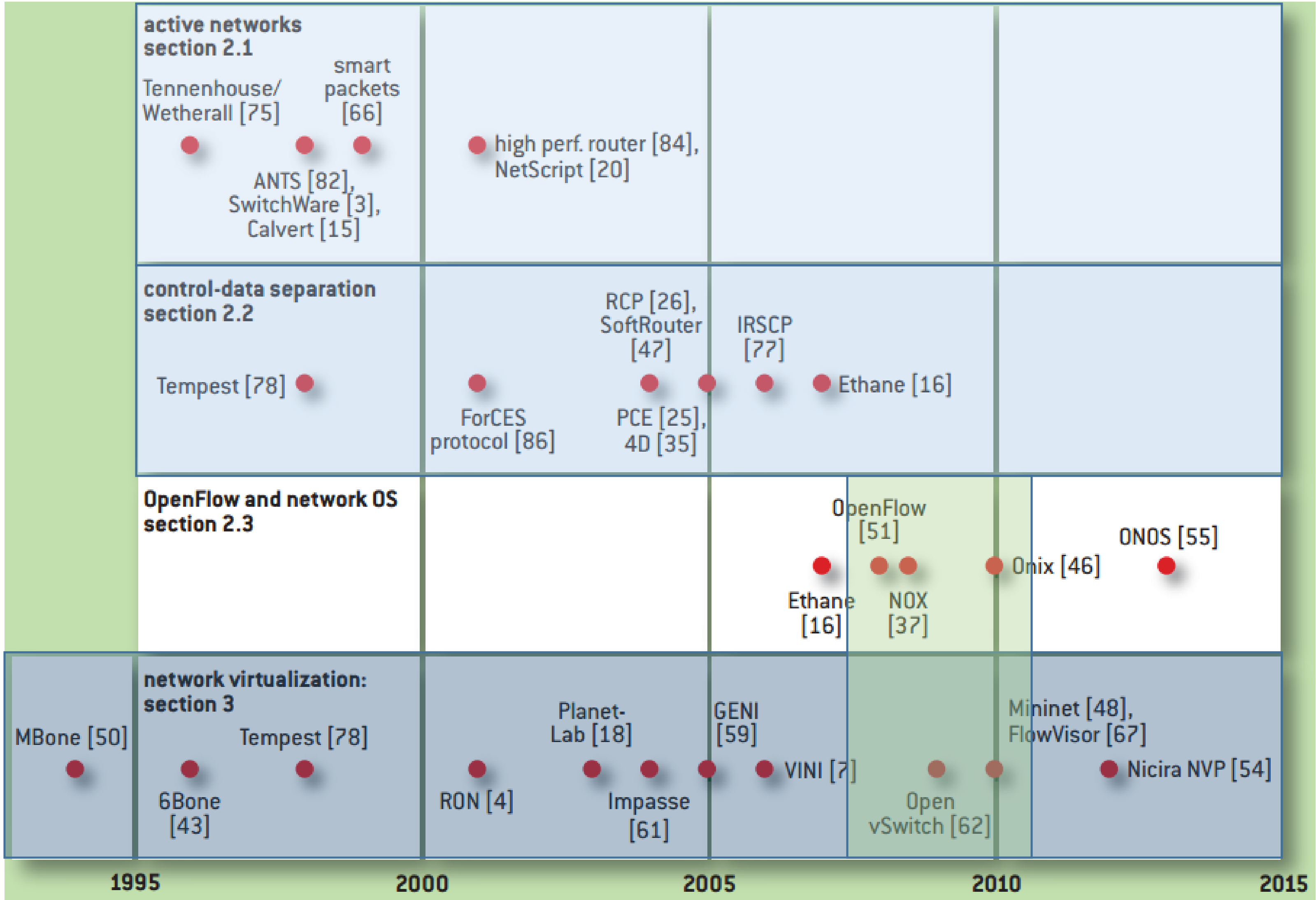
- All topics of the lecture will be covered.
  - Wireless
  - P2P
  - ICN/CCN
  - SDN
  - DCN
  - (Guest talk not relevant for exam)

- Know how concepts work, you will be asked to perform some operations
  - e.g., lookup in a Chord DHT

- Know why we need the concepts
  - (e.g., what are the reasons for using SDN or CCN

Partly based on slides of Nick McKeown, Scott Shenker, Nick Feamster, Jin Xin, and Jennifer Rexford

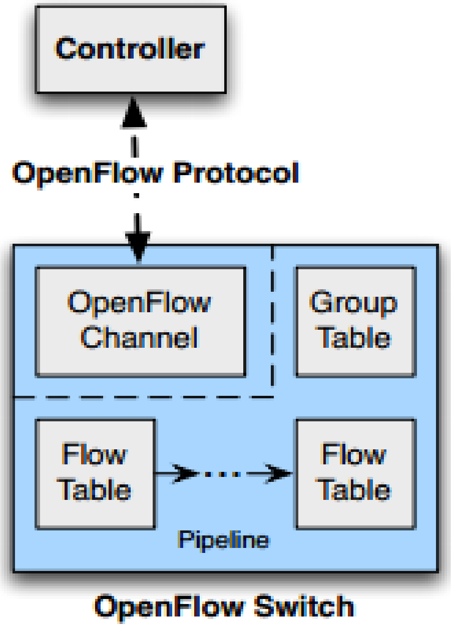
Recap



N. Feamster et al.: „The Road to SDN – An intellectural history of programmable networks" *ACM SIGCOMM Computer Communication Review* 44.2 (2014): 87-98.

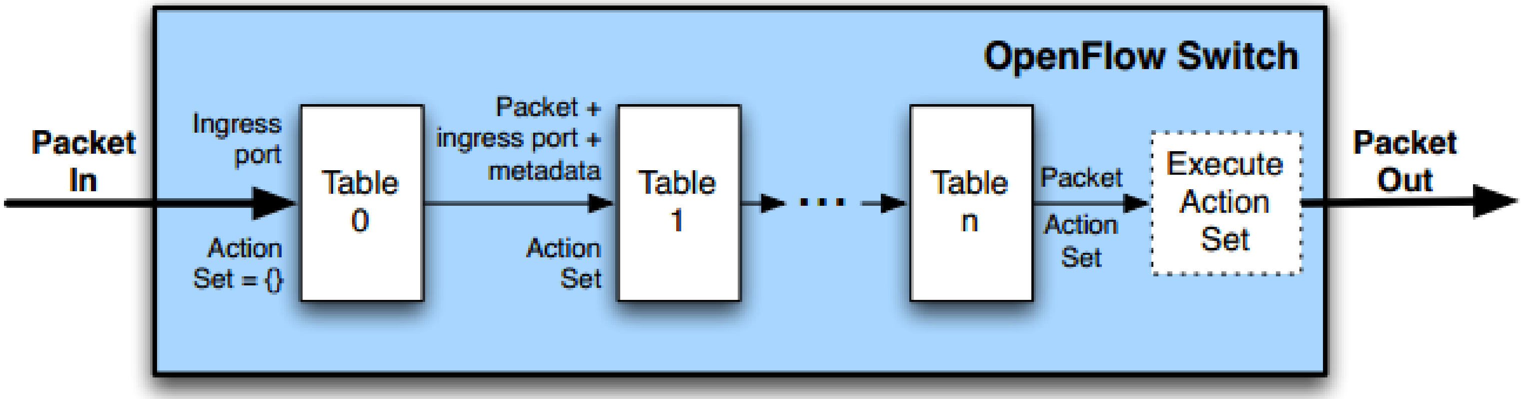
Recap: OpenFlow – A SDN Protocol

- Main components: *Flow* and *Group Tables*
  - Controller can manipulate these tables via the OpenFlow protocol (*add, update, delete*)
  - Flow Table: reactively or proactively defines how incoming packets are forwarded
  - Group Table: additional processing



Recap: OpenFlow – Switches

- Incoming packets are matched against Table 0 first
- Find highest priority match and execute instructions (might be a Goto-Table instruction)
- Goto: Only possible forward



Recap: Examples

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

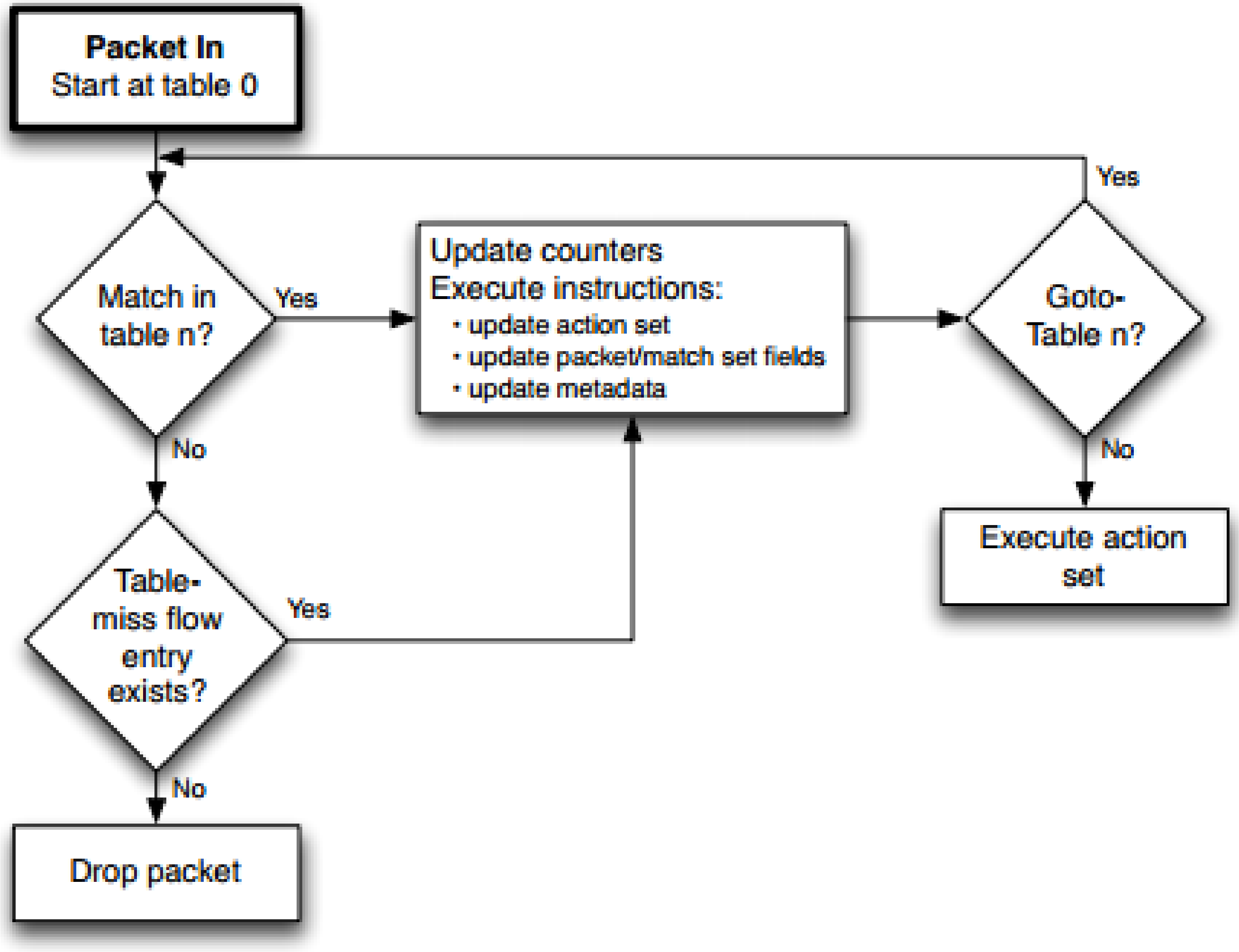
Flow Switching

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

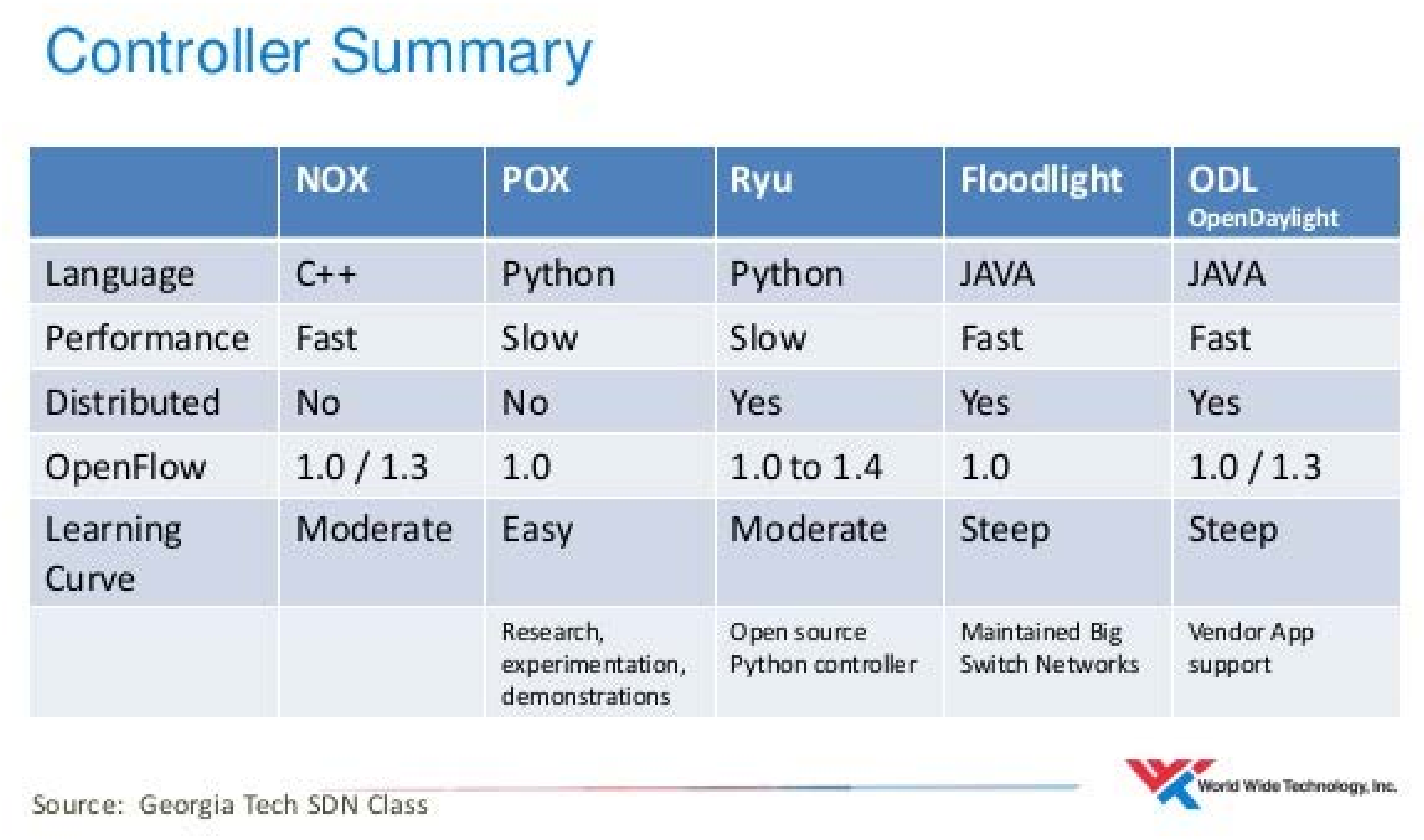
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

OpenFlow - Matching



OpenFlow Controllers

OpenFlow Controllers



Controller Summary

	NOX	POX	Ryu	Floodlight	ODL OpenDaylight
Language	C++	Python	Python	JAVA	JAVA
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0 / 1.3	1.0	1.0 to 1.4	1.0	1.0 / 1.3
Learning Curve	Moderate	Easy	Moderate	Steep	Steep
		Research, experimentation, demonstrations	Open source Python controller	Maintained Big Switch Networks	Vendor App support

Source: Georgia Tech SDN Class

…and many more: Beacon, Trema, OpenContrail, POF, etc.

That‘s a Lot of Controllers!?

„There are almost as many controllers for SDNs as there are SDNs“ – Nick Feamster

Which controller should I use for what problem?

Which controller?

Concept?

Architecture?

Programming language and model?

Advantages / Disadvantages?

Learning Curve?

Developing Community?

Type of target network?

NOX [1]

- **The first controller**
  - Open source
  - Stable

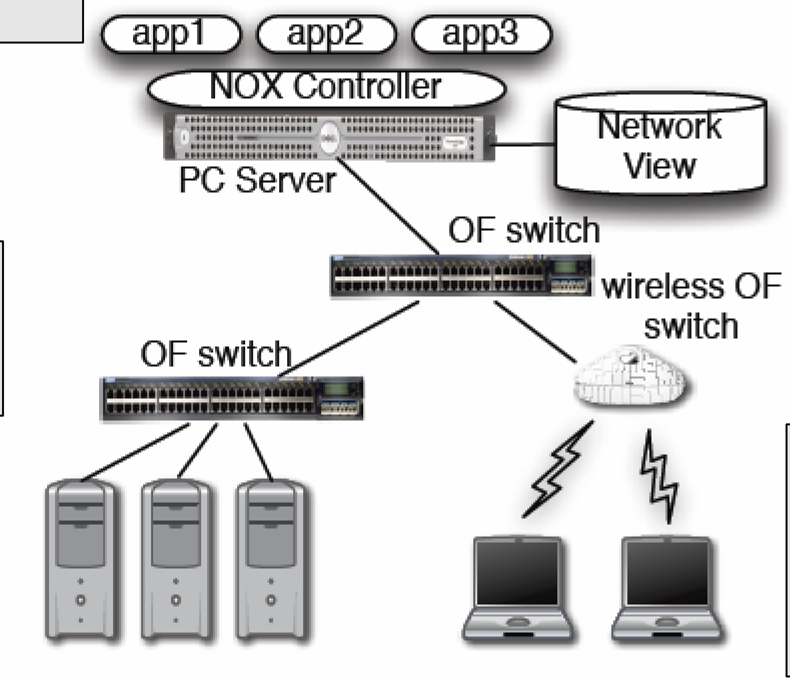
No longer supported

- „New“ NOX: C++ only
  - OF version supported: 1.0

[1] Gude et al. "NOX: towards an operating system for networks." *ACM SIGCOMM CCR* 38.3 (2008): 105-110.

NOX Architecture

Granularity of Control: Per Flow



Controller maintains a network view

switches and attached servers

OpenFlow is used to control switches

[1] Gude et al. "NOX: towards an operating system for networks." *ACM SIGCOMM CCR* 38.3 (2008): 105-110.

NOX Architecture

Programming model: Controller listens for OF events

Programmer writes action handlers for events

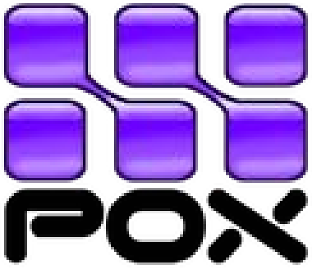
When to use NOX

- Need to use low-level semantics of OpenFlow
  - NOX does not come with many abstractions
- Need of good performance (C++)
  - E.g.: production networks

POX [1]

- **POX = NOX in Python**

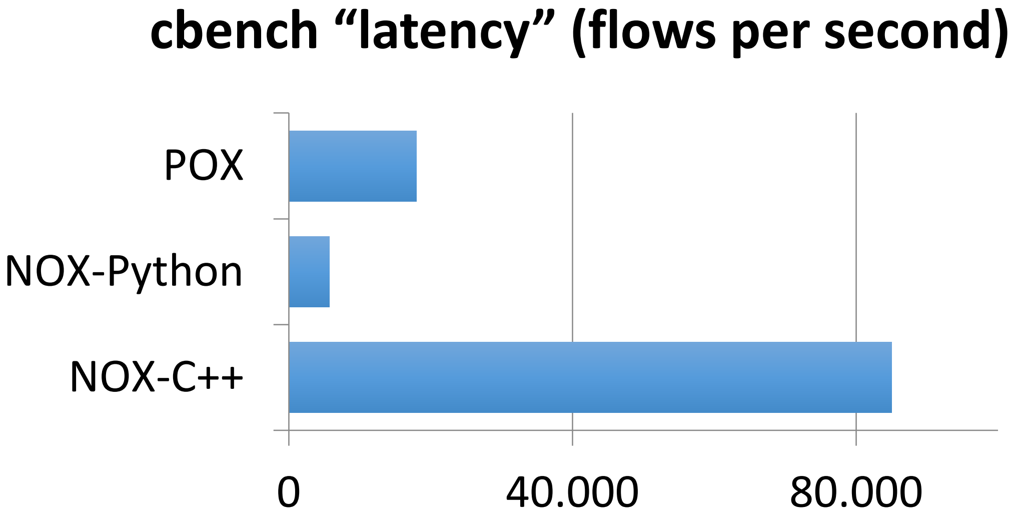
- Advantages:
  - Widely used, maintained and supported
  - Relatively easy to write code for

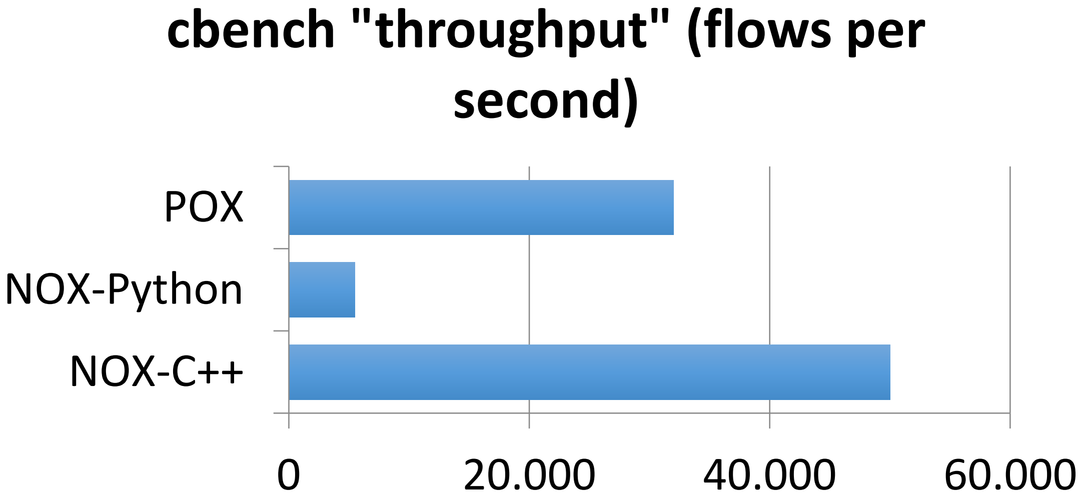


- Disadvantage:
  - Performance (Python is slower than C++)
  - But: can feed POX ideas back to NOX for production use

[1] Mccauley, J. "Pox: A python-based openflow controller.“ http://www.noxrepo.org/pox/about-pox/

POX





http://www.noxrepo.org/pox/about-pox/

When to use POX

- Learning, testing, debugging, evaluation

- Probably not in large production networks

Programming POX

- Recall: controller listens for OF events, here: packetIn

```
def _handle_PacketIn (self, event):
    """
    Handles packet in messages from the switch.
    """

    packet = event.parsed # This is the parsed packet data.
    if not packet.parsed:
      log.warning("Ignoring incomplete packet")
      return

    packet_in = event.ofp # The actual ofp_packet_in message.

    # process packet like a switch
    self.act_like_switch(packet, packet_in)
```

Programming POX

```
def act_like_switch (self, packet, packet_in):
    """
    The controller will check whether or not the destination host
    is in the MAC-TO-PORT table.
    IF that is the case, the controller instructs the switch to
    forward via the corresponding port.
    IF NOT, the controller instructs the switch to flood the packet.
    """


    #update MAC-TO-PORT table for source of packet
    self.mac_to_port[packet.src] =  packet_in.in_port

    if packet.dst in self.mac_to_port:
        out_port = self.mac_to_port[packet.dst]
        # Send packet out the associated port
        self.resend_packet(packet_in, self.mac_to_port[packet.dst])

    else:
        self.resend_packet(packet_in, of.OFPP_ALL)
```

Programming POX

```python
def resend_packet (self, packet_in, out_port):
    """
    Instructs the switch to resend a packet that it had sent to us.
    "packet_in" is the ofp_packet_in object the switch had sent to the
    controller due to a table-miss.
    """
    msg = of.ofp_packet_out()
    msg.data = packet_in

    # Add an action to send to the specified port
    action = of.ofp_action_output(port = out_port)
    msg.actions.append(action)

    # Send message to switch
    self.connection.send(msg)
```

Just one more: Floodlight [1]

- Java

- Advantages:
  - Documentation,
  - REST API conformity
  - Production-level performance

- Disadvantage:
  - Steep learning curve

[1] http://www.projectfloodlight.org/floodlight/

Floodlight: Users

intel
Leland Stanford Junior University 1891
ORACLE
Goldman Sachs
Mellanox Technologies
boundary
Georgia Tech
IBM
FUJITSU
ARMOUR
DELL
BROCADE
Elbrys Networks
NC State
hp
ARISTA
PLURIBUS NETWORKS
Microsoft
CISCO
extreme networks
NEC
LUXOFT Engineering Business Performance
Princeton University
CITRIX
JUNIPer NETWORKS
CLEMSON UNIVERSITY
INDIANA UNIVERSITY
UNIVERSITÄT PASSAU
THALES
UNIVERSITY OF KENTUCKY
MARIST

Floodlight Adopters:

- University research
- Networking vendors
- Users
- Developers / startups

Floodlight Overview

- FloodlightProvider (IFloodlightProviderService)
- TopologyManager (ITopologyManagerService)
- LinkDiscovery (ILinkDiscoveryService)
- Forwarding
- DeviceManager (IDeviceService)
- StorageSource (IStorageSourceService)
- RestServer (IRestApiService)
- StaticFlowPusher (IStaticFlowPusherService)
- VirtualNetworkFilter (IVirtualNetworkFilterService)

- Floodlight is a collection of modules
- Some modules (not all) export services
- All modules in Java
- Rich, extensible REST API

Taken from: Cohen et al, “Software-Defined Networking and the Floodlight Controller”, available at http://de.slideshare.net/openflowhub/floodlight-overview-13938216

Floodlight Overview

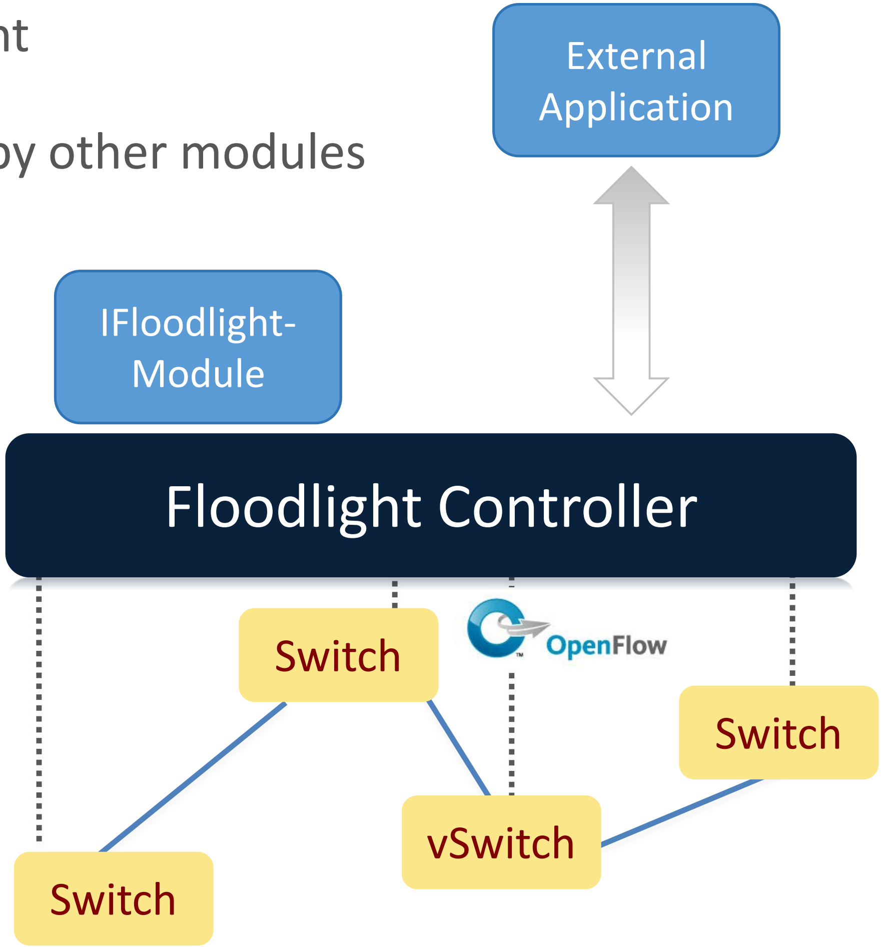
FloodlightProvider (IFloodlightProviderService)	• Translates OF messages to Floodlight events • Managing connections to switches via Netty
TopologyManager (ITopologyManagerService)	• Computes shortest path using Dijsktra • Keeps switch to cluster mappings
LinkDiscovery (ILinkDiscoveryService)	• Maintains state of links in network • Sends out LLDPs
Forwarding	• Installs flow mods for end-to-end routing • Handles island routing
DeviceManager (IDeviceService)	• Tracks hosts on the network • MAC -> switch,port, MAC->IP, IP->MAC
StorageSource (IStorageSourceService)	
RestServer (IRestApiService)	• Implements via Restlets (restlet.org) • Modules export RestletRoutable
StaticFlowPusher (IStaticFlowPusherService)	• Supports the insertion and removal of static flows • REST-based API
VirtualNetworkFilter (IVirtualNetworkFilterService)	• Create layer 2 domain defined by MAC address

Taken from: Cohen et al, "Software-Defined Networking and the Floodlight Controller", available at http://de.slideshare.net/openflowhub/floodlight-overview-13938216

Floodlight Programming Model

IFloodlightModule

- Java module that runs as part of Floodlight
- Consumes services and events exported by other modules
    - OpenFlow (ie. Packet-in)
    - Switch add / remove
    - Device add /remove / move
    - Link discovery

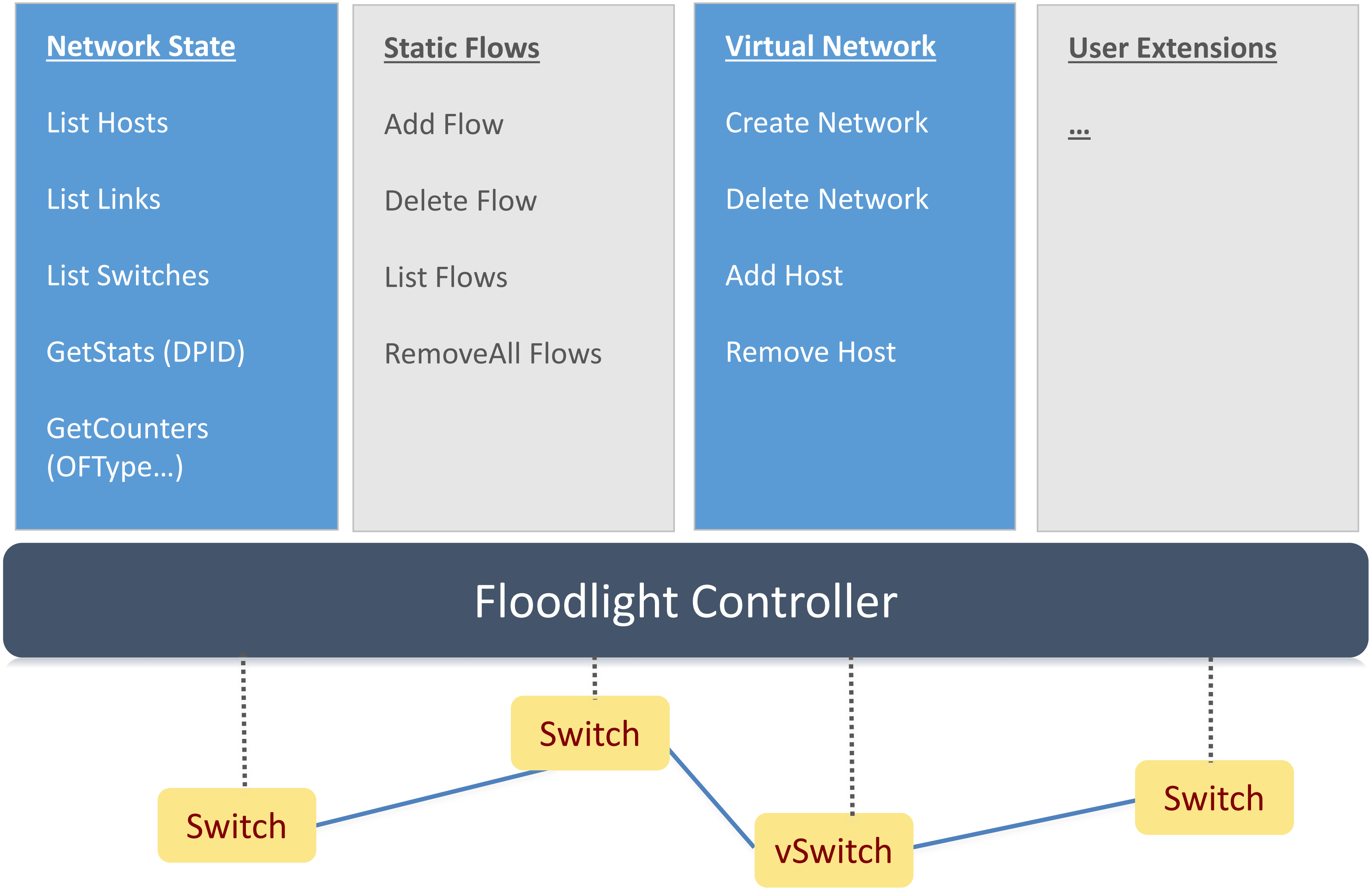


External Application

- Communicates with Floodlight via REST

Taken from: Cohen et al, “Software-Defined Networking and the Floodlight Controller”, available at http://de.slideshare.net/openflowhub/floodlight-overview-13938216

Floodlight Modules



Network State

List Hosts

List Links

List Switches

GetStats (DPID)

GetCounters (OFType…)

Static Flows

Add Flow

Delete Flow

List Flows

RemoveAll Flows

Virtual Network

Create Network

Delete Network

Add Host

Remove Host

User Extensions

…

Floodlight Controller

Taken from: Cohen et al, "Software-Defined Networking and the Floodlight Controller", available at http://de.slideshare.net/openflowhub/floodlight-overview-13938216

When to use Floodlight

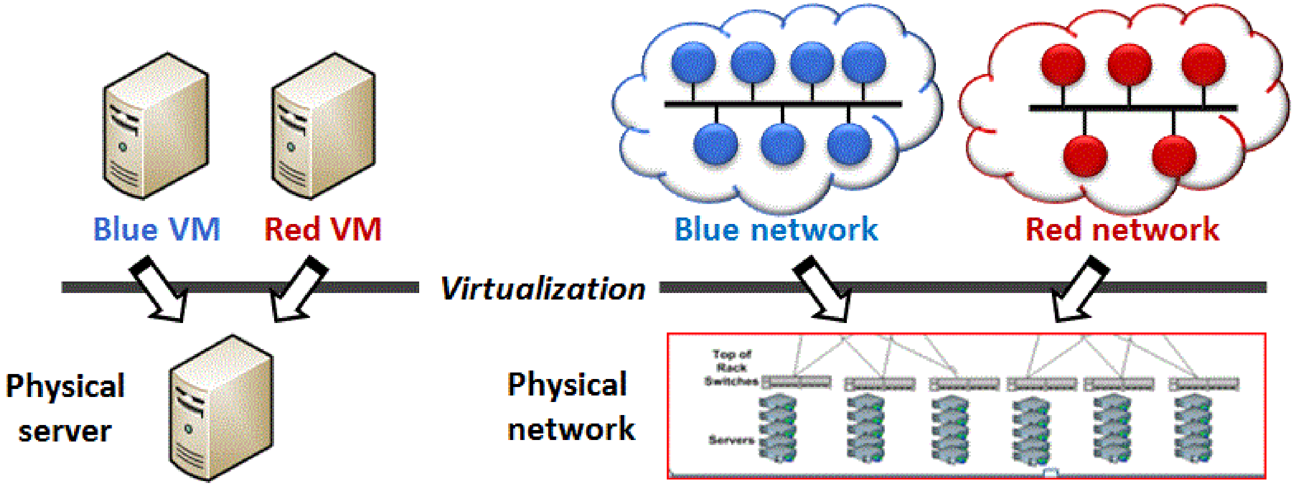
- If you know JAVA
- If you need production-level performance
- Have/want to use REST API

Network Virtualization with OpenFlow

Virtualizing OpenFlow

- Network operators “Delegate” control of subsets of network hardware and/or traffic to other network operators or users
- Multiple controllers can talk to the same set of switches
- Imagine a hypervisor for network equipments
- Allow experiments to be run on the network in isolation of each other and production traffic

Virtualizing OpenFlow



Server virtualization

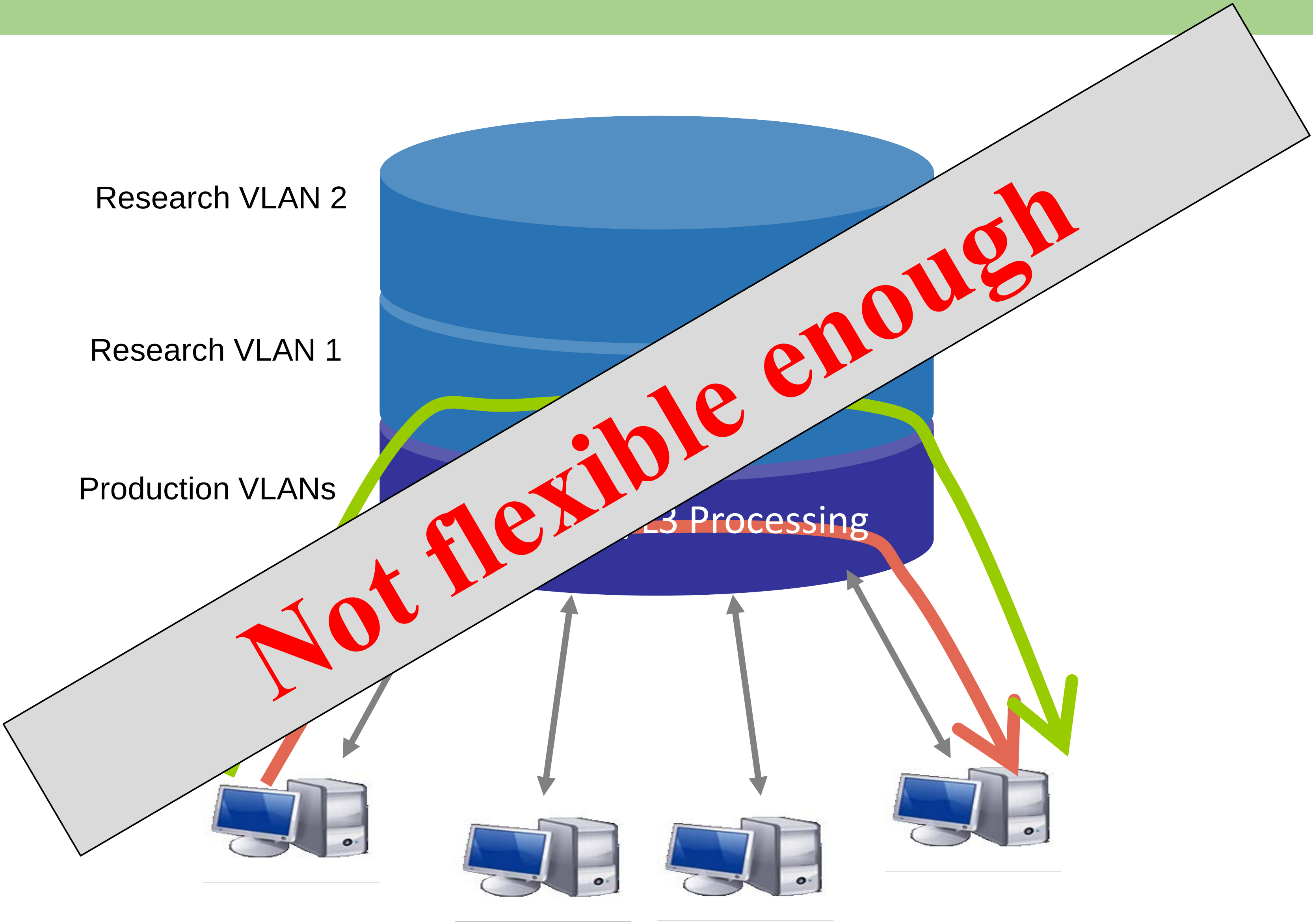
- Run multiple virtual servers on a physical server
- Each VM has illusion it is running as a physical server

Network virtualization

- Run multiple virtual networks on a physical network
- Each virtual network has illusion it is running as a physical network

https://gallery.technet.microsoft.com/scriptcenter/Simple-Hyper-V-Network-d3efb3b8

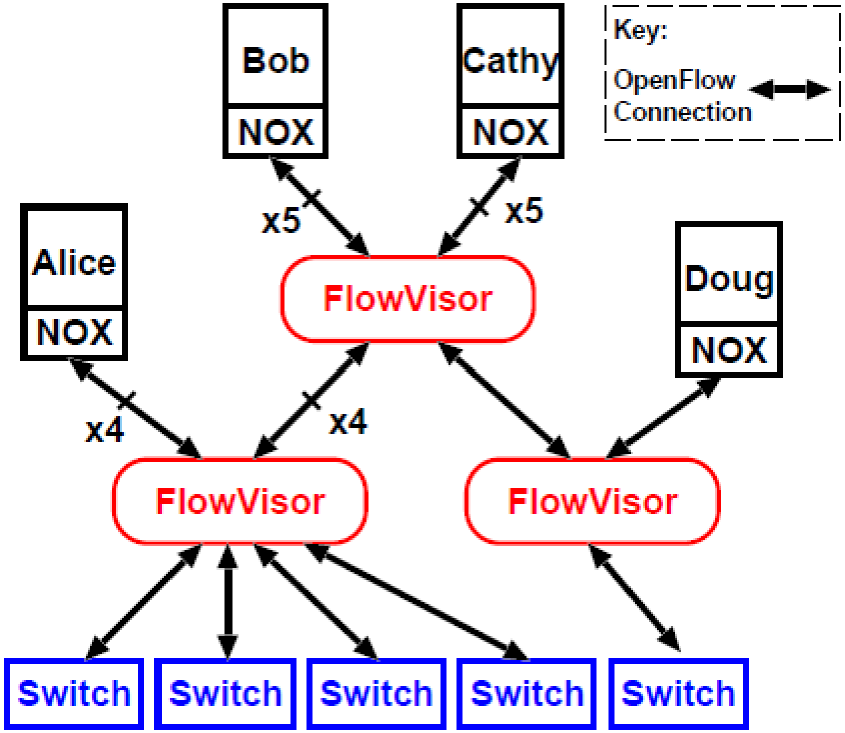
Virtualization: VLANs



Not flexible enough

FlowVisor [1]

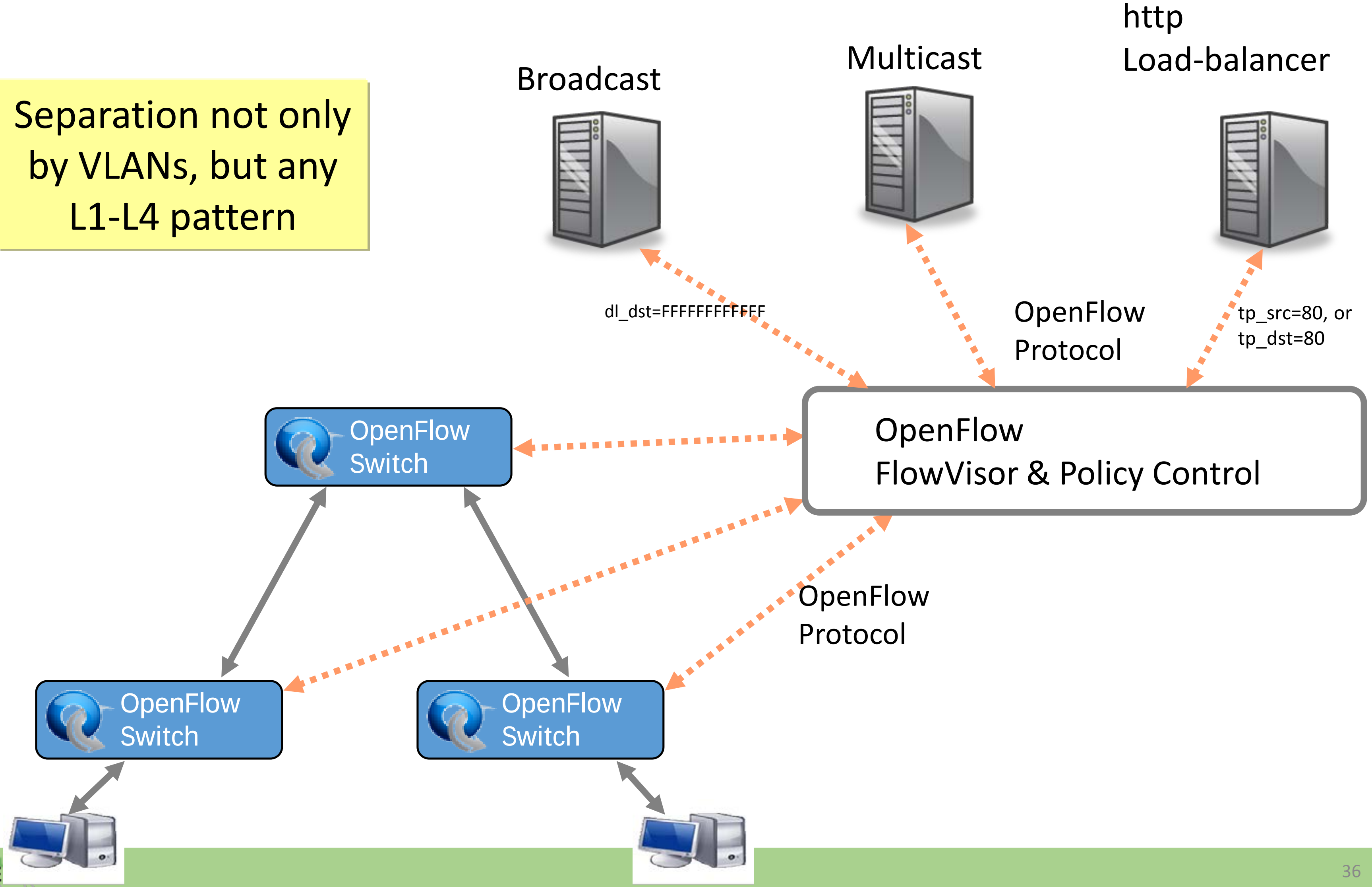
- A network hypervisor developed by Stanford
- A software proxy between the forwarding and control planes of network devices



[1] Sherwood, et al. *"Flowvisor: A network virtualization layer." OpenFlow Switch Consortium, Tech. Rep* (2009).

FlowVisor-based Virtualization

Separation not only by VLANs, but any L1-L4 pattern



Slicing Policies

- The policy specifies resource limits for each slice:
  - Link bandwidth
  - Maximum number of forwarding rules
  - Topology
  - Fraction of switch/router CPU
  - *FlowSpace: which packets does the slice control?*

FlowVisor Resource Limits

- FV assigns hardware resources to “Slices”
  - Topology
    - Network Device or Openflow Instance (DPID)
    - Physical Ports
  - Bandwidth
    - Each slice can be assigned a per port queue with a fraction of the total bandwidth

FlowVisor Resource Limits (cont.)

- FV assigns hardware resources to “Slices”
  - CPU
    - Employs Course Rate Limiting techniques to keep new flow events from one slice from overrunning the CPU
  - Forwarding Tables
    - Each slice has a finite quota of forwarding rules per device

FlowVisor FlowSpace

- FlowSpace is defined by a collection of packet headers and assigned to “Slices”
  - Source/Destination MAC address
  - VLAN ID
  - Ethertype
  - IP protocol
  - Source/Destination IP address
  - ToS/DSCP
  - Source/Destination port number

Use Case: VLAN Partitioning

- Basic Idea: Partition Flows based on Ports and VLAN Tags
  - Traffic entering system (e.g. from end hosts) is tagged
  - VLAN tags consistent throughout substrate

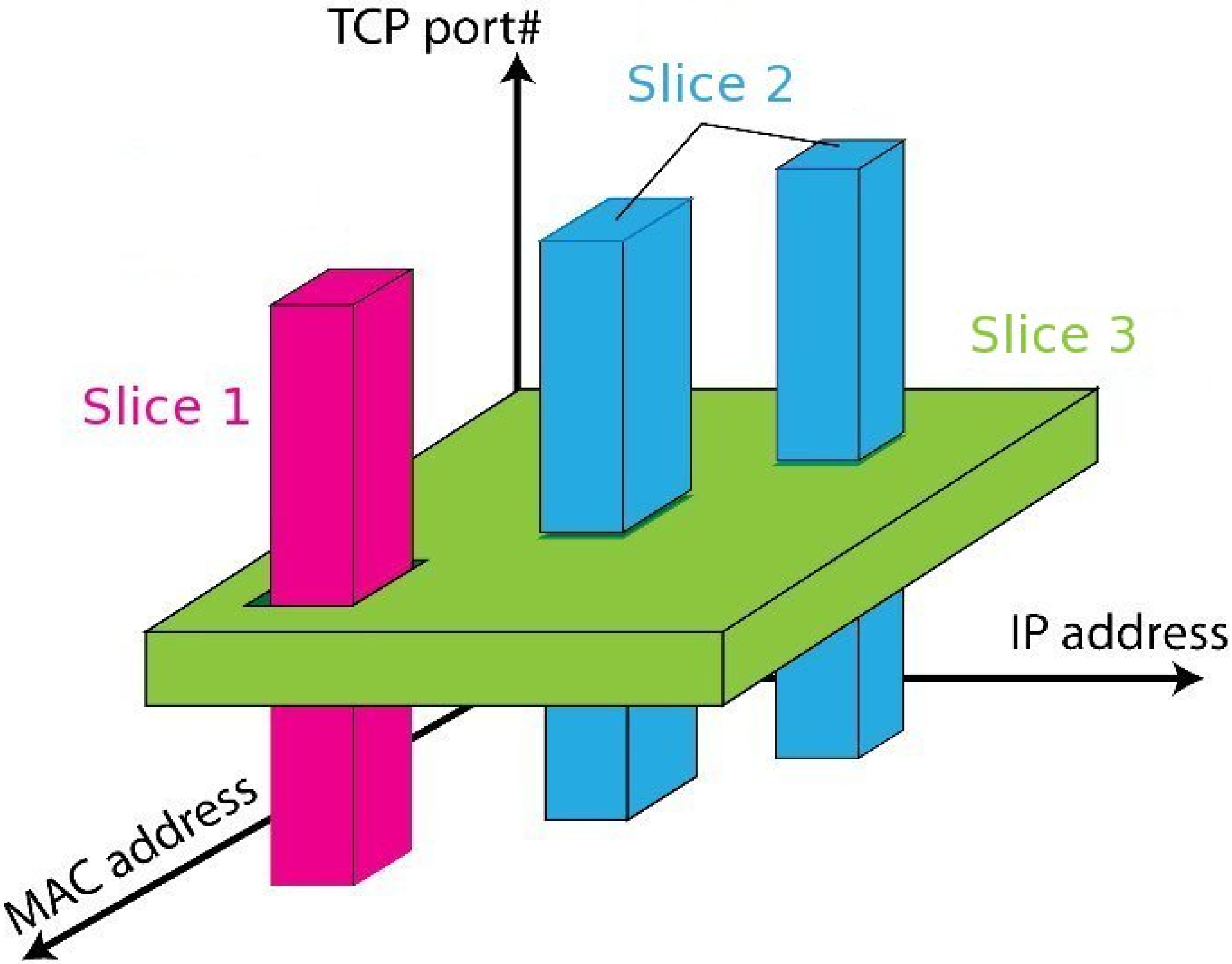
	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
Dave	*	*	*	*	1,2,3	*	*	*	*	*
Larry	*	*	*	*	4,5,6	*	*	*	*	*
Steve	*	*	*	*	7,8,9	*	*	*	*	*

Use Case: Content Distribution Network

- Basic Idea: Build a CDN where you control the entire network
  - All traffic to or from CDN IP space controlled by Experimenter
  - All other traffic controlled by default routing
  - Topology is the entire network

	Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport
From CDN	*	*	*	*	*	84.65.*	*	*	*	*
To CDN	*	*	*	*	*	*	84.65.*	*	*	*
Default	*	*	*	*	*	*	*	*	*	*

FlowSpace: Maps Packets to Slices



Taken from: Rob Sherwood's presentation at ONS:
http://www.opennetsummit.org/archives/apr12/sherwood-mon-flowvisor.pdf

FlowVisor Slicing Policy

- FlowVisor intercepts OpenFlow messages from devices
    - Send control plane messages to the slice controller only if source is in slice topology.
    - Rewrite OpenFlow feature negotiation messages so the slice controller only sees the ports in it’s slice
    - Port up/down messages are pruned and only forwarded to affected slices

FlowVisor Slicing Policy

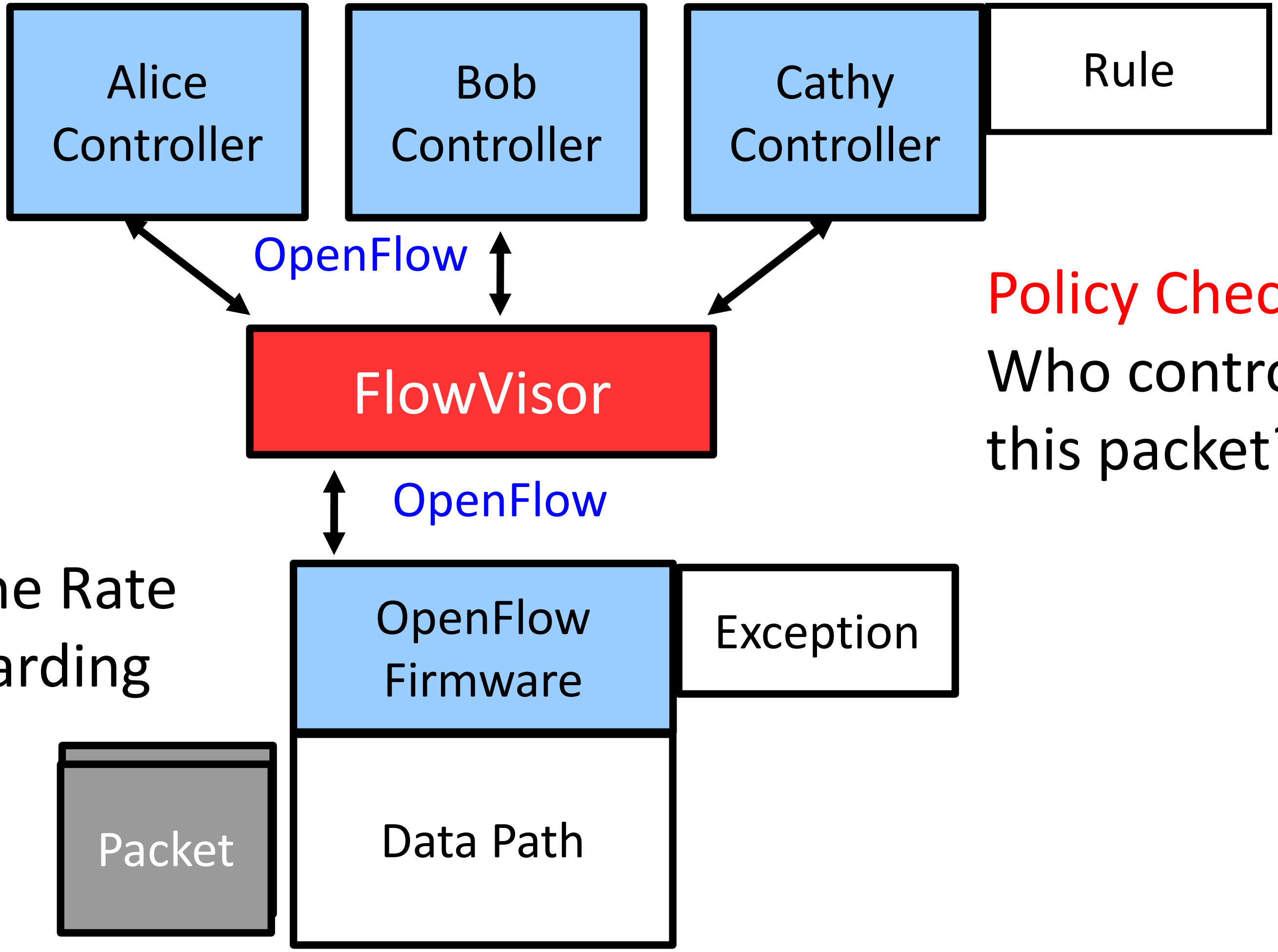
- FlowVisor intercepts OpenFlow messages from controllers
  - Rewrites flow insertion, deletion & modification rules so they don’t violate the slice definition
    - Flow definition – ex. Limit Control to HTTP traffic only
    - Actions – ex. Limit forwarding to only ports in the slice

FlowVisor Slicing Policy

- FlowVisor intercepts OpenFlow messages from controllers
  - Expand Flow rules into multiple rules to fit policy
    - Flow definition – ex. If there is a policy for John’s HTTP traffic and another for Uwe’s HTTP traffic, FV would expand a single rule intended to control all HTTP traffic into 2 rules.
    - Actions – ex. Rule action is send out all ports. FV will create one rule for each port in the slice.

    - Returns “action is invalid” error if trying to control a port outside of the

FlowVisor Message Handling



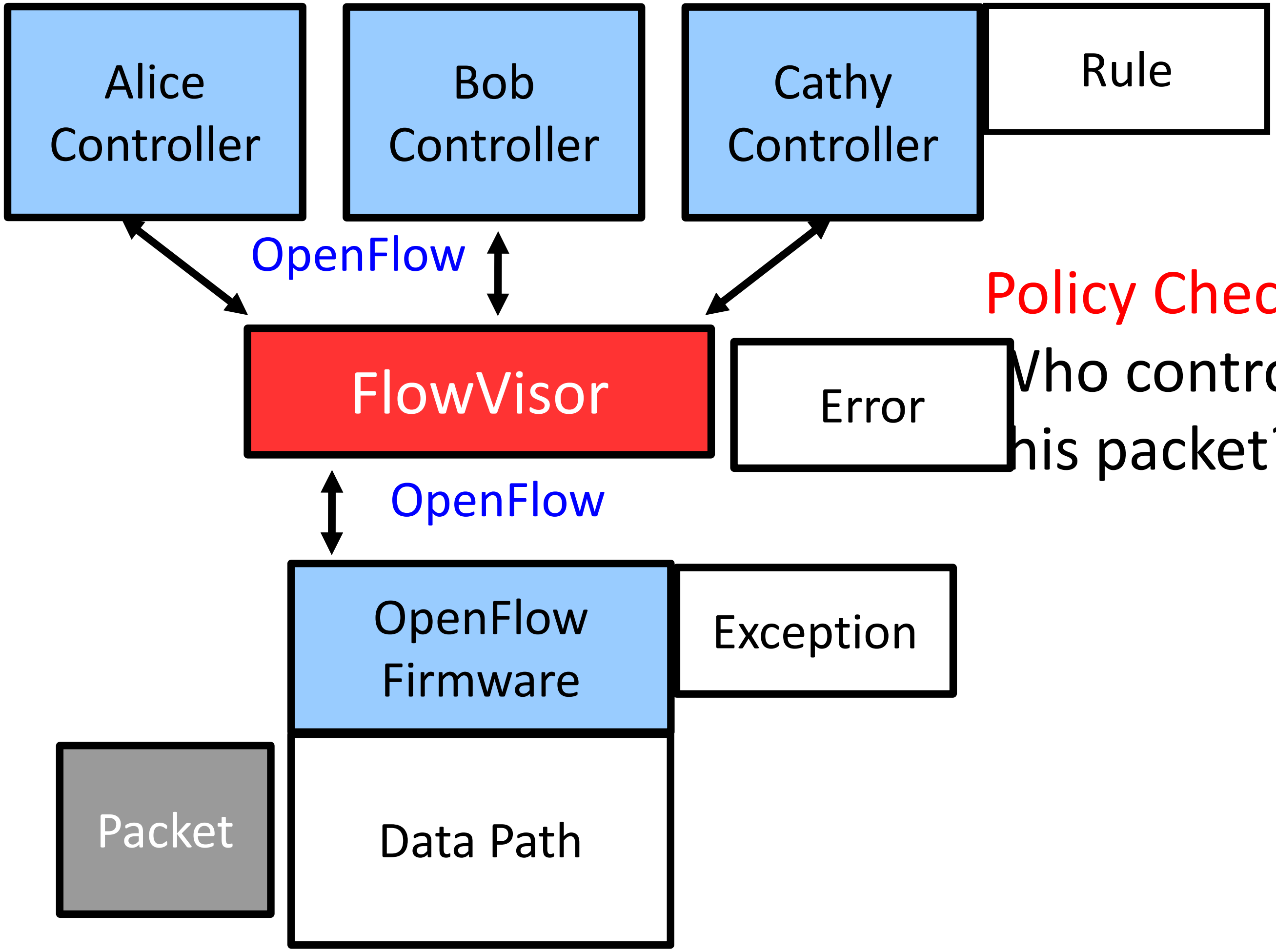
Policy Check:
Is this rule allowed?

Policy Check:
Who controls this packet?

Full Line Rate Forwarding

Taken from: Rob Sherwood's presentation at ONS:
http://www.opennetsummit.org/archives/apr12/sherwood-mon-flowvisor.pdf

FlowVisor Message Handling



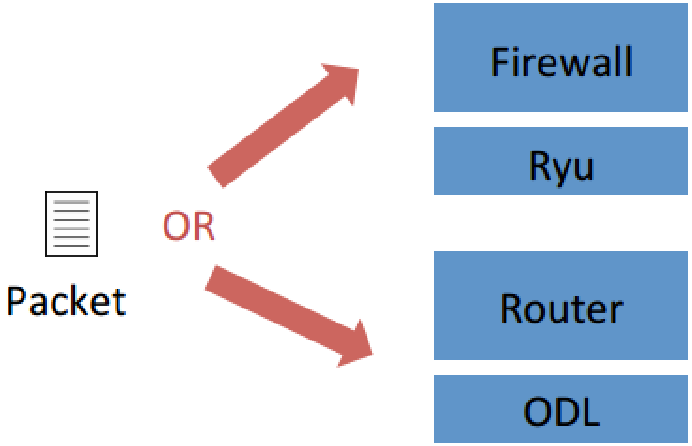
Policy Check:
Is this rule allowed?

Policy Check:
Who controls this packet?

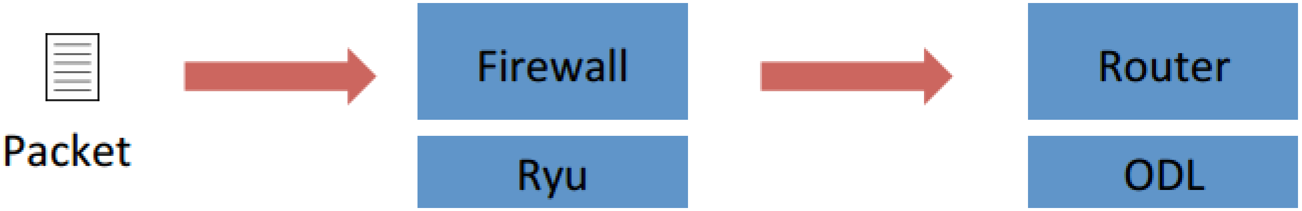
Taken from: Rob Sherwood's presentation at ONS:
http://www.opennetsummit.org/archives/apr12/sherwood-mon-flowvisor.pdf

CoVisor [1]

- FlowVisor allows controllers to work on disjoint slices of traffic only



- How about multiple controllers collaborating on the same traffic?

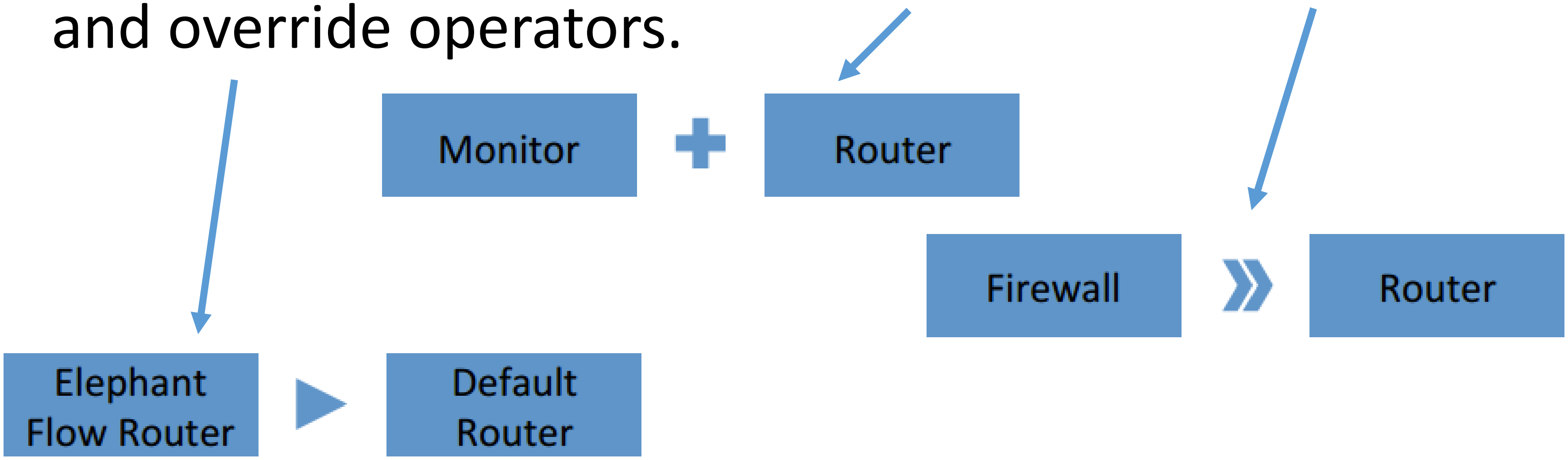


[1] **Jin et al:** “CoVisor: A Compositional Hypervisor for Software-Defined Networks”, *USENIX NSDI 2015*

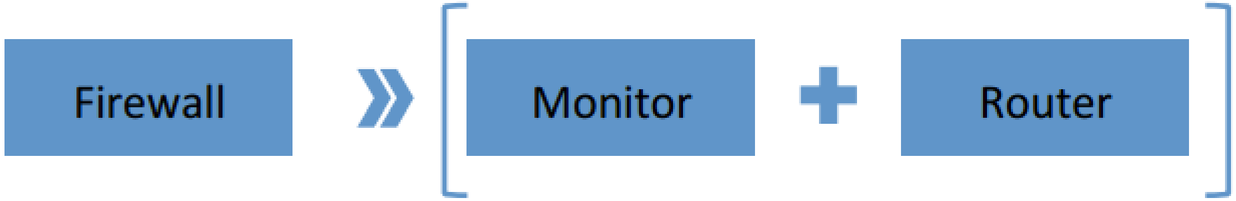
Slides from the presentation at NSDI’15

CoVisor – Controller Composition

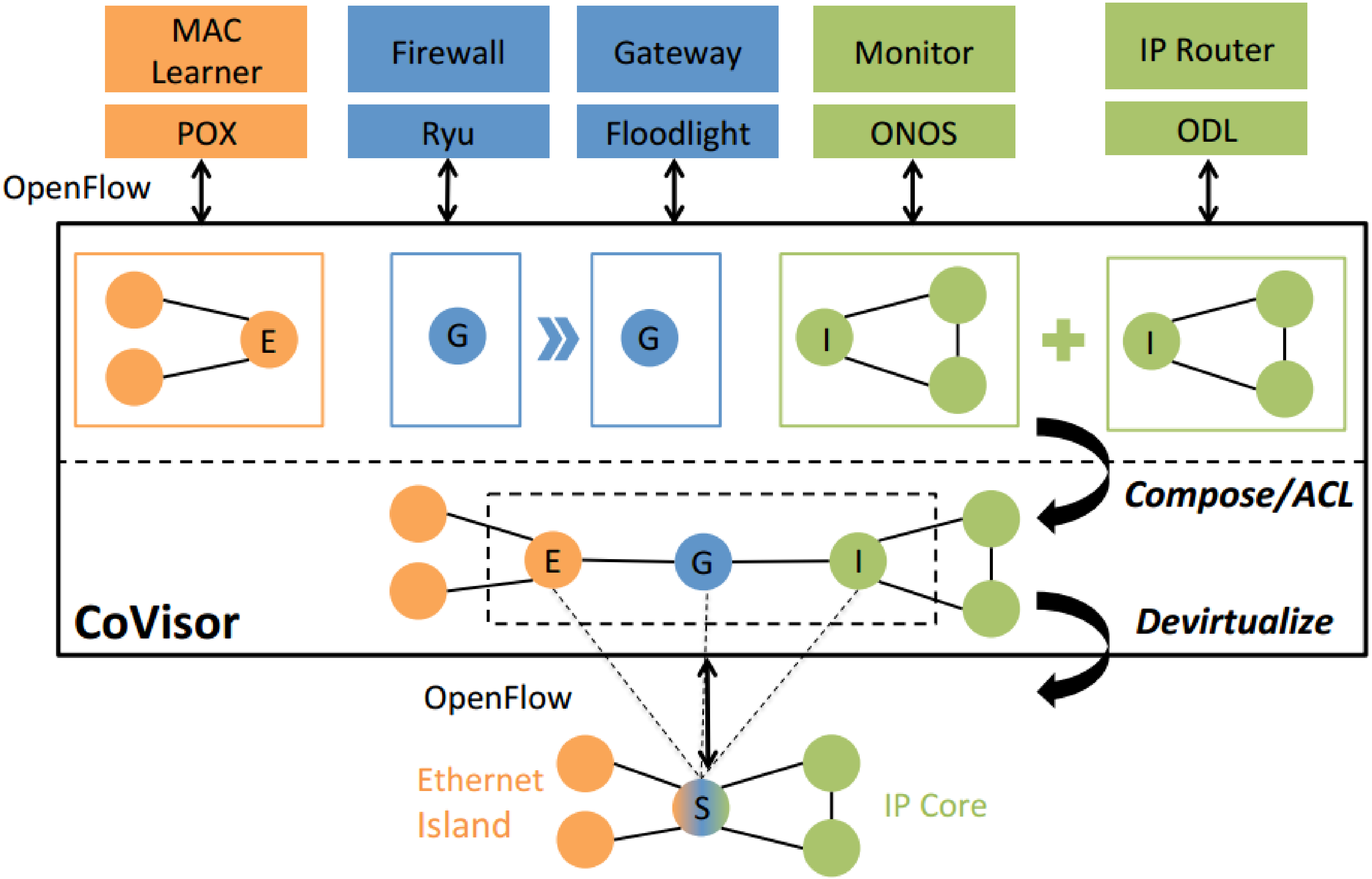
- CoVisor allows combinations of parallel, sequential and override operators.



- Combination:

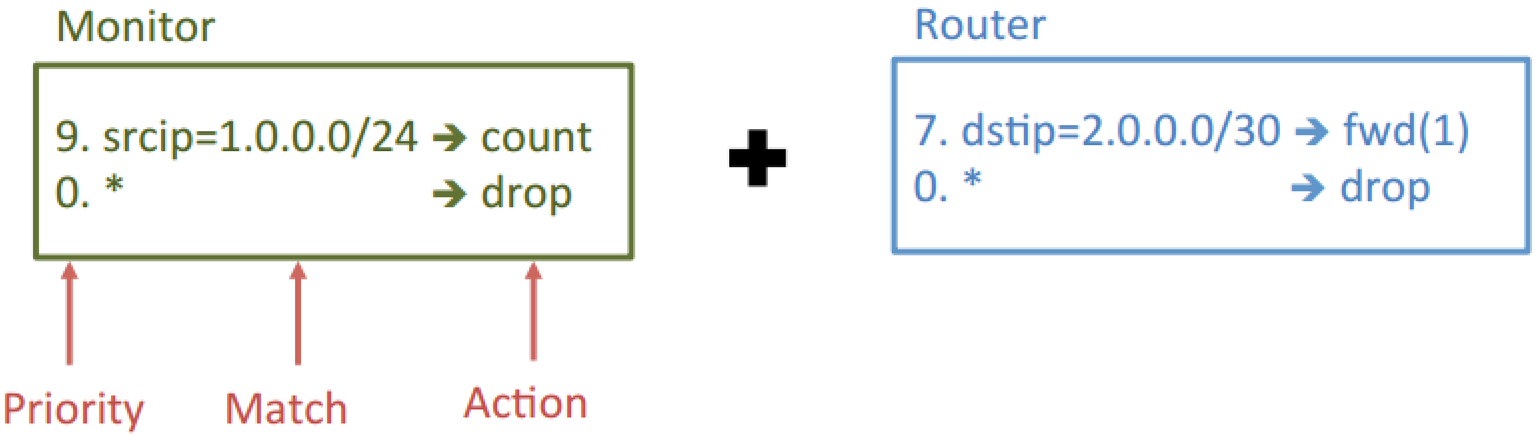


CoVisor – Overview



CoVisor – Policy Composition

- Policy: a list of rules
- Compile policies from controllers to a single policy



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

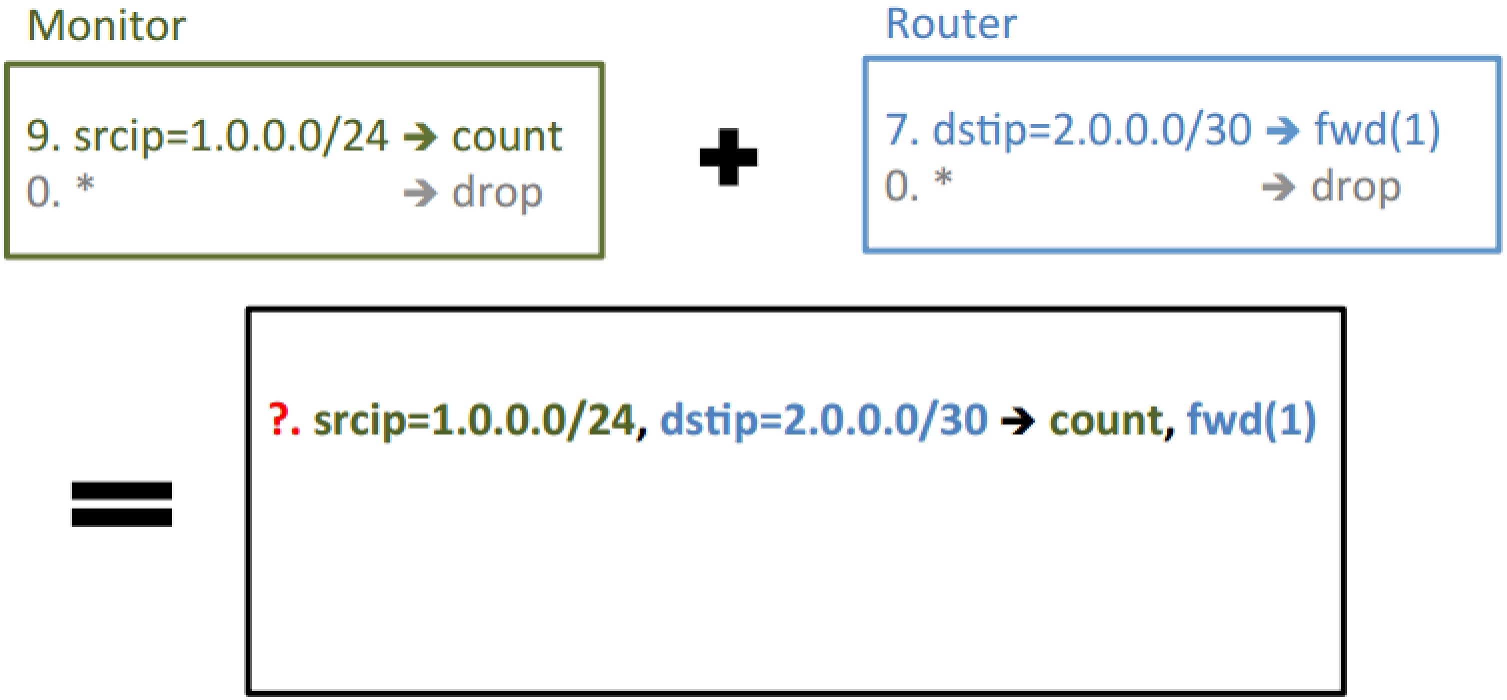
➕

Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

CoVisor – Policy Composition

- Policy: a list of rules
- Compile policies from controllers to a single policy



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

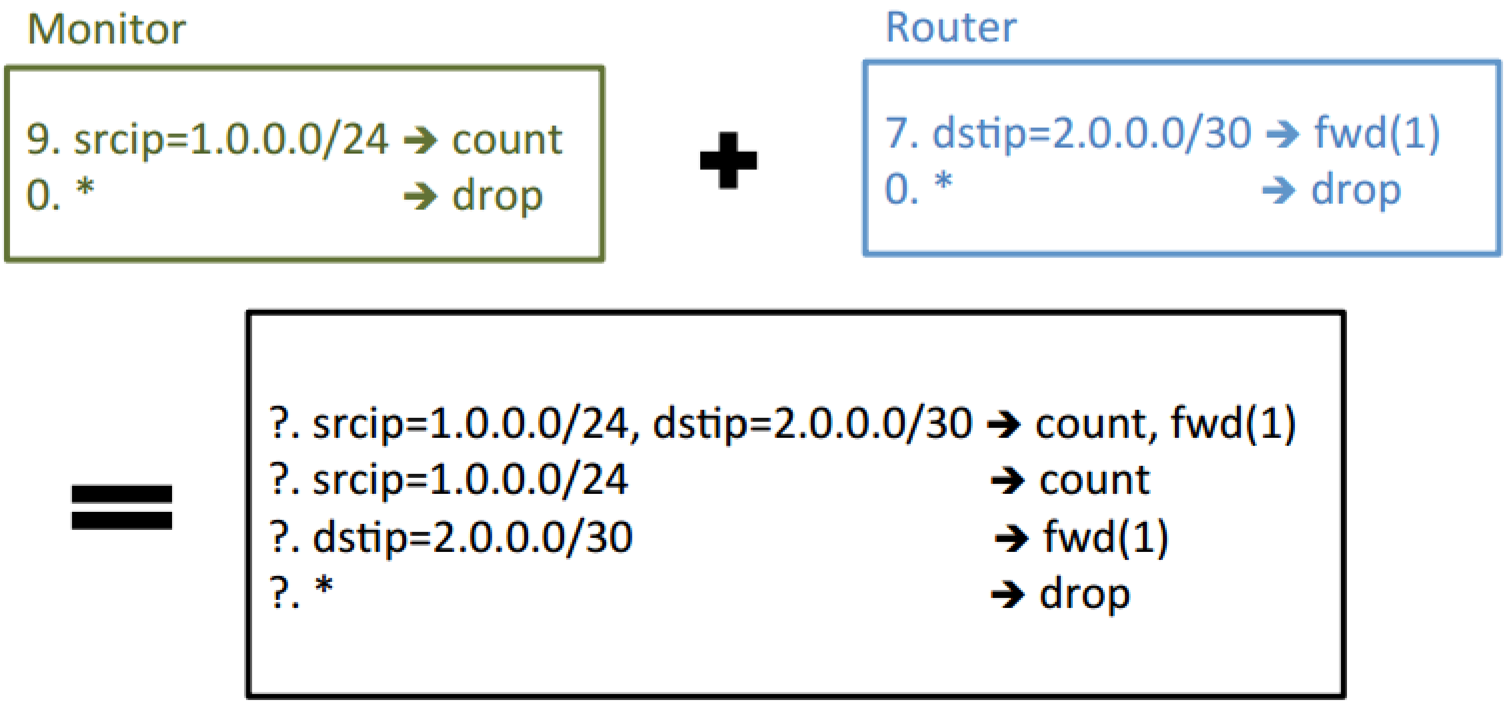
Router

7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)

CoVisor – Policy Composition



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

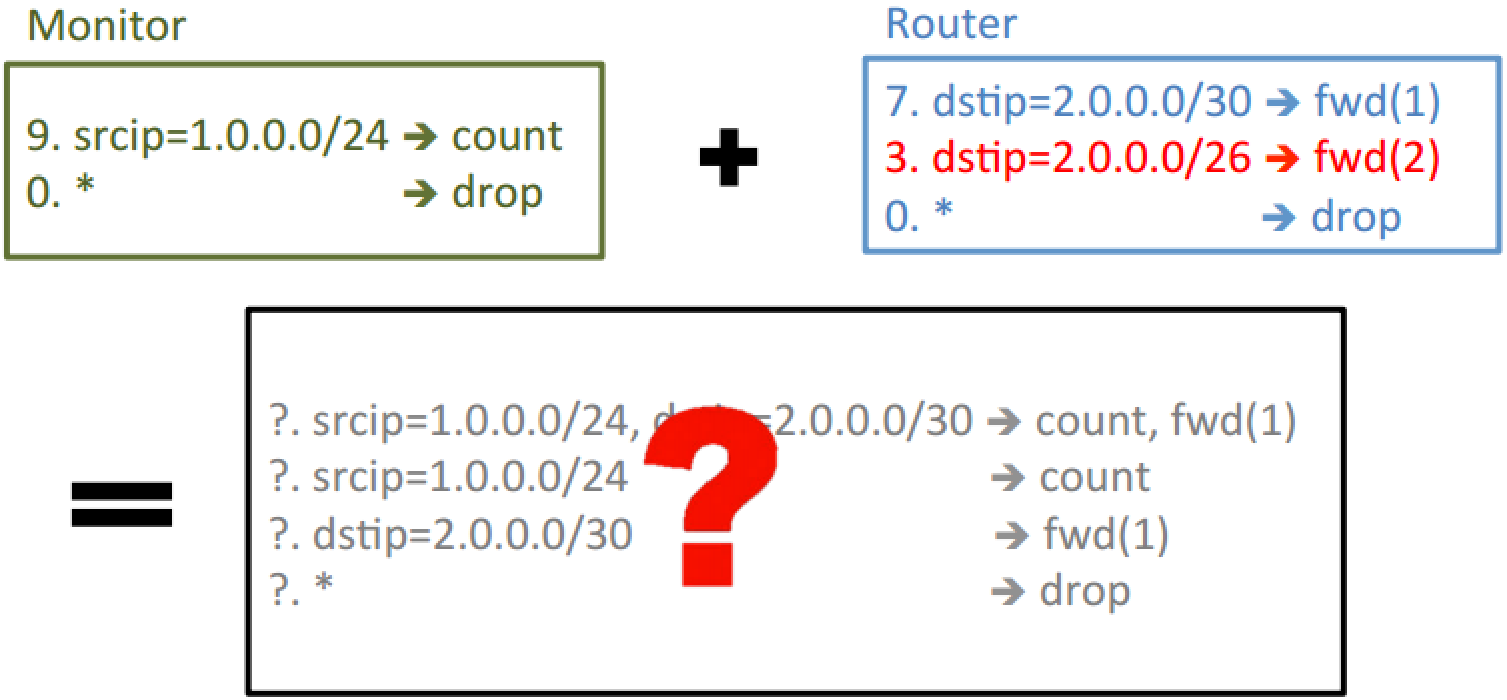
7. dstip=2.0.0.0/30 → fwd(1)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24 → count
?. dstip=2.0.0.0/30 → fwd(1)
?. * → drop

CoVisor – Policy Composition

- Controllers continuously update their policies
- Hypervisor recompiles them and update switches



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

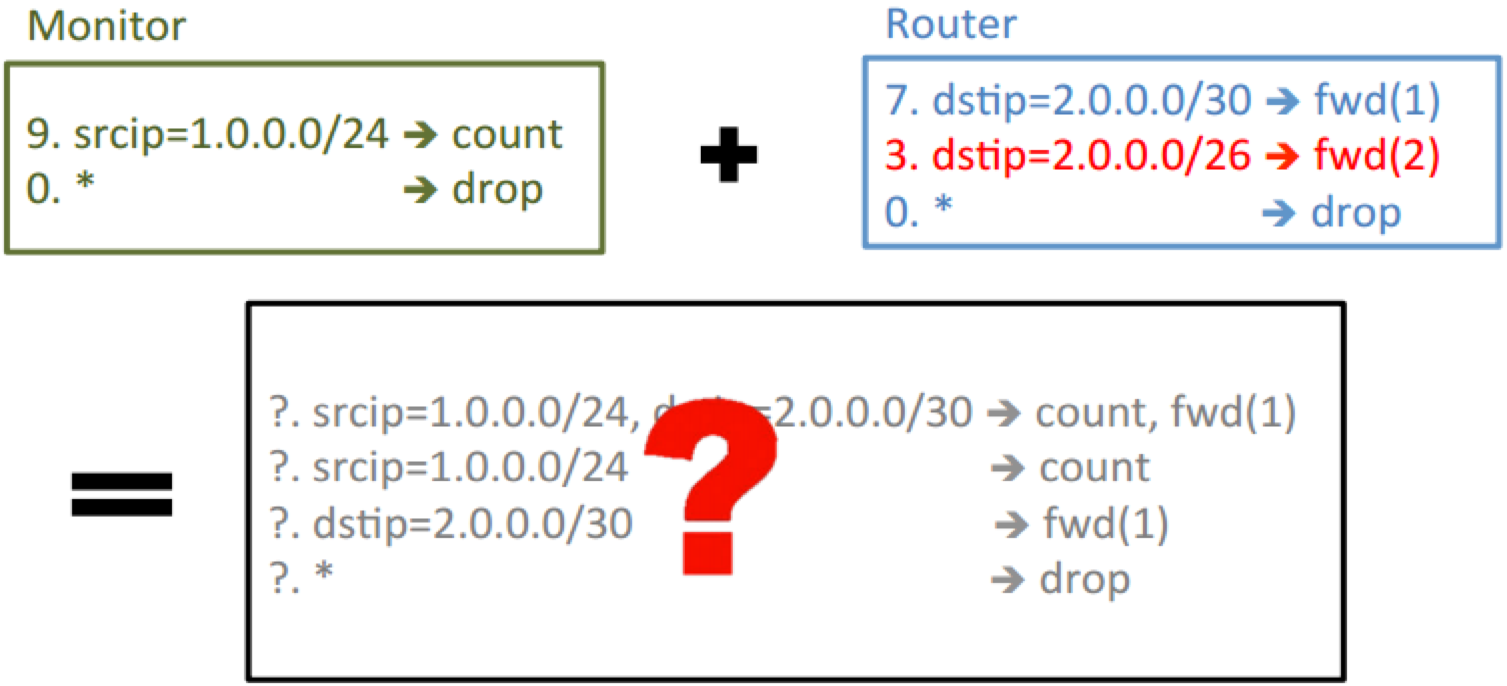
7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

=

?. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24 → count
?. dstip=2.0.0.0/30 → fwd(1)
?. * → drop

CoVisor – Policy Composition

- Computation overhead
  - The computation to recompile the new policy
- Rule-update overhead
  - The rule-updates to update switches to the new policy



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

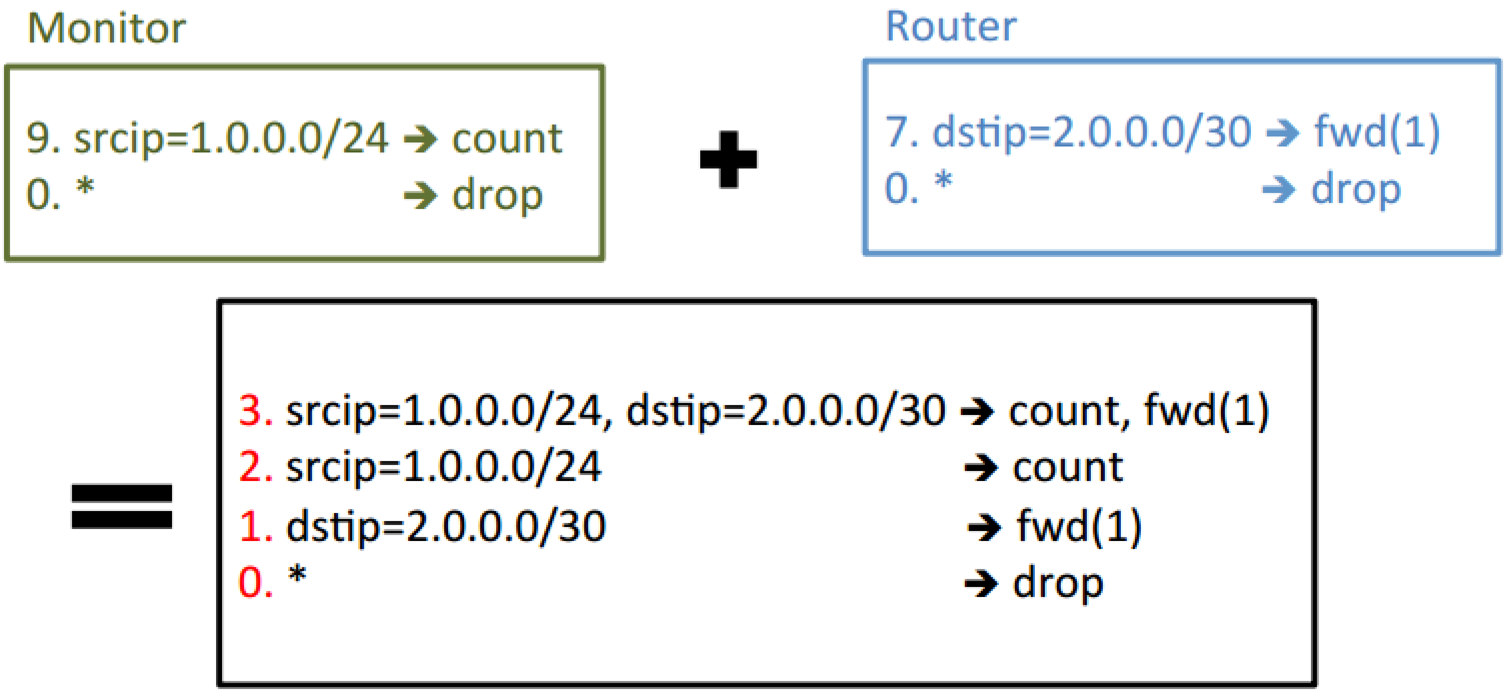
Router

7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

?. srcip=1.0.0.0/24, d[illegible]=2.0.0.0/30 → count, fwd(1)
?. srcip=1.0.0.0/24 → count
?. dstip=2.0.0.0/30 → fwd(1)
?. * → drop

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1



Monitor

```
9. srcip=1.0.0.0/24 → count
0. *                → drop
```

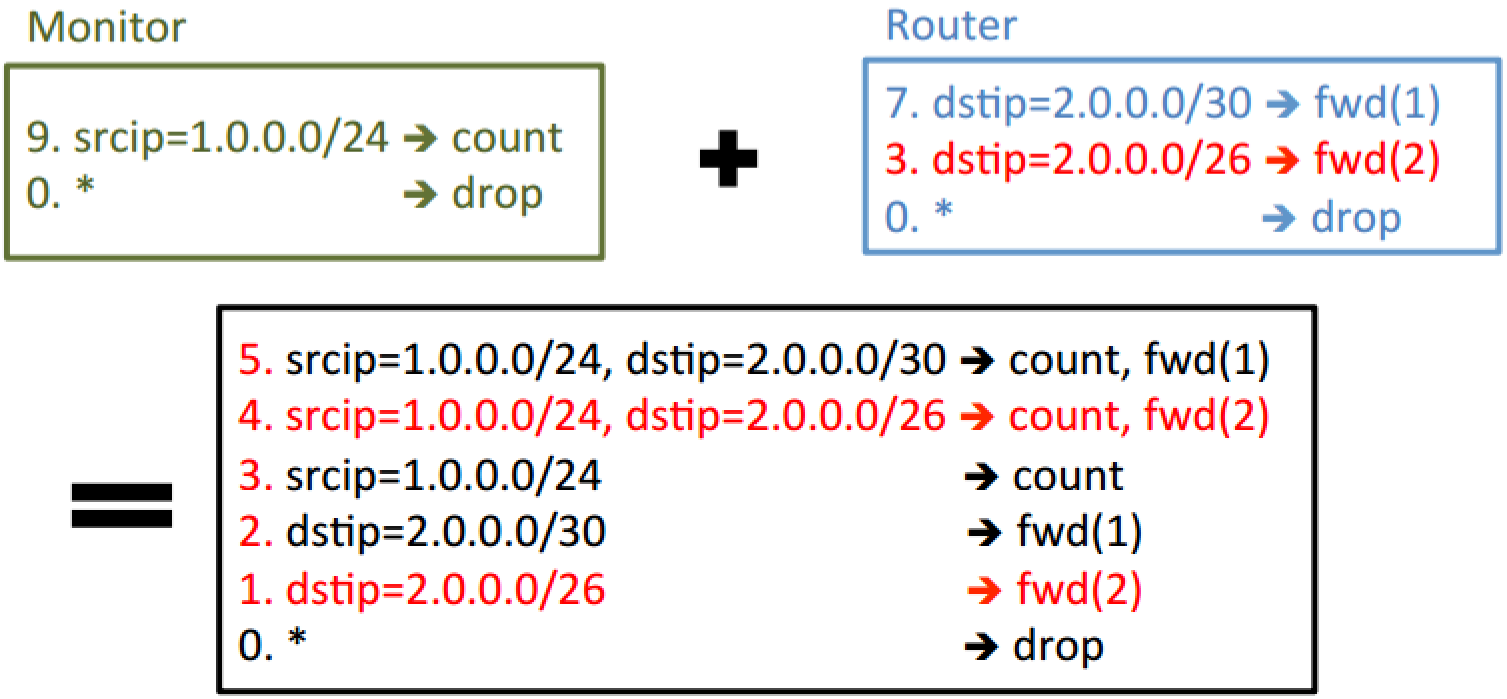
Router

```
7. dstip=2.0.0.0/30 → fwd(1)
0. *                → drop
```

```
3. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
2. srcip=1.0.0.0/24                   → count
1. dstip=2.0.0.0/30                   → fwd(1)
0. *                                  → drop
```

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

Router

7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

5. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
4. srcip=1.0.0.0/24, dstip=2.0.0.0/26 → count, fwd(2)
3. srcip=1.0.0.0/24 → count
2. dstip=2.0.0.0/30 → fwd(1)
1. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

CoVisor – Naïve Policy Composition

- Assign priorities from top to bottom by decrement of 1

```
3. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
2. srcip=1.0.0.0/24                   → count
1. dstip=2.0.0.0/30                   → fwd(1)
0. *                                  → drop
```

Update

```
5. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
4. srcip=1.0.0.0/24, dstip=2.0.0.0/26 → count, fwd(2)
3. srcip=1.0.0.0/24                   → count
2. dstip=2.0.0.0/30                   → fwd(1)
1. dstip=2.0.0.0/26                   → fwd(2)
0. *                                  → drop
```

Computation overhead

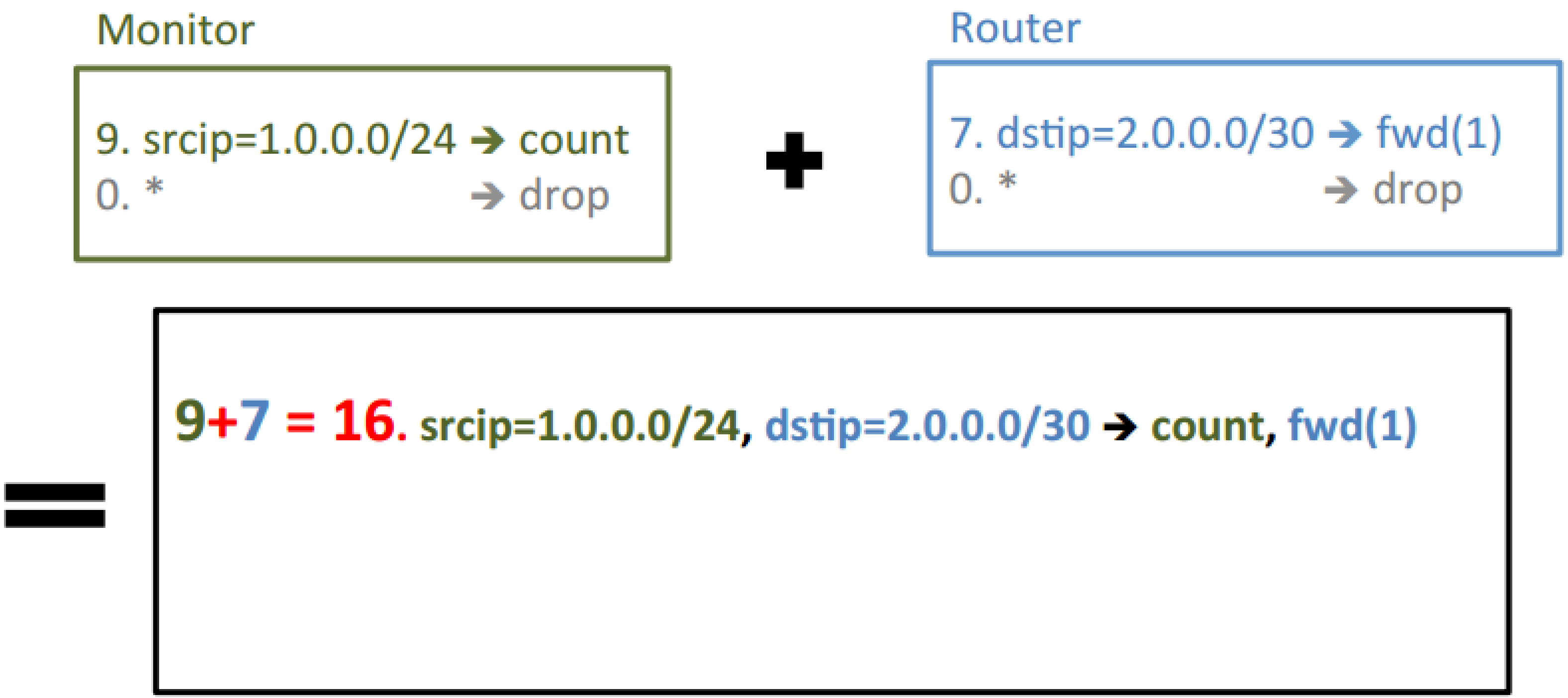
- Recompute the entire switch table and assign priorities

Rule-update overhead

- Only 2 new rules, but 3 more rules change priority

CoVisor – Incremental Solution

- Add priorities for parallel composition



Monitor

```
9. srcip=1.0.0.0/24 → count
0. *                → drop
```

+

Router

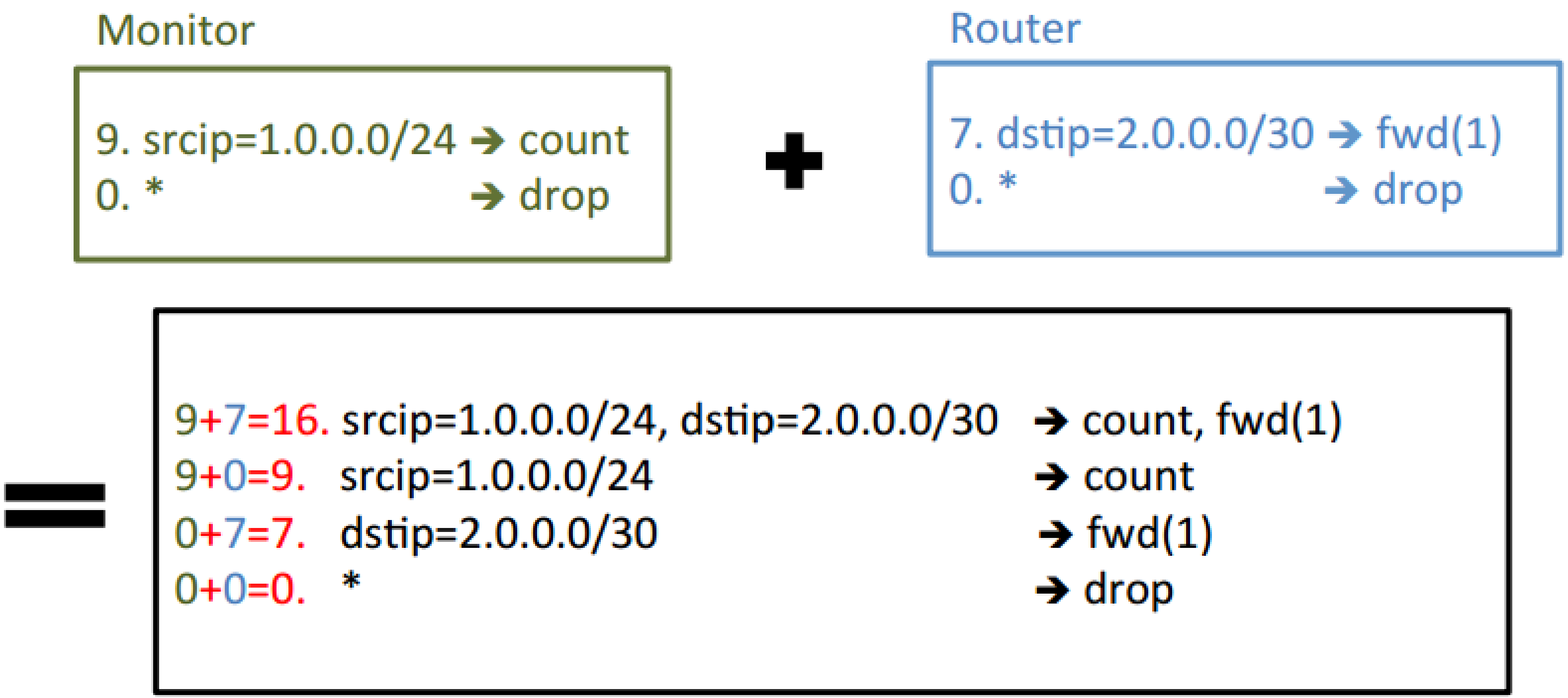
```
7. dstip=2.0.0.0/30 → fwd(1)
0. *                → drop
```

=

9+7 = 16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)

CoVisor – Incremental Solution

- Add priorities for parallel composition



Monitor

```
9. srcip=1.0.0.0/24 → count
0. *                → drop
```

+

Router

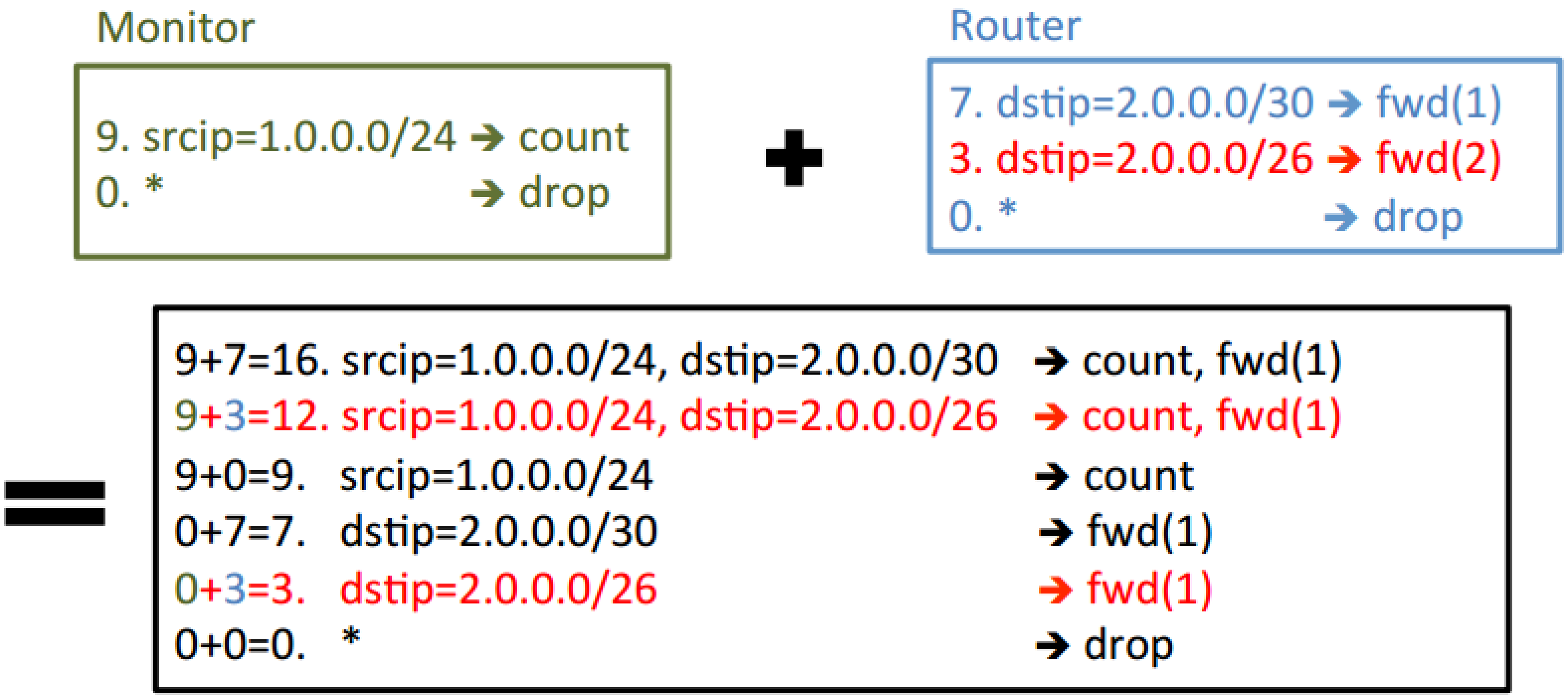
```
7. dstip=2.0.0.0/30 → fwd(1)
0. *                → drop
```

=

```
9+7=16. srcip=1.0.0.0/24, dstip=2.0.0.0/30  → count, fwd(1)
9+0=9.  srcip=1.0.0.0/24                    → count
0+7=7.  dstip=2.0.0.0/30                    → fwd(1)
0+0=0.  *                                   → drop
```

CoVisor – Incremental Solution

- Add priorities for parallel composition



Monitor

9. srcip=1.0.0.0/24 → count
0. * → drop

+

Router

7. dstip=2.0.0.0/30 → fwd(1)
3. dstip=2.0.0.0/26 → fwd(2)
0. * → drop

=

9+7=16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
9+3=12. srcip=1.0.0.0/24, dstip=2.0.0.0/26 → count, fwd(1)
9+0=9. srcip=1.0.0.0/24 → count
0+7=7. dstip=2.0.0.0/30 → fwd(1)
0+3=3. dstip=2.0.0.0/26 → fwd(1)
0+0=0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition

```
16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
9.  srcip=1.0.0.0/24                   → count
7.  dstip=2.0.0.0/30                   → fwd(1)
0.  *                                  → drop
```

Update

```
16. srcip=1.0.0.0/24, dstip=2.0.0.0/30 → count, fwd(1)
12. srcip=1.0.0.0/24, dstip=2.0.0.0/26 → count, fwd(2)
9.  srcip=1.0.0.0/24                   → count
7.  dstip=2.0.0.0/30                   → fwd(1)
3.  dstip=2.0.0.0/26                   → fwd(2)
0.  *                                  → drop
```

Computation overhead

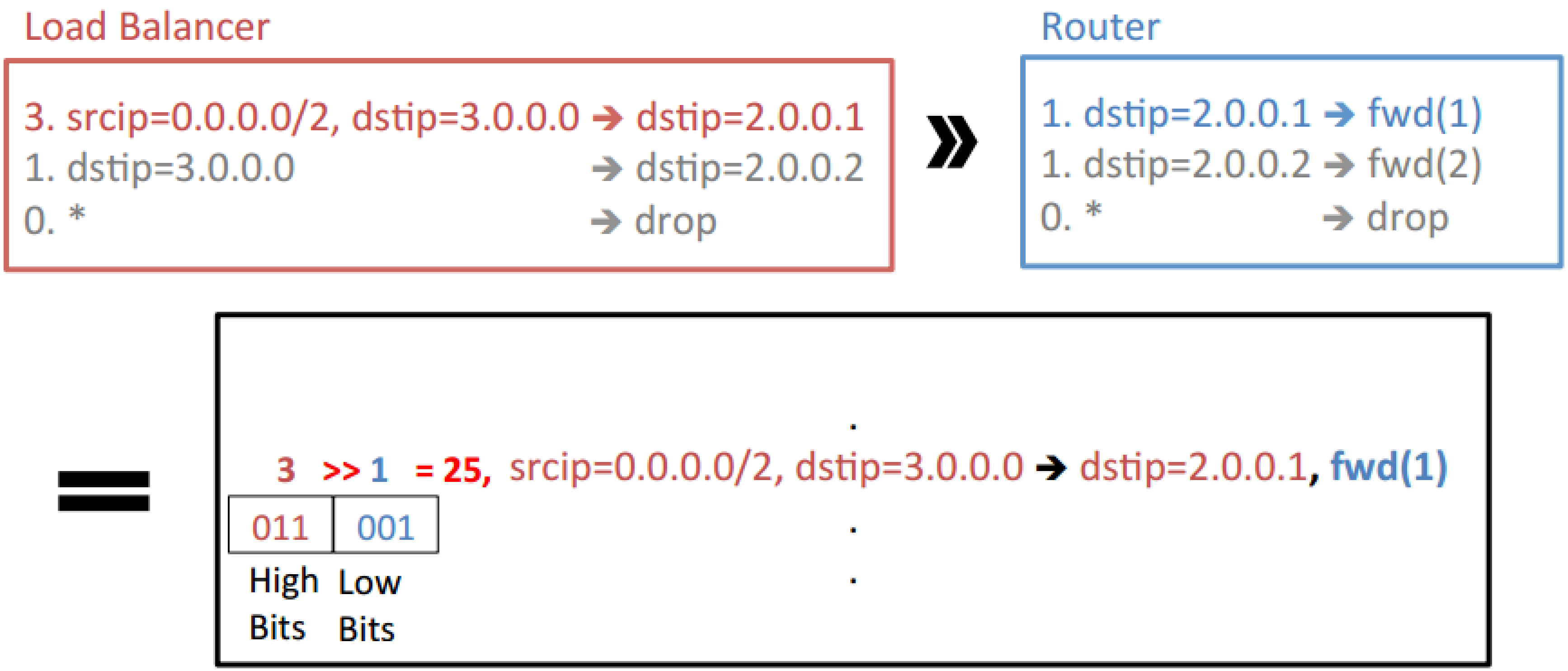
- Only compose the new rule with rules in monitor

Rule-update overhead

- Add 2 new rules

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition



Load Balancer

3. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1
1. dstip=3.0.0.0 → dstip=2.0.0.2
0. * → drop

»

Router

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

.

3 >> 1 = 25, srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1, **fwd(1)**

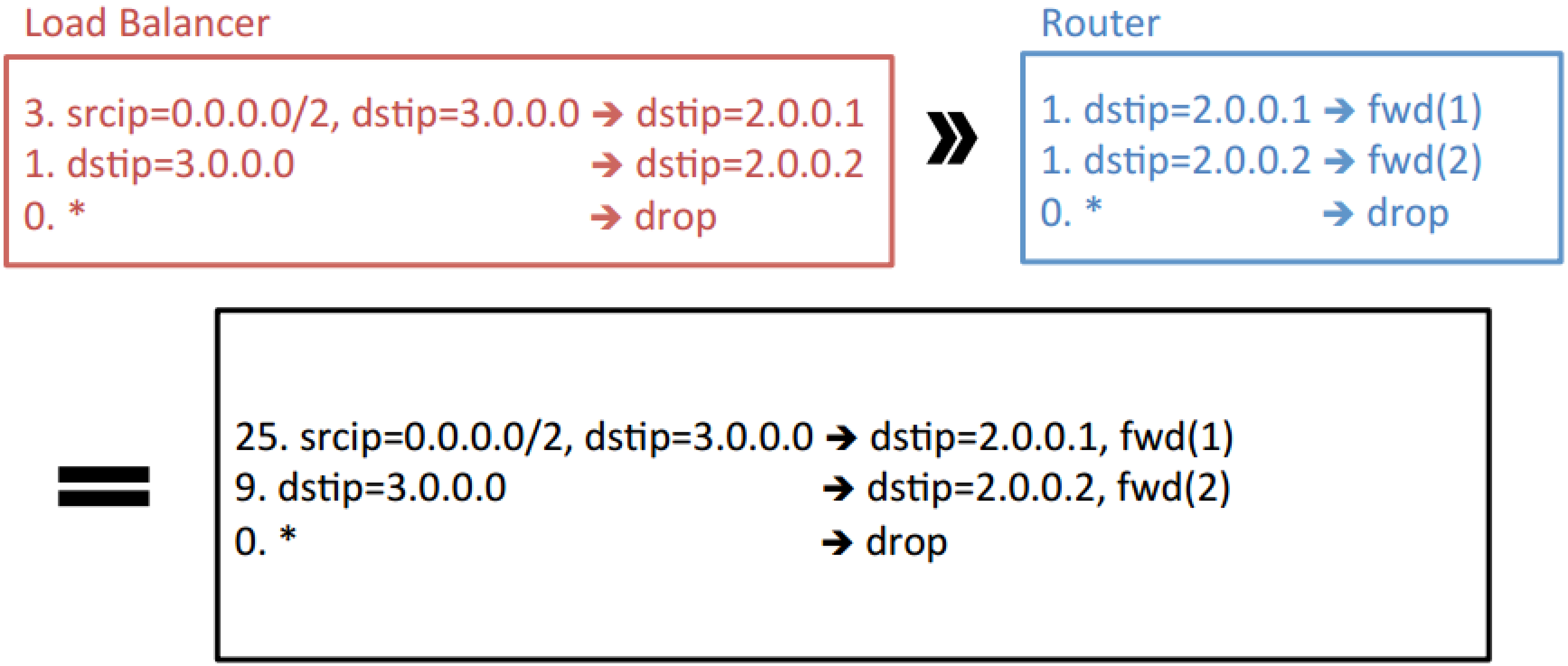
011	001
High Bits	Low Bits

.

.

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition



Load Balancer

3. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1
1. dstip=3.0.0.0 → dstip=2.0.0.2
0. * → drop

»

Router

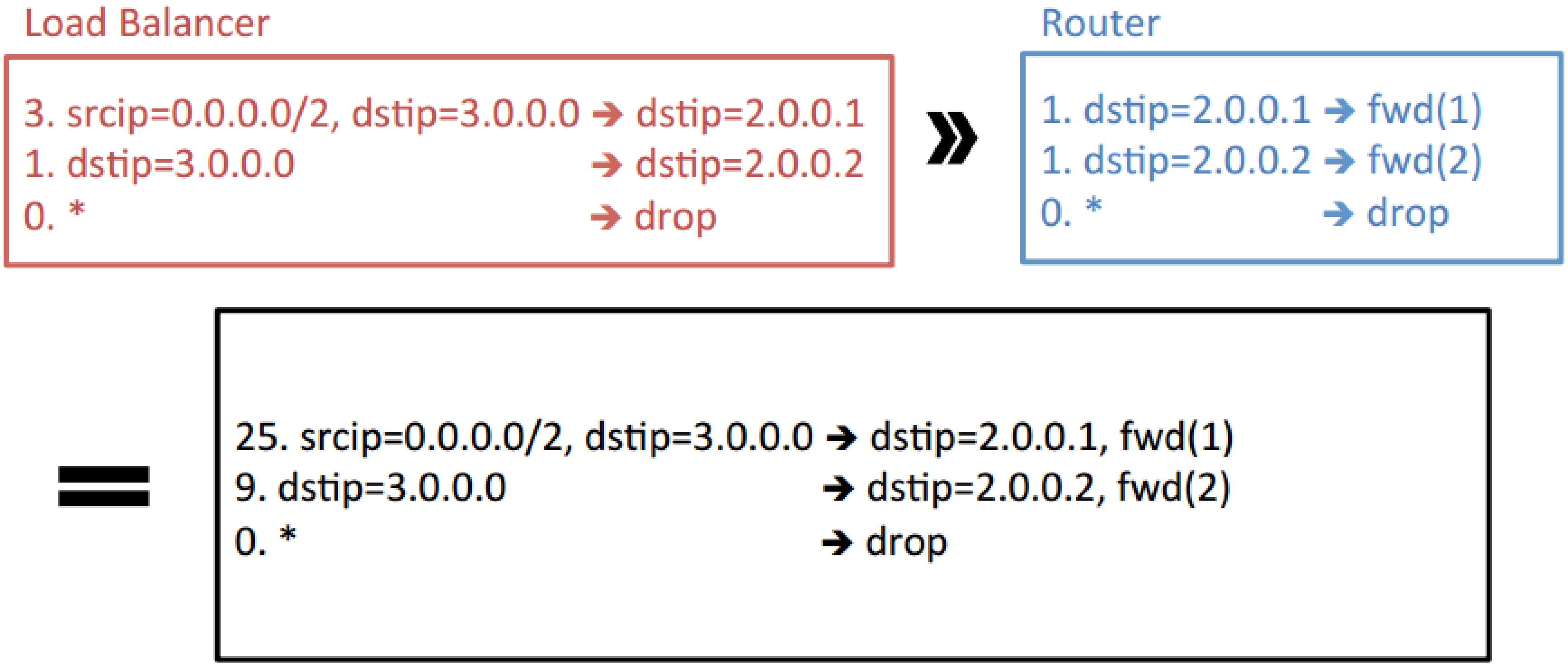
1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

25. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1, fwd(1)
9. dstip=3.0.0.0 → dstip=2.0.0.2, fwd(2)
0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition



Load Balancer

3. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1
1. dstip=3.0.0.0 → dstip=2.0.0.2
0. * → drop

»

Router

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

25. srcip=0.0.0.0/2, dstip=3.0.0.0 → dstip=2.0.0.1, fwd(1)
9. dstip=3.0.0.0 → dstip=2.0.0.2, fwd(2)
0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition
- Stack priorities for override composition

Elephant Flow Router

1. srcip=1.0.0.0, dstip=3.0.0.0 → fwd(3)

▷

Default Router (Max priority = 8)

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

=

1 + 8 = 9. srcip=1.0.0.0, dstip=3.0.0.0 → fwd(3)
1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

CoVisor – Incremental Solution

- Add priorities for parallel composition
- Concatenate priorities for sequential composition
- Stack priorities for override composition

Elephant Flow Router

1. srcip=1.0.0.0, dstip=3.0.0.0 → fwd(3)

▷

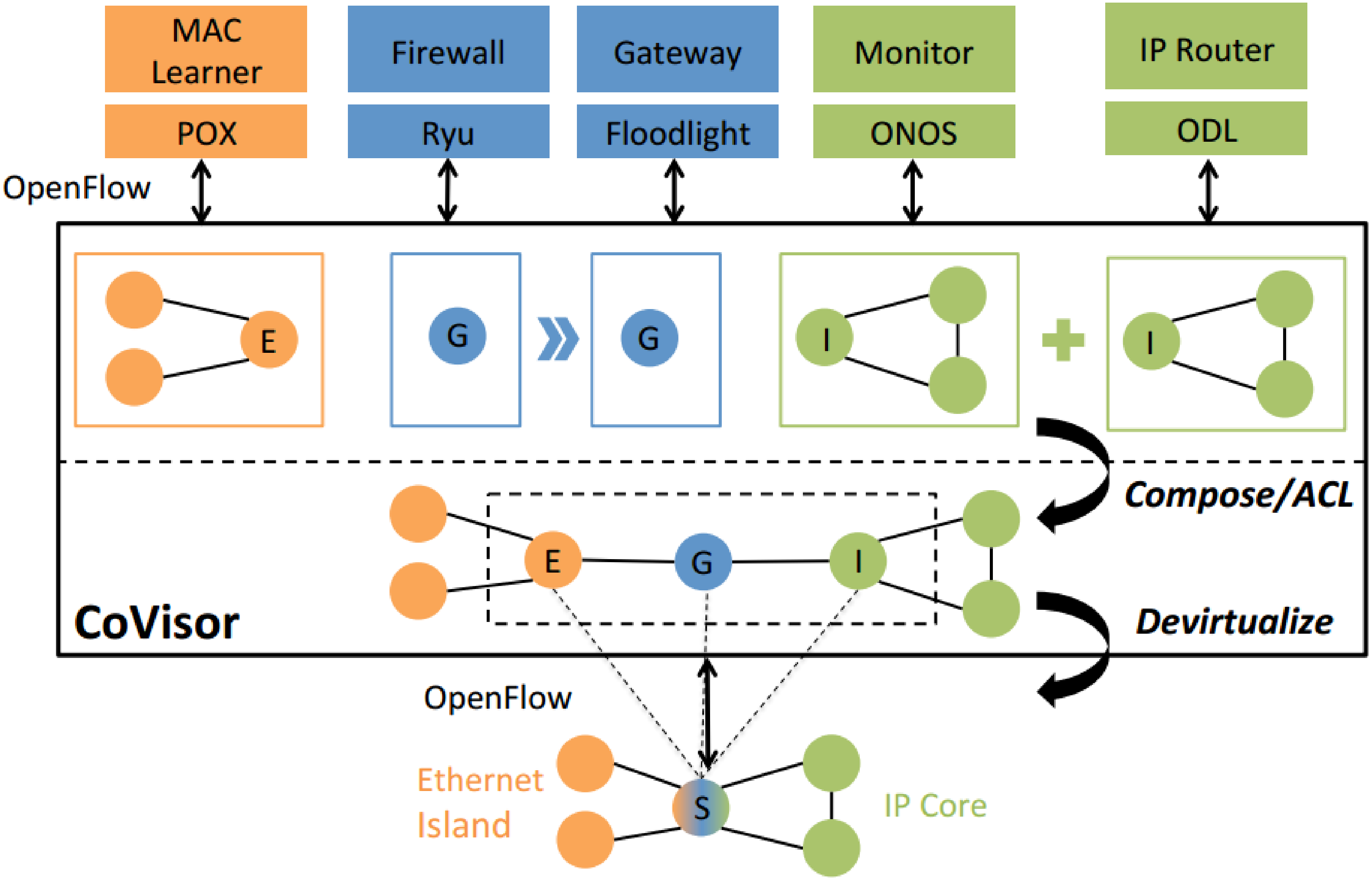
Default Router (Max priority = 8)

1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

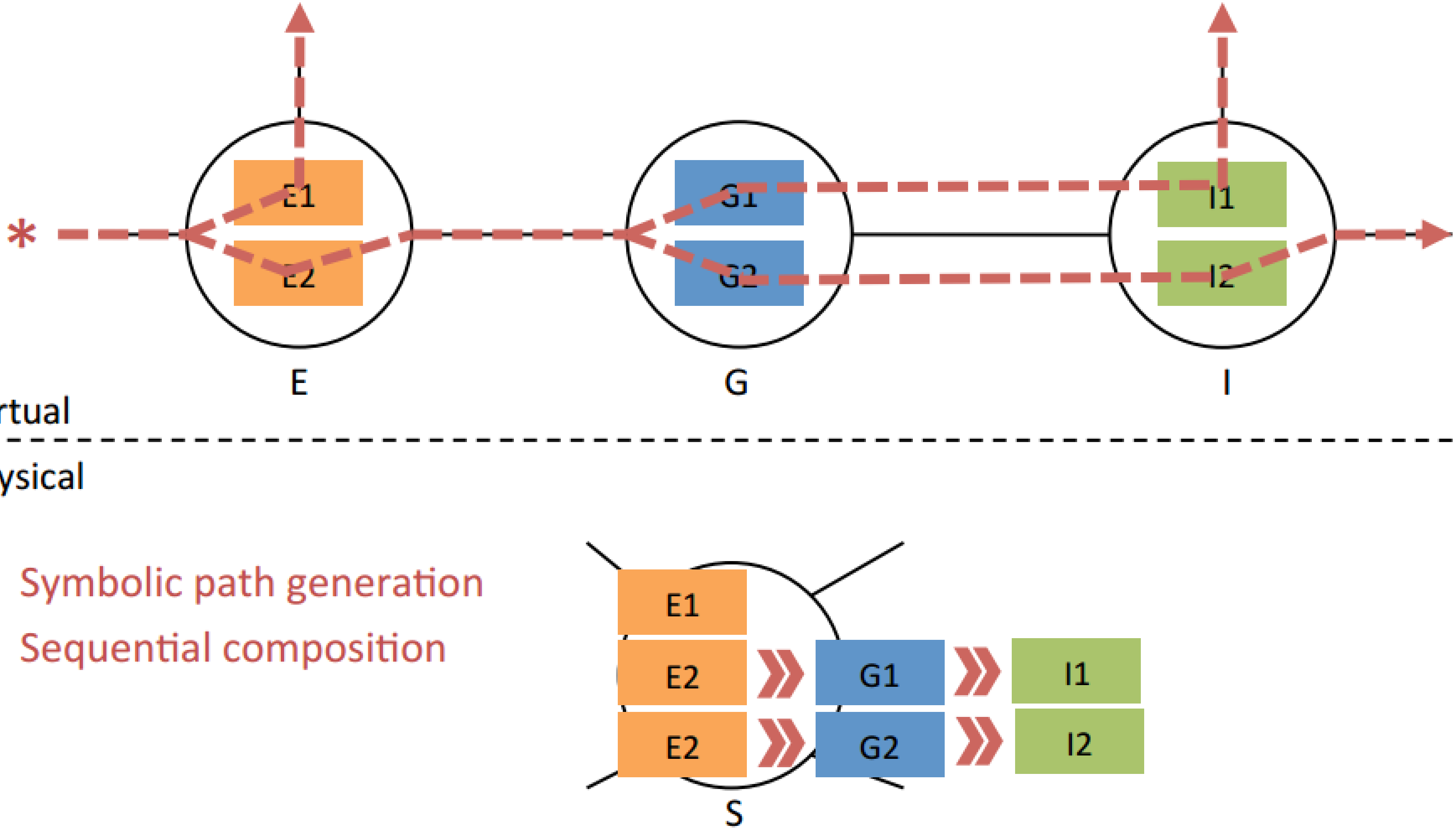
=

1 + 8 = 9. srcip=1.0.0.0, dstip=3.0.0.0 → fwd(3)
1. dstip=2.0.0.1 → fwd(1)
1. dstip=2.0.0.2 → fwd(2)
0. * → drop

CoVisor – Overview

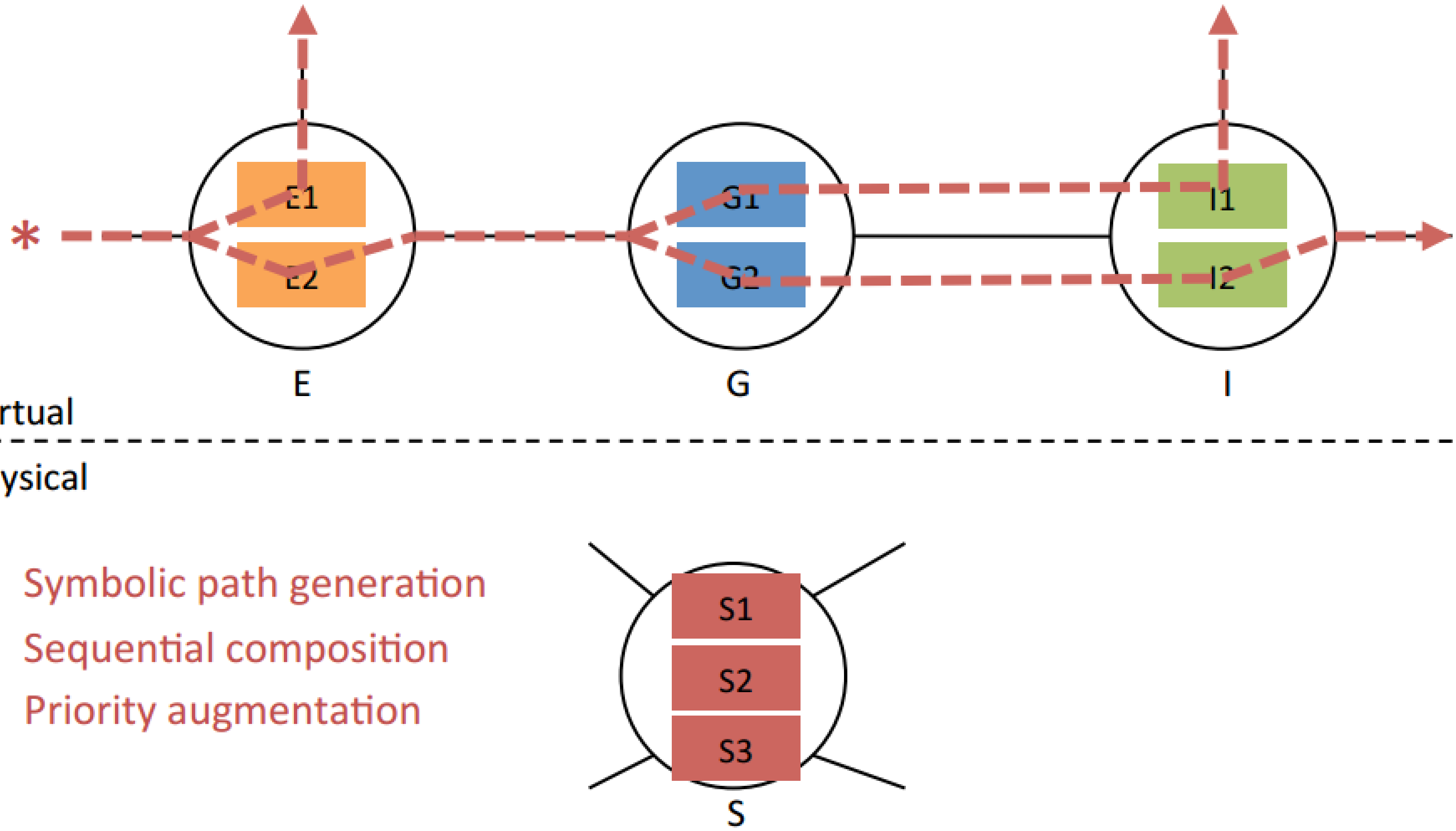


CoVisor - Devirtualization



- Symbolic path generation
- Sequential composition

CoVisor - Devirtualization



- Symbolic path generation
- Sequential composition
- Priority augmentation

Summary SDN

- SDN as a new way of networking that exploits existing concepts
  - Separation of planes, etc.

- OpenFlow as the de-facto standard protocol
- Controllers as operating systems

- Application: network virtualization
  - Slicing
  - Co-existence of different controllers
    - On disjoint traffic
    - On same traffic

Outlook SDN

- There is a lot more, just a small subset covered so far

- If you’re interested:
  - Block courses on Software-defined Networking (probably at the end of the upcoming winter semester, i.e., March 2016)
    - Introduction to SDN (1 week)
    - Advanced SDN (1 week)
  - Some things from this lecture will be familiar
  - Add-ons: practical work on SDNs, researching on SDNs