Advanced Computer Networks

Software Defined Networking

Lin Wang

Period 2, Fall 2021

VU VRIJE UNIVERSITEIT AMSTERDAM

Course outline

Warm-up

- Introduction (history, principles)
- Networking basics
- Networking data structures and algorithms
- Network transport

Data centers

- Data center networking
- Data center transport

Programmability

- **Software defined networking**
- Network automation
- Network function virtualization
- Programmable data plane

Application

- Network monitoring
- In-network computing
- Machine learning for networking

Marc Andreessen: co-author of Mosaic (the first widely used browser), co-founder of Netscape, co-founder of VC firm Andreessen Horowitz (a16z).

Learning objectives

Why software defined networking (SDN)? What is SDN?

How to use SDN to slice a network?

How to compose network control programs in SDN?

Why do we need SDN and what is it?

Internet has become a critical infrastructure, but…

NME

Xbox Live is down – service outage causes chaos for Xbox users

Around 7pm BST this evening, the Xbox Live service, now known as Xbox Network, began suffering widespread outages, seemingly affecting users...

13 hours ago

Messenger Newspapers

Sky Broadband down: Sky issue update on internet outage

Customers across South-East England and Wales have been left without internet as people report of Sky Broadband outages with internet an...

2 weeks ago

City AM

Solana back online after day long network outage

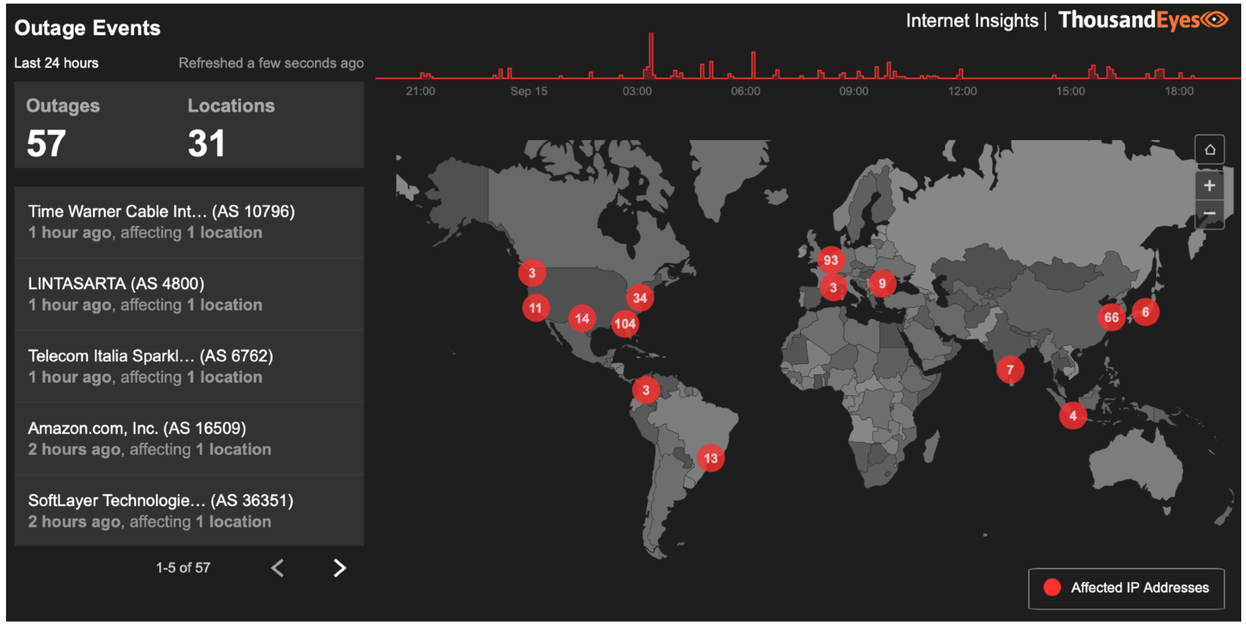
The Solana network is back online after experiencing a major outage for more than 17 hours, causing the price of its native token SOL to...

22 hours ago

S&P Global

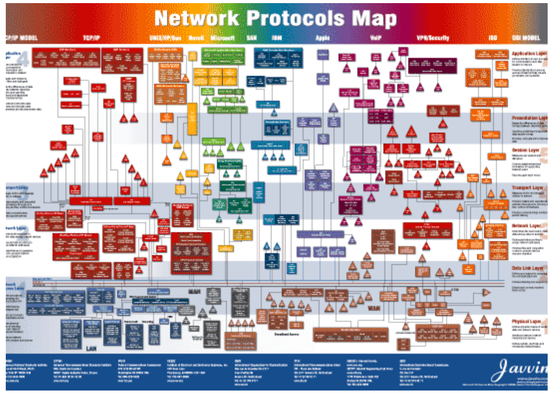
Global internet outages continue to increase in last week of ...

This weekly feature from S&P Global Market Intelligence, in collaboration with internet-service monitoring company ThousandEyes



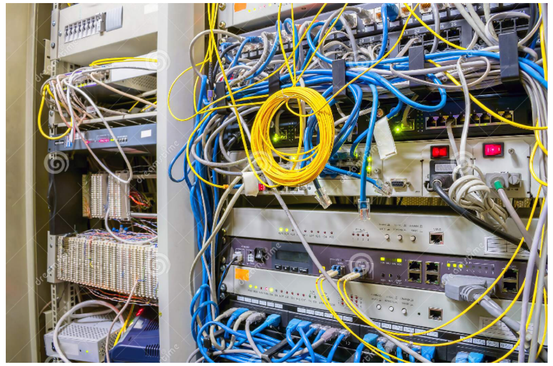
Surprisingly, most of these outages are due to **human errors** in network configuration!

We keep building a lot of complex artifacts…

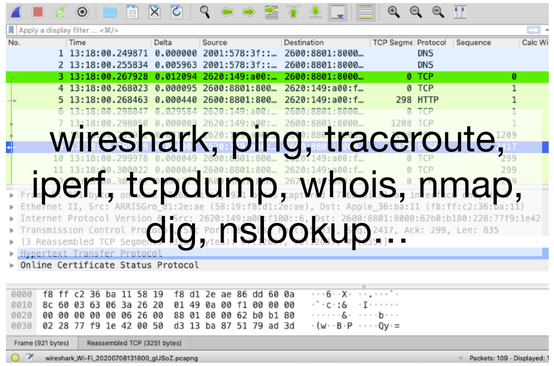


A plethora of network protocols

A stack of packet headers



A bunch of boxes and cables



A ton of network tools

Complexity in networking

RFC 2460 (1998)

We need **different functionalities**, also new ones

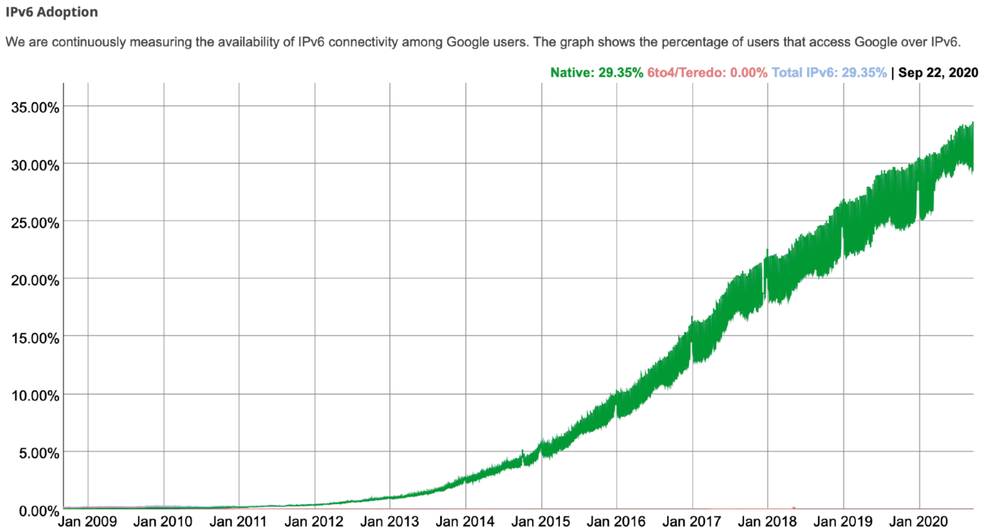
- Different physical layers and applications, traffic engineering, congestion control, security

Networks run in a **distributed**, **autonomous** way

- Scalability is important

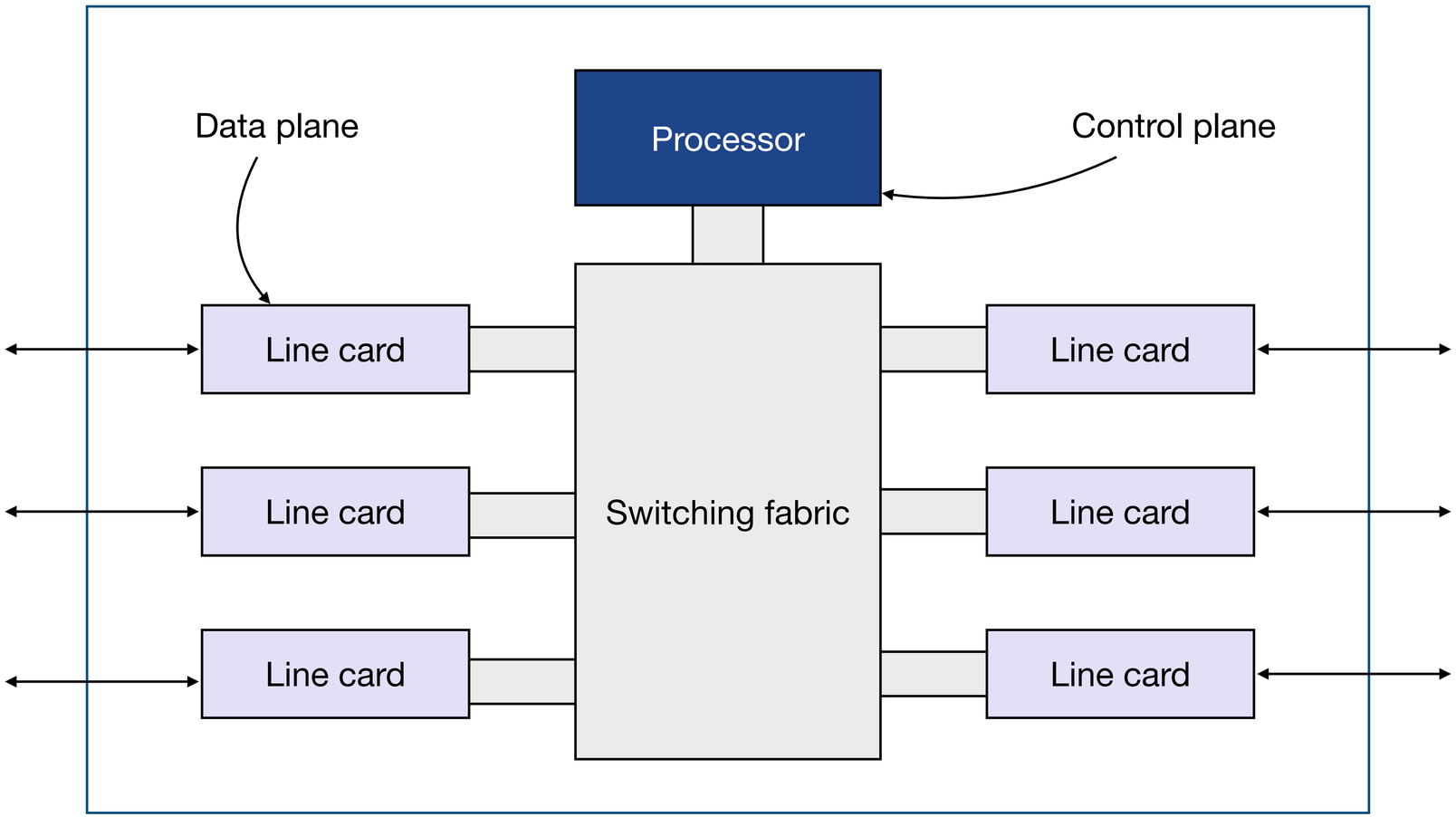
All these add to **complexity**, innovations are active in academia, but suffer from poor adoption of deployment

- Example: IPv6
- Deadlock between innovation and adoption



https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption

Network planes



Complexity in the control plane

Control plane needs to achieve goals such as connectivity, inter-domain policy, isolation, access control...

Currently, these goals are achieved by many mechanisms/protocols:

- Globally **distributed**: routing algorithms
- **Manual**/scripted configuration: Access Control Lists, VLANs
- **Centralized** computation: traffic engineering (indirect control)

Even worse, these mechanisms/protocols **interact with each other**

- Routing, addressing, access control, QoS

Network control plane is a complicated mess!

How have we managed to survive?

Network administrators miraculously master this complexity

- Understand all aspects of networks
- Must keep myriad details in mind

The ability to master complexity is both a blessing and a curse!

The **ability to master complexity** is valuable but not the same as **the ability to extract simplicity**

How to extract simplicity?

Example: programming

Machine languages: no abstractions

- Hard to deal with low-level details
- Mastering complexity is crucial

High-level languages: operating systems and other abstractions

- File systems, virtual memory, abstract data types...

Modern languages: even more abstractions

- Object oriented, garbage collection...

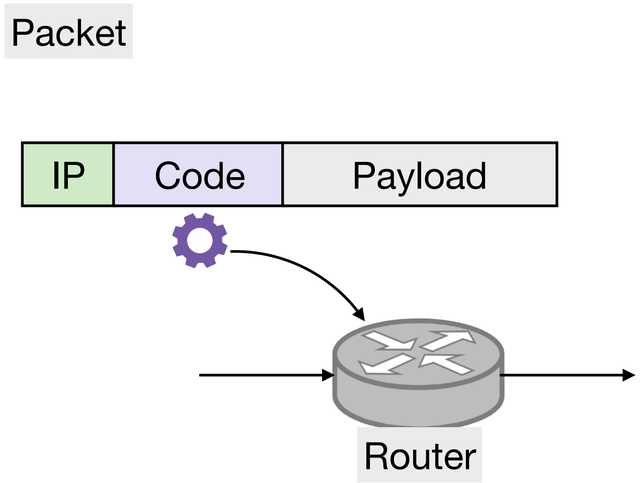
"Modularity based on abstractions is the way things get done!"

Barbara Liskov
(MIT, ACM Turing Award 2008, pioneer in programming languages, operating systems, distributed computing)

We need abstractions and ultimately, we should be able to **program the network** as we do for computers.

The evolution: active networking (1990s)

First attempt making networks **programmable**: demultiplexing packets to software programs



In-band approach: The packet encapsulates a small piece of code that can be executed on the router, based on which the router decides what to do with the packet

Out-band approach: User injects the code to be executed beforehand → the programmable network approach which received a lot of attention recently.

The evolution: control/data plane separation (2003-2007)

4D (ACM SIGCOMM CCR 2004)

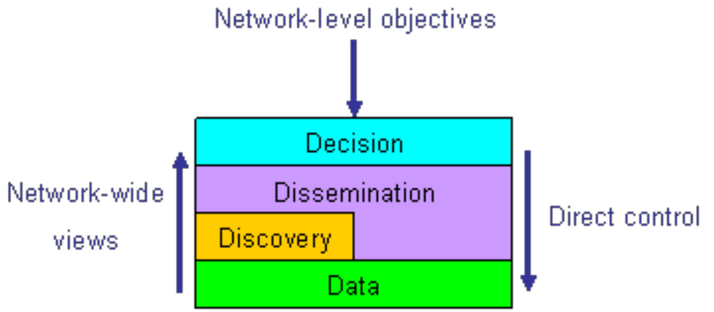
- Data, discovery, dissemination, decision
- Clean-slate: network-wide view, direct control, network-global objectives

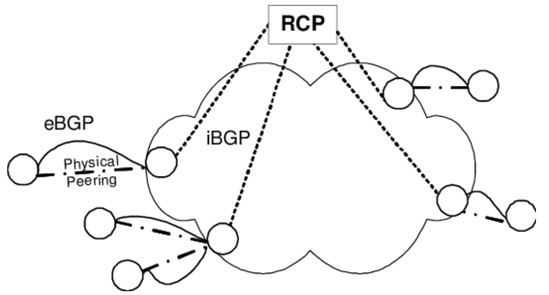
RCP (USENIX NSDI 2005)

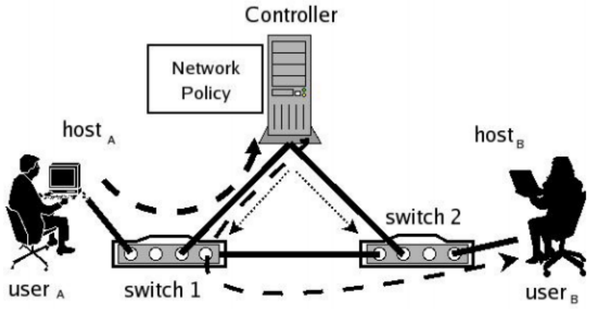
- Routing Control Platform for centralized intra-AS routing, replacing iBGP

Ethane (ACM SIGCOMM 2007)

- Flow-based switching with centralized control for enterprise
- Precursor of SDN



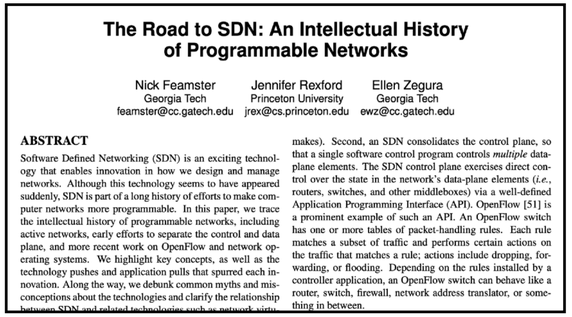




Software defined network

A network in which

- The **control** plane is physically **separate from** the **data** plane
- A **single** (logically centralized) **control** plane controls several forwarding devices



The Road to SDN: An Intellectual History of Programmable Networks

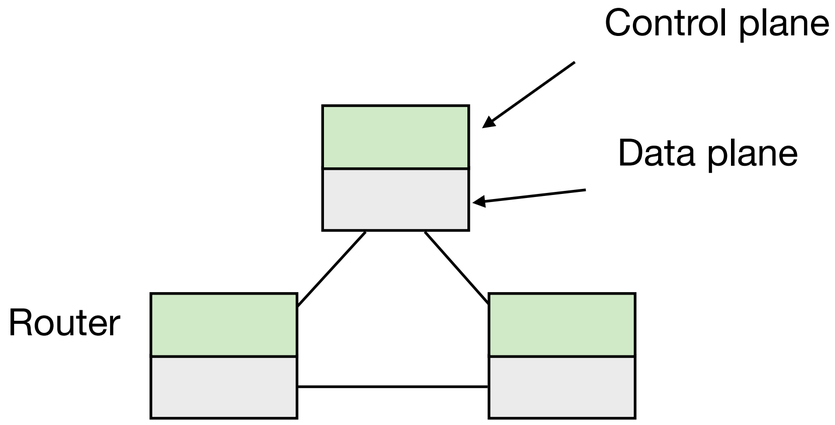
Nick Feamster, Georgia Tech, feamster@cc.gatech.edu
Jennifer Rexford, Princeton University, jrex@cs.princeton.edu
Ellen Zegura, Georgia Tech, ewz@cc.gatech.edu

ABSTRACT

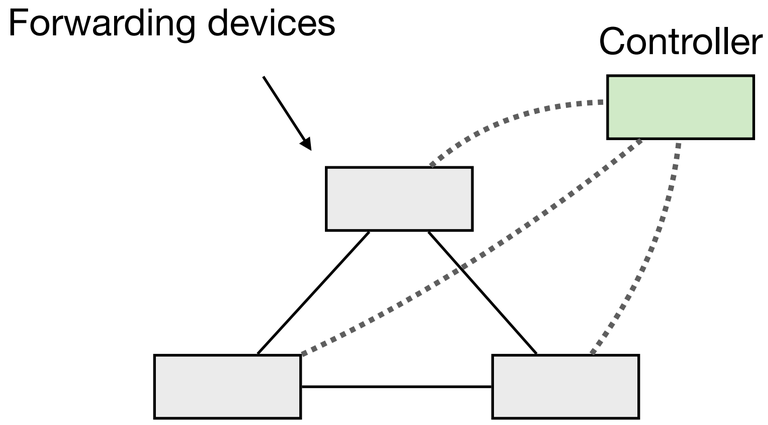
Software Defined Networking (SDN) is an exciting technology that enables innovation in how we design and manage networks. Although this technology seems to have appeared suddenly, SDN is part of a long history of efforts to make computer networks more programmable. In this paper, we trace the intellectual history of programmable networks, including active networks, early efforts to separate the control and data plane, and more recent work on OpenFlow and network operating systems. We highlight key concepts, as well as the technology pushes and application pulls that spurred each innovation. Along the way, we debunk common myths and misconceptions about the technologies and clarify the relationship

makes). Second, an SDN consolidates the control plane, so that a single software control program controls *multiple* data-plane elements. The SDN control plane exercises direct control over the state in the network's data-plane elements (*i.e.*, routers, switches, and other middleboxes) via a well-defined Application Programming Interface (API). OpenFlow [51] is a prominent example of such an API. An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of traffic and performs certain actions on the traffic that matches a rule; actions include dropping, forwarding, or flooding. Depending on the rules installed by a controller application, an OpenFlow switch can behave like a router, switch, firewall, network address translator, or something in between.

ACM SIGCOMM CCR 2014

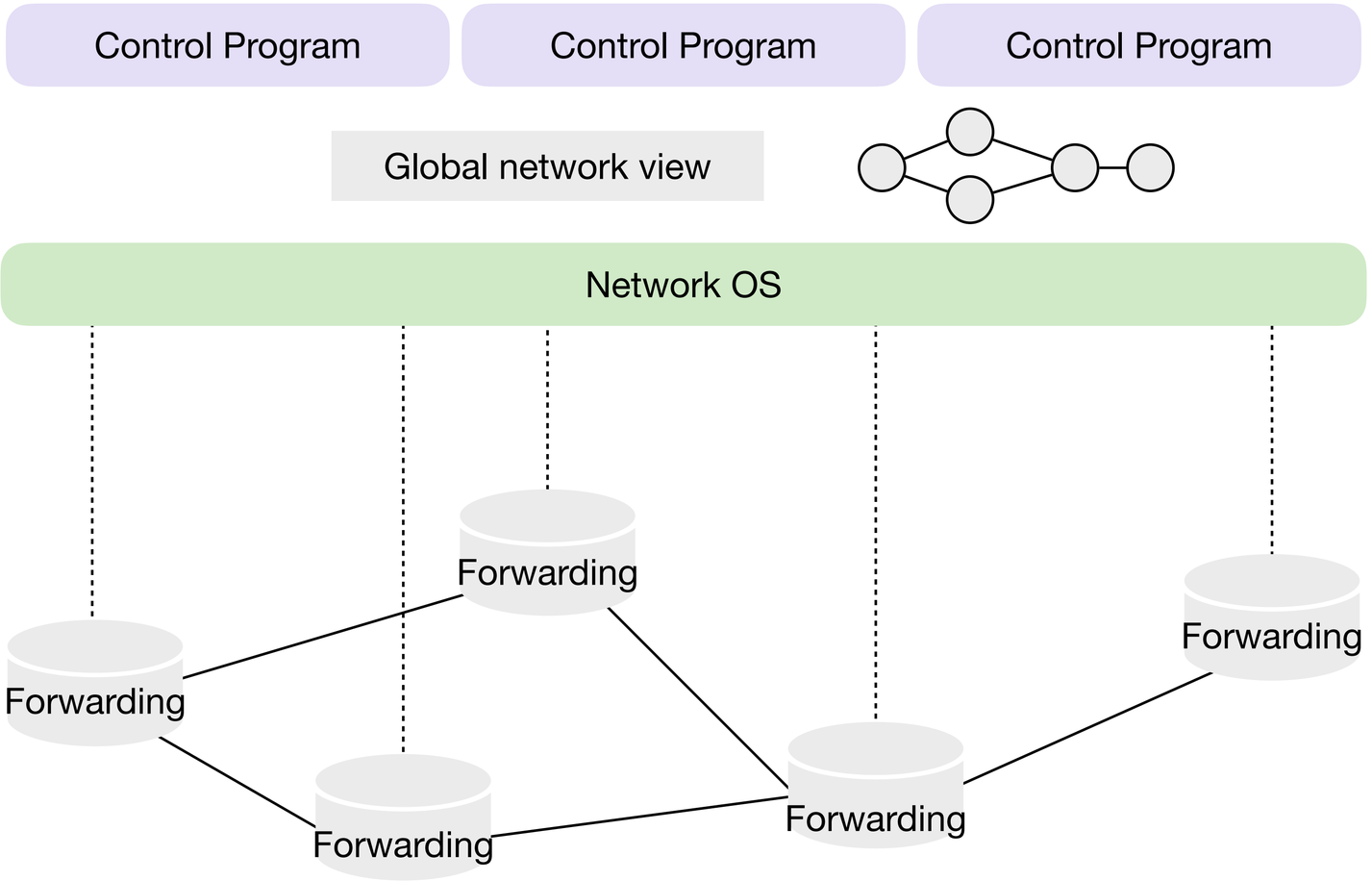


Traditional network

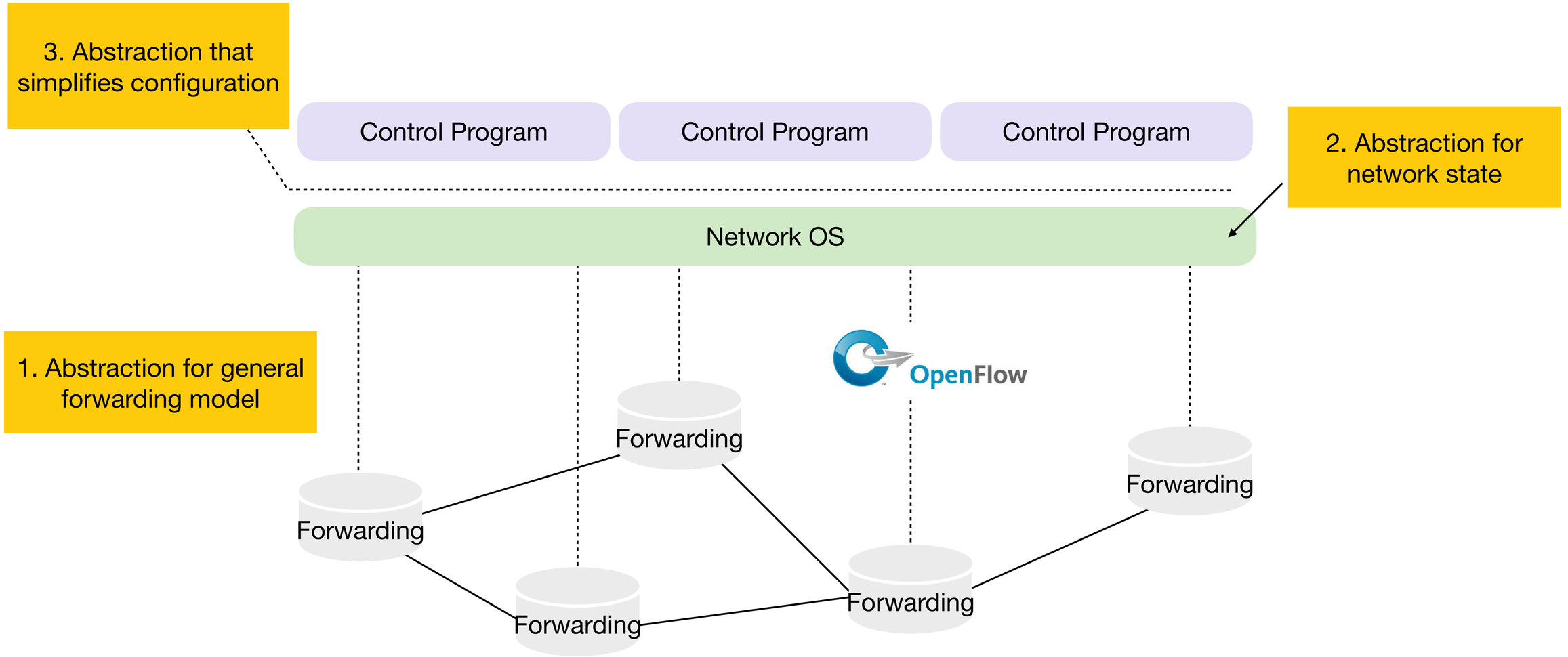


Software define network

SDN architecture overview



Abstractions in SDN



Abstraction #1: forwarding abstraction

Express intent independent of implementation

OpenFlow is the current proposal for forwarding

- Standardized interface to switch: non-proprietary COTS hardware and software
- Configuration in terms of flow entries: <header, action>
- No hardware modifications needed, simply a firmware update

Design details concern exact nature of **match+action**

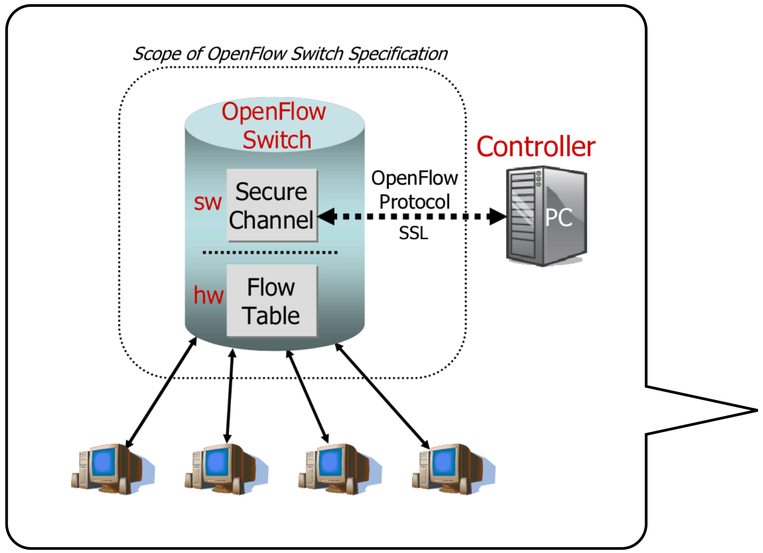
Benefits

- Much cheaper, no more $27K for a single switch
- No vendor lock-in

OpenFlow

Control Program | Control Program | Control Program

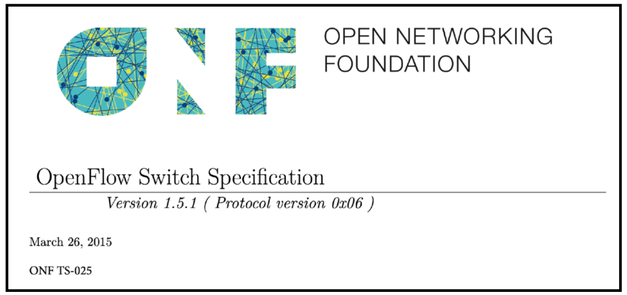
Network OS



OpenFlow switch

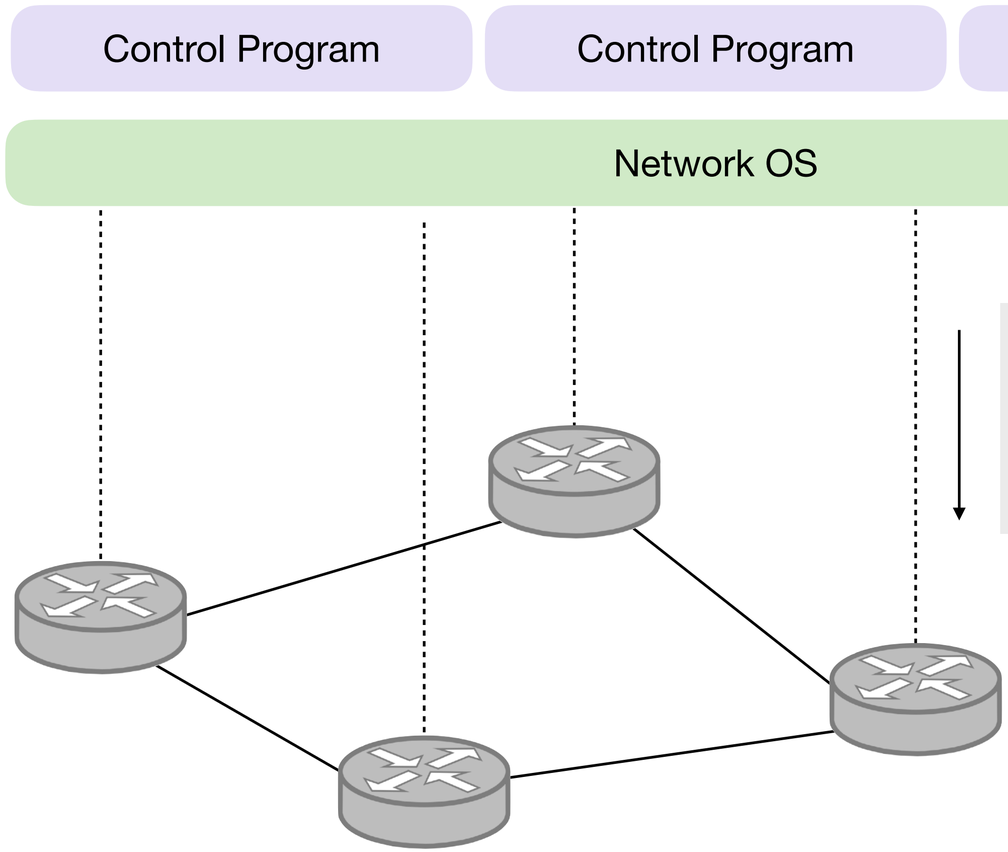
Flow tables: match+action

OpenFlow protocol



https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

OpenFlow example



OpenFlow switch

If header = "p", send to port 4
If header = "q", rewrite header to "r", add header "s", and send to port 5 and 6
If header = "?", send to me

match: "p", **action:** forward to 4
match: "q", **action:** rewrite..., forward to 5&6
match: "?", **action:** forward to Network OS

Flow table

Flow table(s) on OpenFlow switches

Flow 1	Rule (exact & wildcard)	Action	Statistics	Priority
Flow 2	Rule (exact & wildcard)	Action	Statistics	Priority
Flow 3	Rule (exact & wildcard)	Action	Statistics	Priority
		.....		
Flow *N*	Rule (exact & wildcard)	Action	Statistics	Priority

Exploit the forwarding tables that are already in routers, switches, and chipsets

Match+action

Match arbitrary fields in headers

- Match on any header, or new header
- Allows any flow granularity

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Action

- Forward to port(s), drop, send to the controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate
- Do not support payload-related network functions like deep packet inspection

Match: 1000X01XX0101001X

Abstraction #2: network state abstraction

Global network view

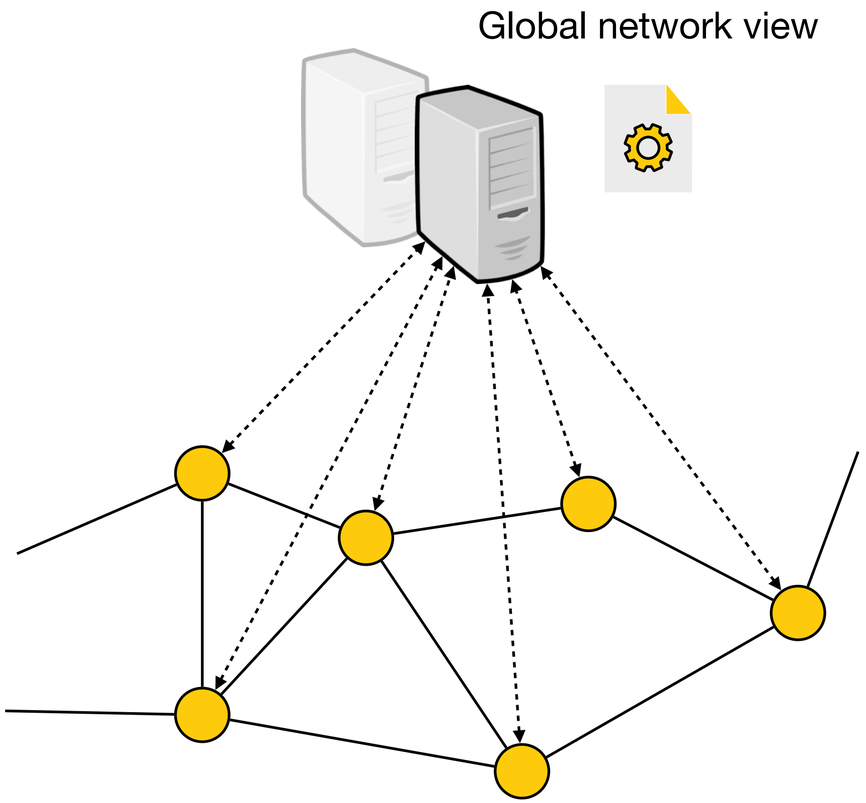
- Annotated network graph provided through an API
- Control program: Configuration = Function(View)

Implementation: "Network Operating Systems"

- Runs on servers in network (as "controllers")
- Replicated for reliability

Information flows both ways

- Information from routers/switches to form view
- Configurations to routers/switches to control forwarding



Abstraction #3: specification abstraction

Control mechanism expresses **desired behavior**

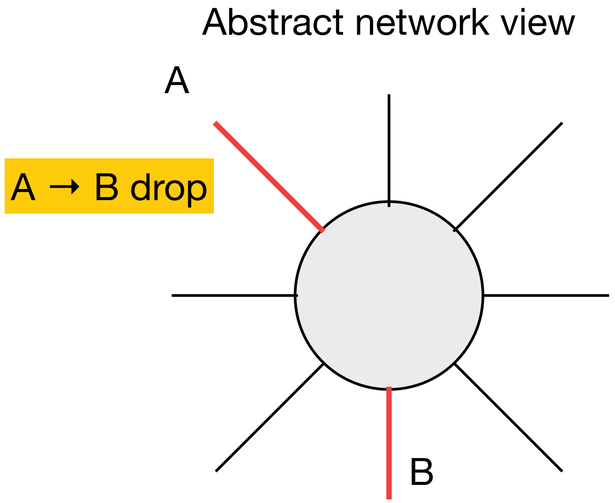
- Whether it be isolation, access control, or QoS

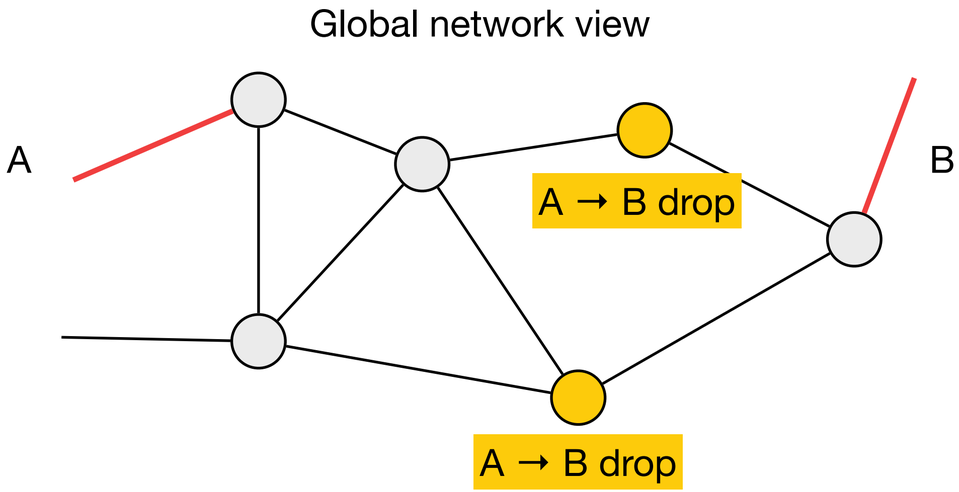
It should not be responsible for implementing that behavior on physical network infrastructure

- Requires configuring the forwarding tables in each switch

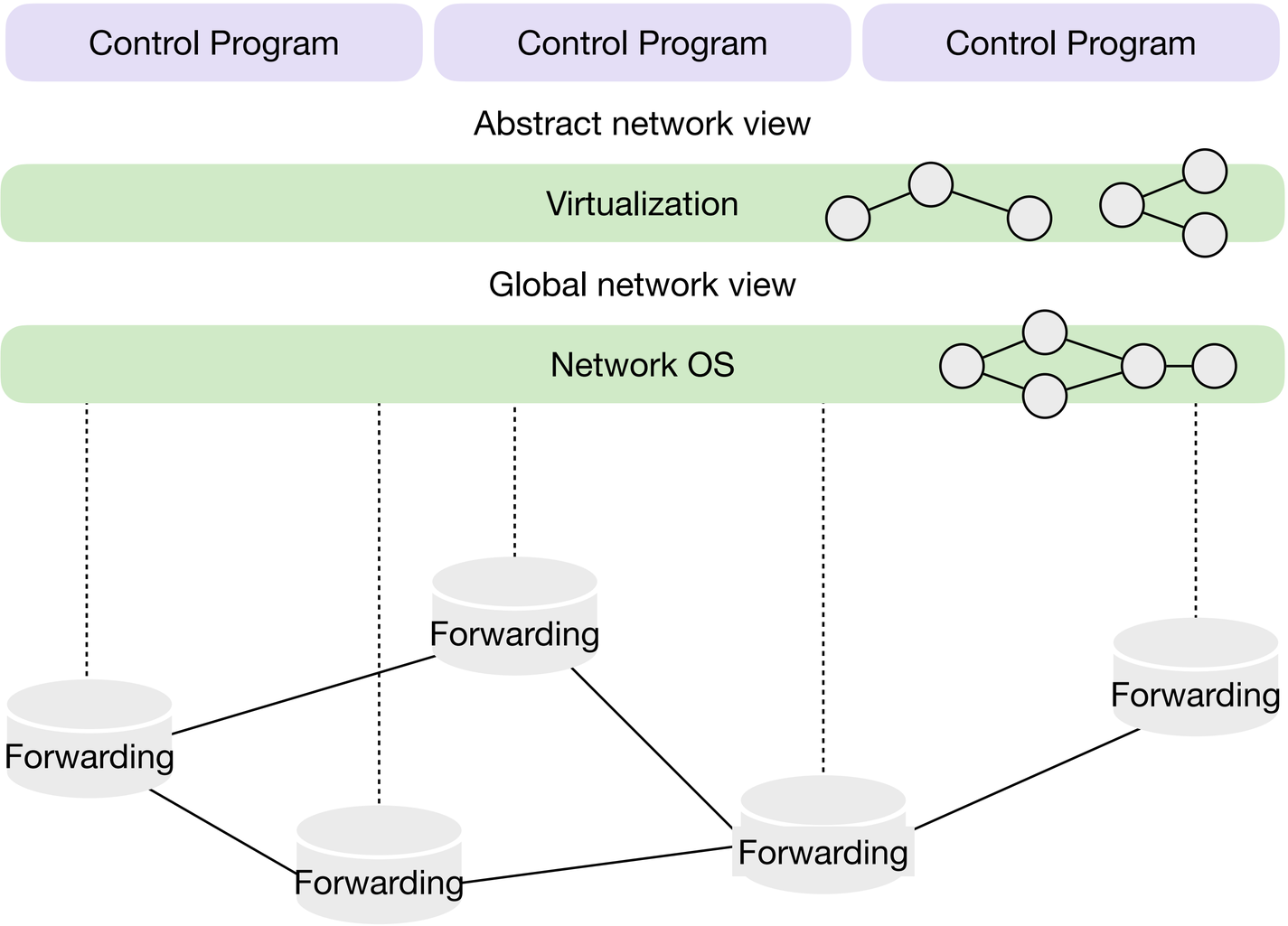
Proposed abstraction: **abstract view of the network**

- Abstract view models only enough detail to specify goals
- Will depend on task semantics





SDN control plane layers

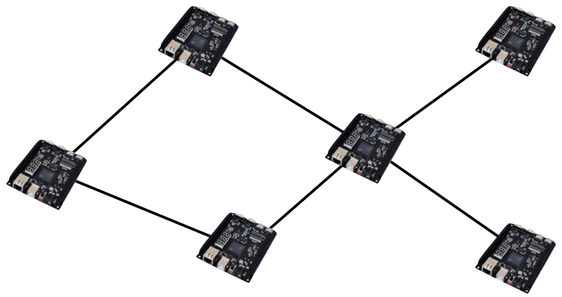


How to use SDN to slice a network?

Network testing

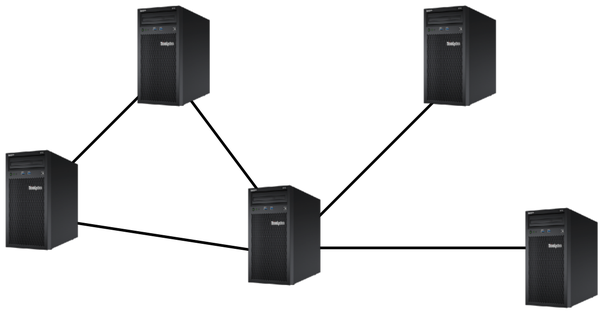
Imagine you come up with a novel network service, e.g., a new routing protocol, network load-balancer, how would you convince people that this is useful?

Hardware testbed



Expensive! Small-scale (fanout is small due to limited port number on NetFPGA)!

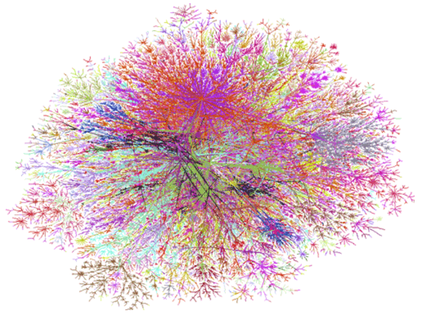
Software testbed



Large-scale (VINI/PlanetLab, Emulab)

Performance is slow (CPU-based), no realistic topology, hard to maintain!

Wild test on the Internet



Convincing network operators to try something new is very difficult! (Outages are the worst)

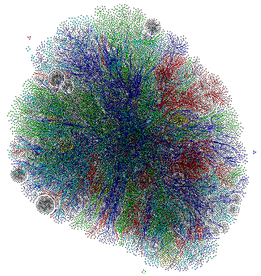
Network testing problems

Realistically evaluating new network services is **hard**

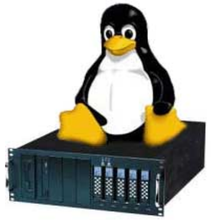
- Services that require changes to switches and routers
- For example: routing protocols, traffic monitoring services, IP mobility

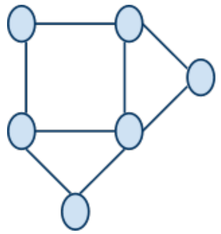
Results

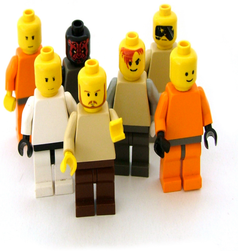
- Many good ideas do not get deployed
- Many deployed services still have bugs



Real networks







Test environments

Solution: network slicing

Divide the production network into logical **slices**

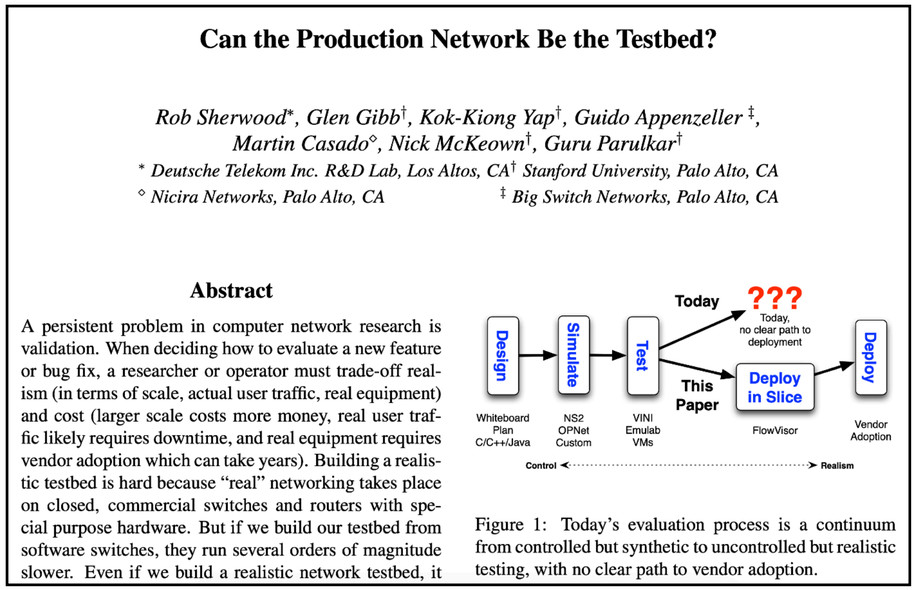
- Each slice/service controls its own packet forwarding
- Users pick which slice controls their traffic: opt-in
- Existing production services run in their own slice: spanning tree, OSPF/BGP

Enforce **strong isolation** between slices

- Actions in one slice do not affect others

Allow the (logical) **testbed** to mirror the **production** network

- Real hardware, performance, topologies, scale, users



Can the Production Network Be the Testbed?

Rob Sherwood, Glen Gibb†, Kok-Kiong Yap†, Guido Appenzeller ‡,*
Martin Casado◇, Nick McKeown†, Guru Parulkar†
** Deutsche Telekom Inc. R&D Lab, Los Altos, CA† Stanford University, Palo Alto, CA*
◇ Nicira Networks, Palo Alto, CA ‡ Big Switch Networks, Palo Alto, CA

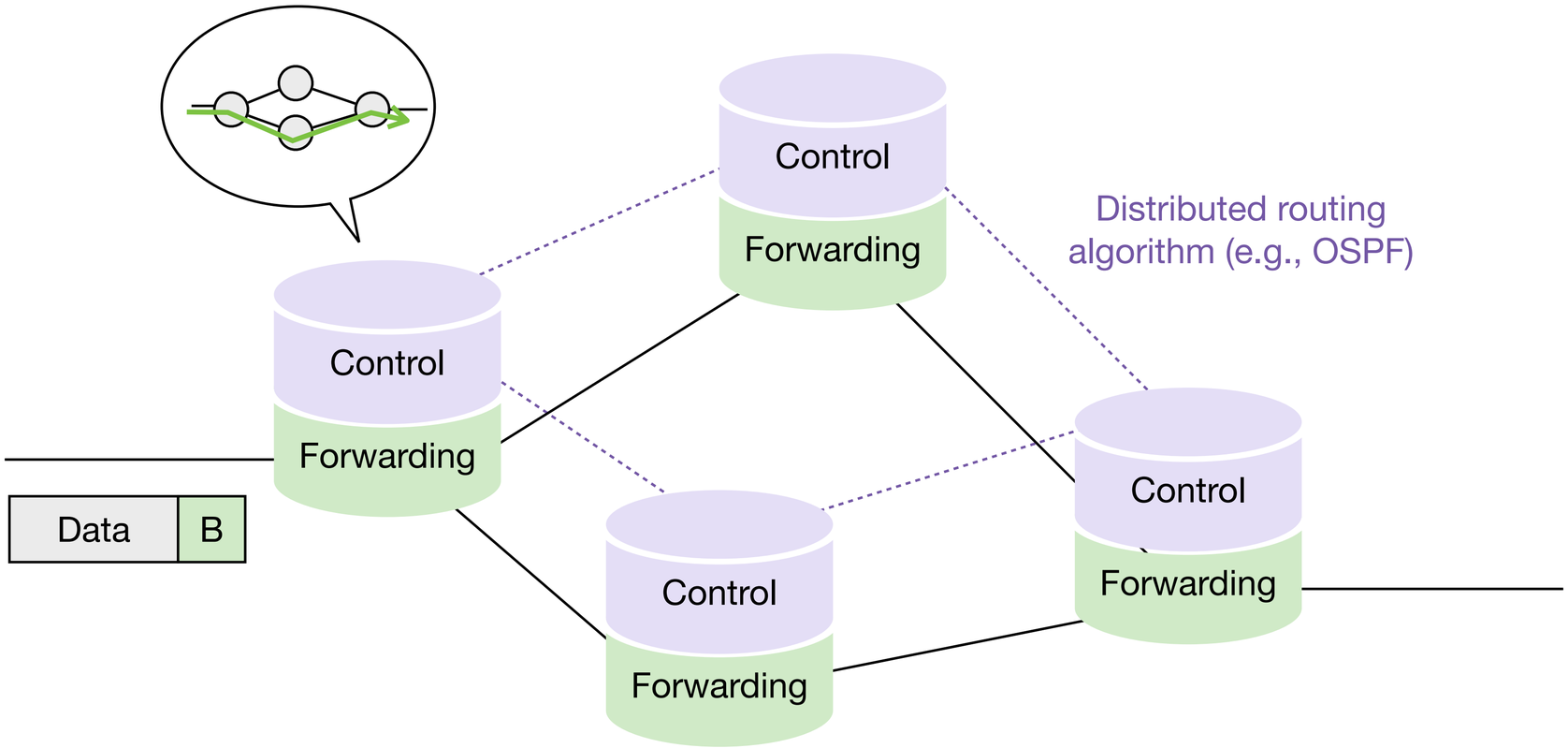
Abstract

A persistent problem in computer network research is validation. When deciding how to evaluate a new feature or bug fix, a researcher or operator must trade-off realism (in terms of scale, actual user traffic, real equipment) and cost (larger scale costs more money, real user traffic likely requires downtime, and real equipment requires vendor adoption which can take years). Building a realistic testbed is hard because "real" networking takes place on closed, commercial switches and routers with special purpose hardware. But if we build our testbed from software switches, they run several orders of magnitude slower. Even if we build a realistic network testbed, it

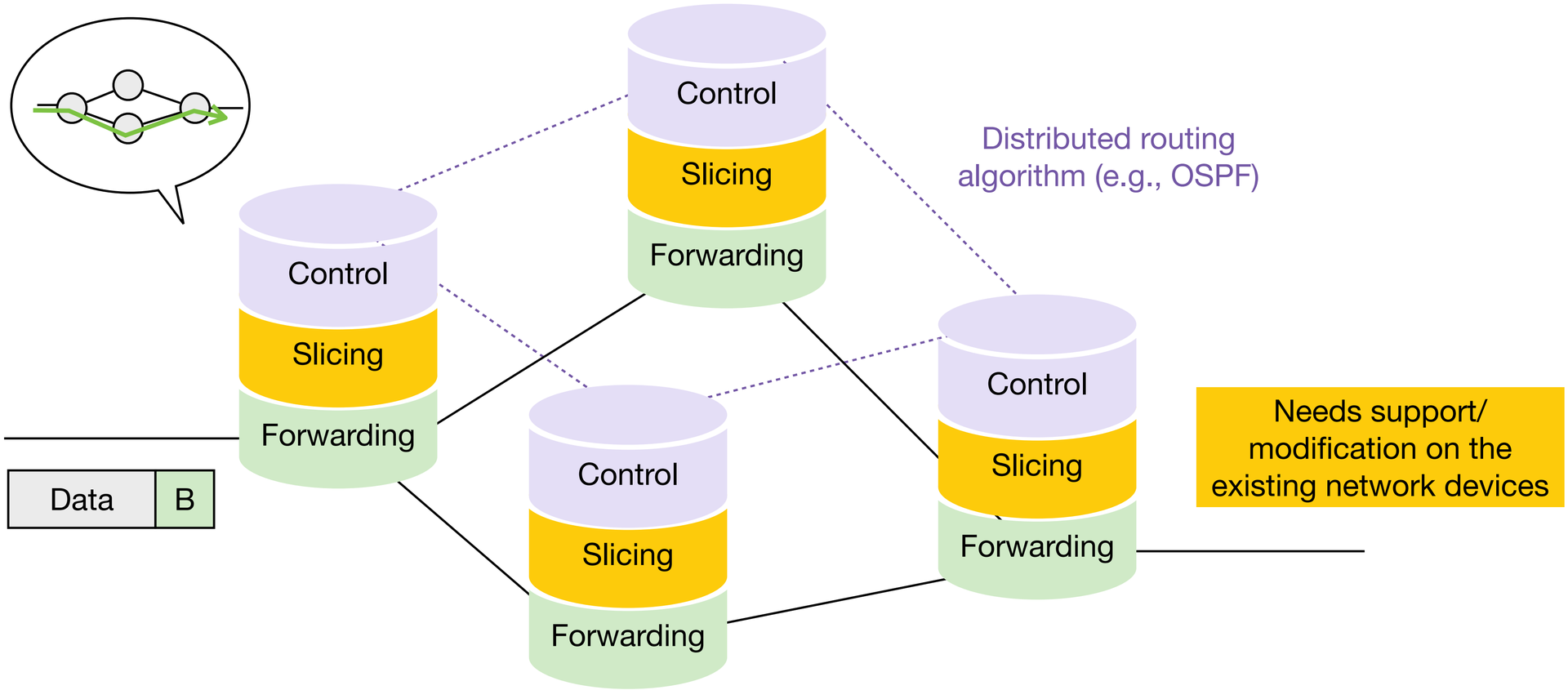
Figure 1: Today's evaluation process is a continuum from controlled but synthetic to uncontrolled but realistic testing, with no clear path to vendor adoption.

USENIX OSDI 2010

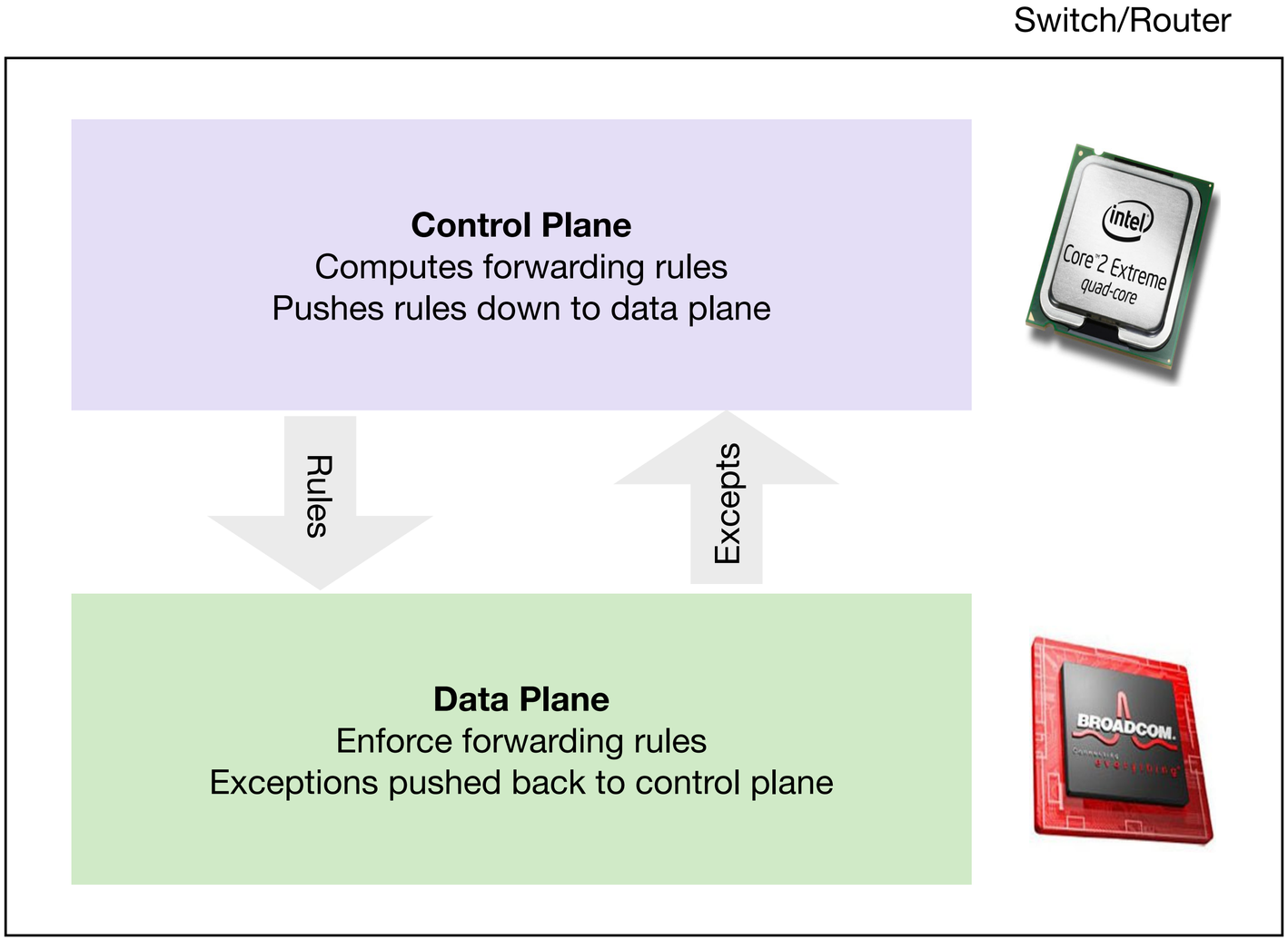
Traditional network



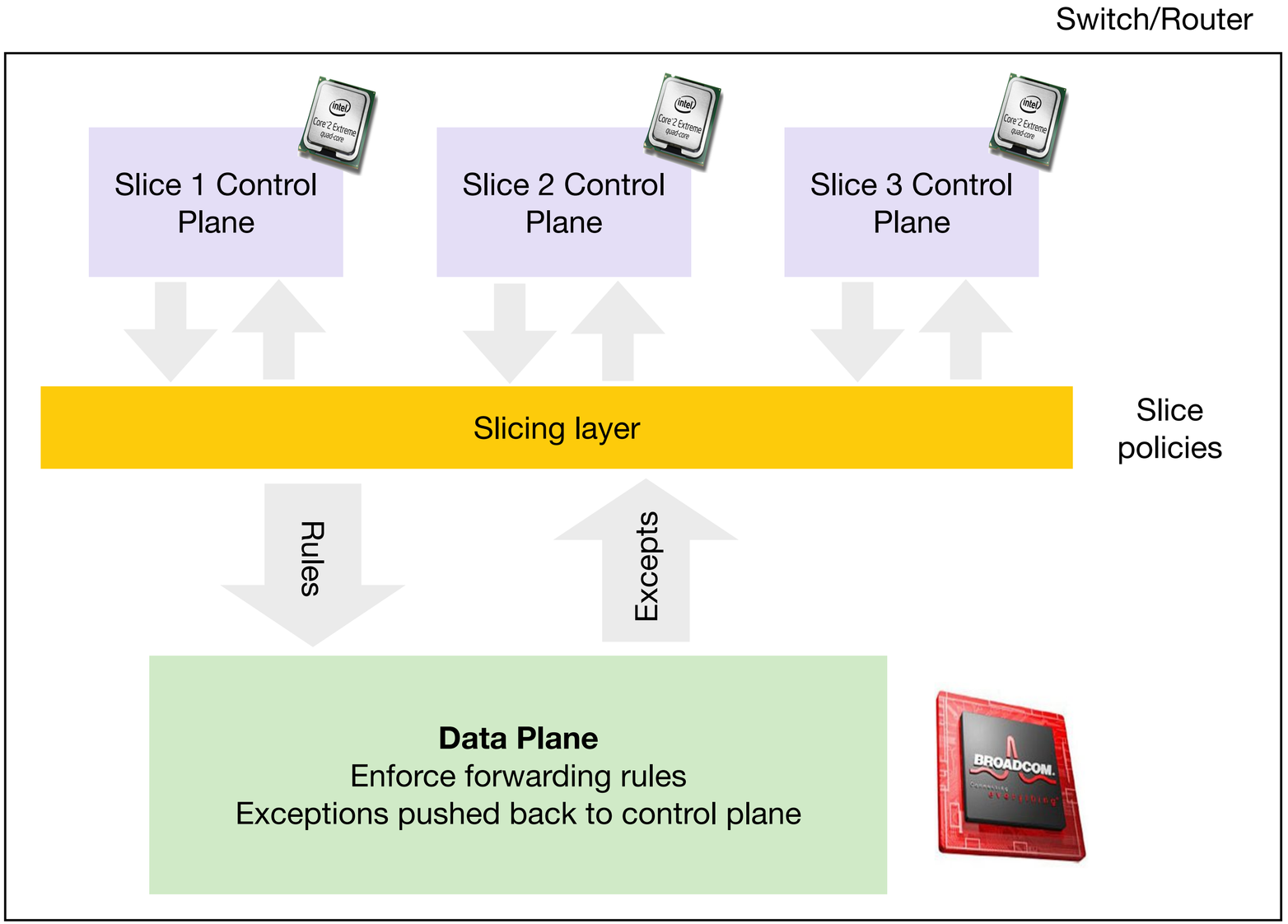
Slicing traditional network



Current network devices



Slicing layer



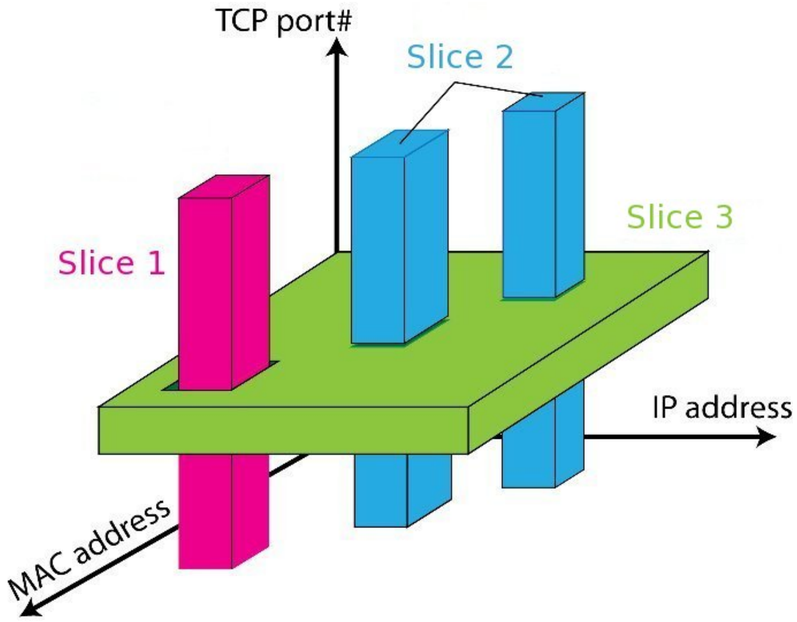
Slicing policies

The slicing policy specifies the **resource limit** for each slice:

- Link bandwidth
- Maximum number of forwarding rules (on switches)
- Topology
- Fraction of switch/router CPU

FlowSpace: which packet does the slice control?

- Maps packets to slices according to their "classes" defined by the packet header fields



Real user traffic: opt-in

Allow users to opt-in to services in real time

- Users can delegate control of individual flows to slices
- Add new FlowSpace to each slice's policy

Examples

- "Slice 1 will handle my HTTP traffic"
- "Slice 2 will handle my VoIP traffic"
- "Slice 3 will handle everything else"

Creates incentives for building high-quality services!

Source: gacovinolack.com

Slice definition

Bob's experimental slice: all HTTP traffic to/from users who opted in

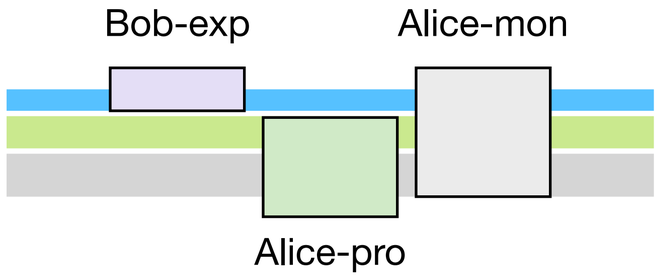
- `Allow: tcp_port=80 and ip=user_ip`

Alice's production slice: complementary to Bob's slice

- `Deny: tcp_port=80 and ip=user_ip`
- `Allow: all`

Alice's monitoring slice: all traffic in all slices

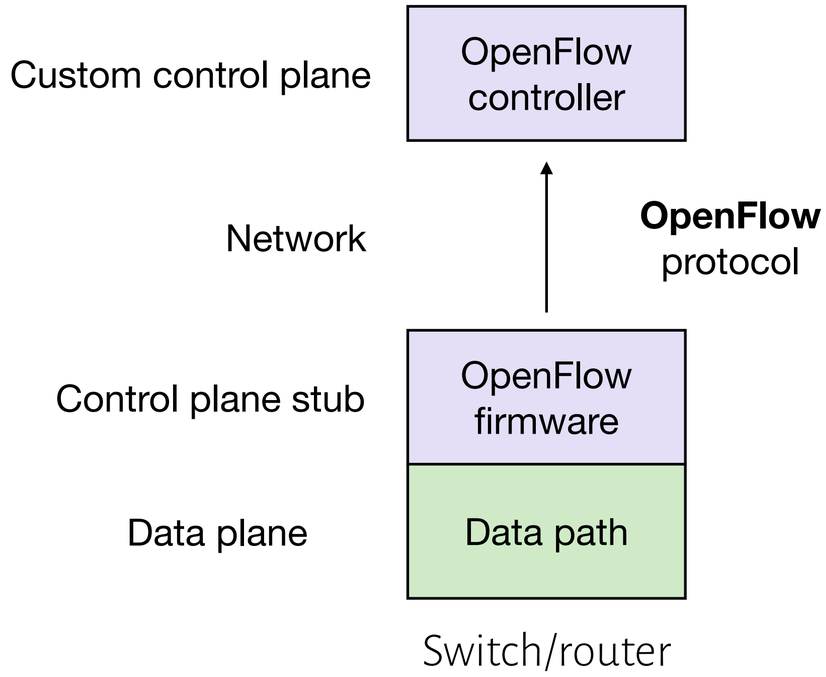
- `Read-only: all`



Slicing with OpenFlow

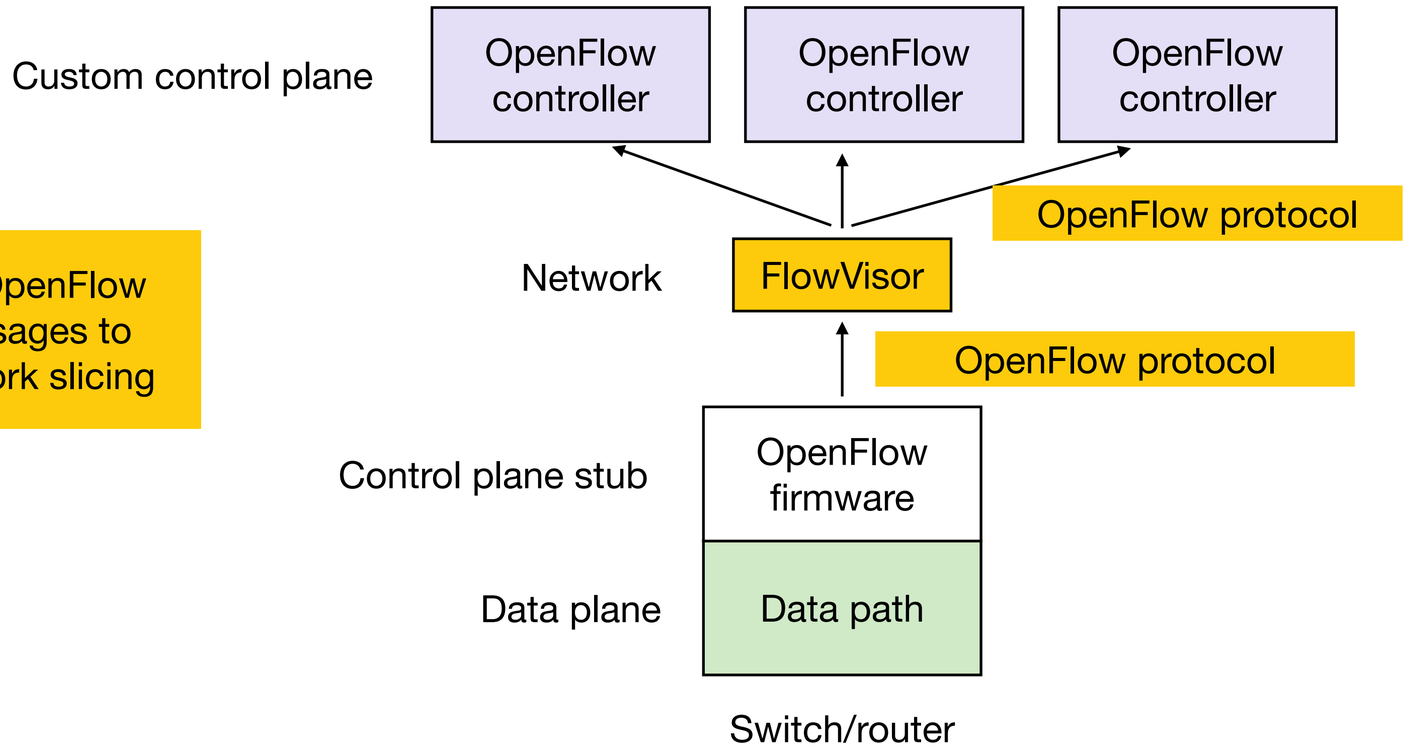
Recall OpenFlow:

- API for controlling packet forwarding
- Abstraction of control/data plane protocols
- Works on commodity hardware (via firmware upgrade)



How should we slice an OpenFlow-based software defined network?

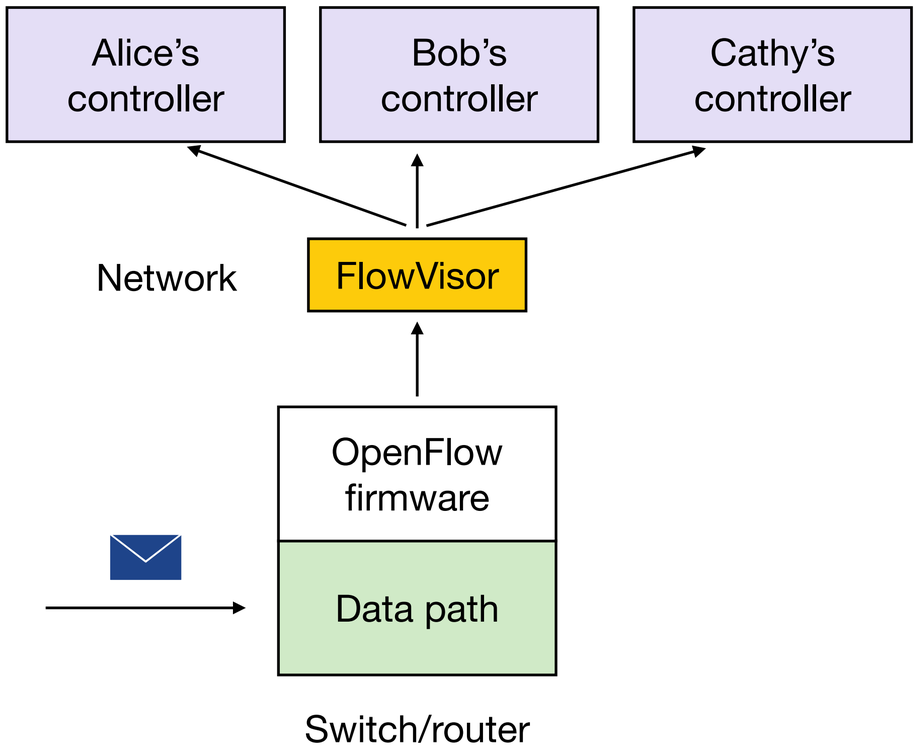
FlowVisor



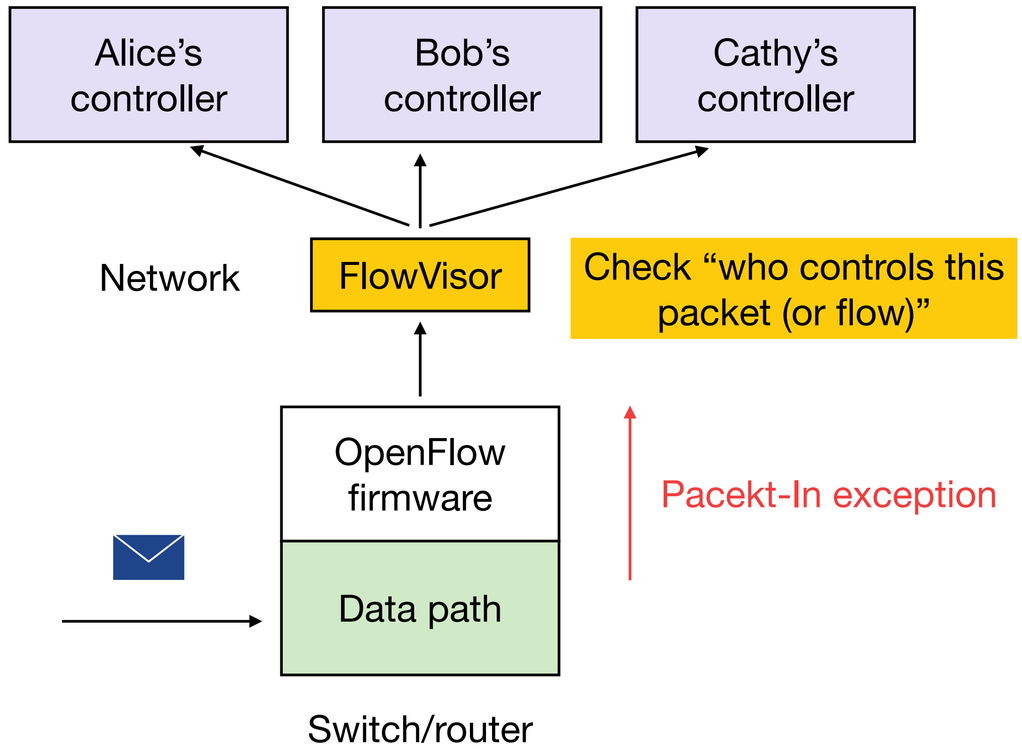
Interposing OpenFlow control messages to enforce network slicing

https://github.com/opennetworkinglab/flowvisor

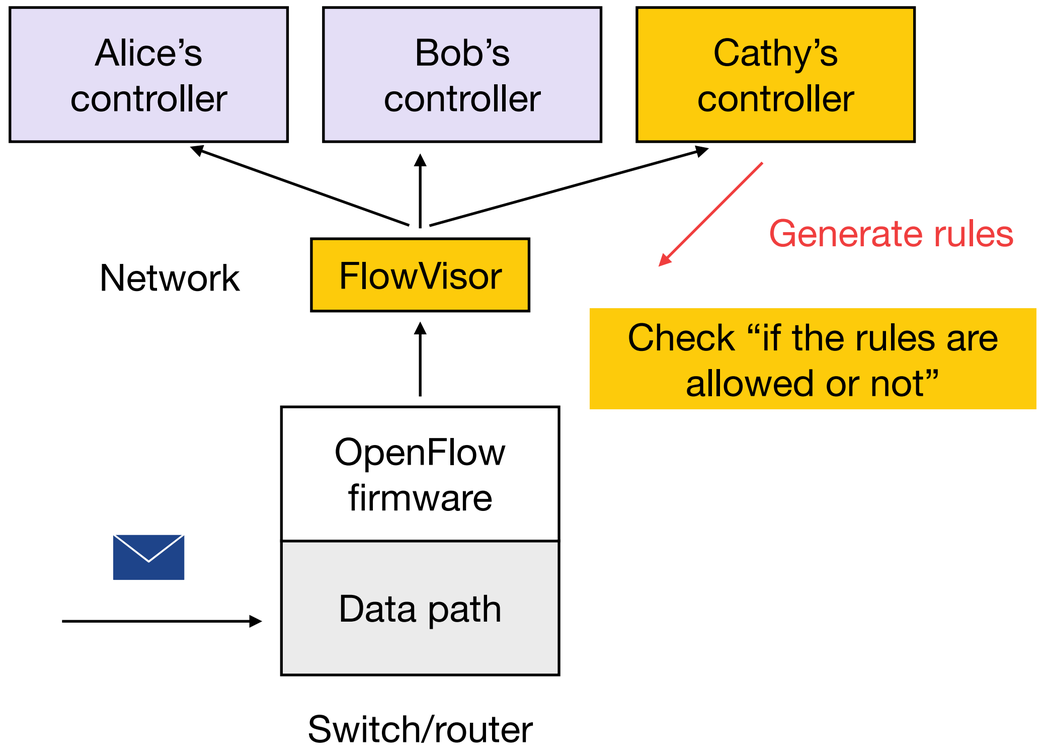
FlowVisor packet handling



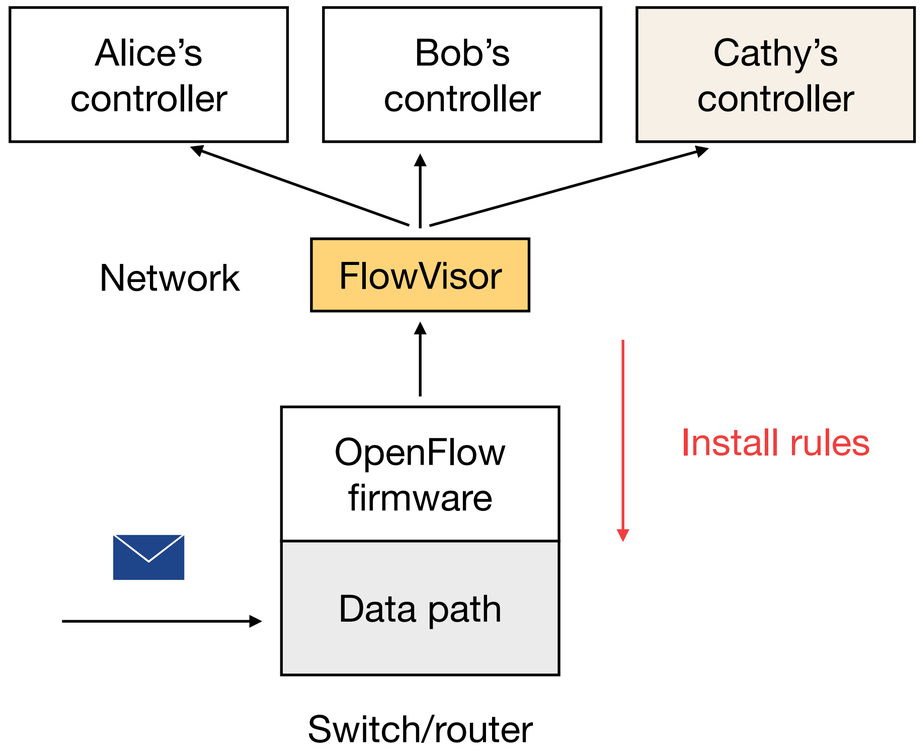
FlowVisor packet handling



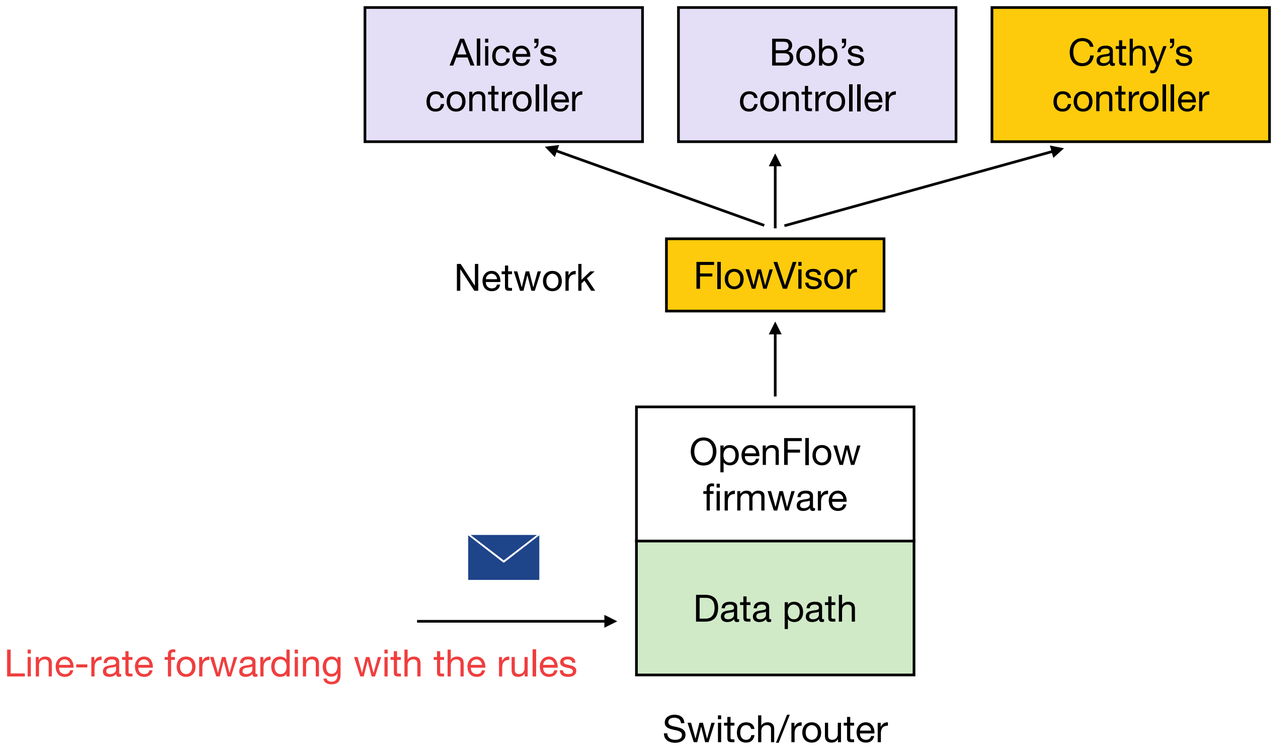
FlowVisor packet handling



FlowVisor packet handling

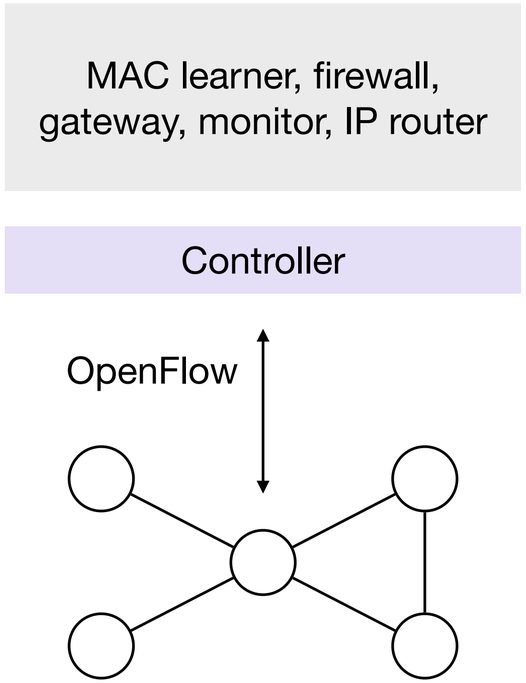


FlowVisor packet handling



How to compose network control programs in SDN?

Multiple management tasks in SDN



Option 1: Maintain a monolithic application

→ hard to debug and maintain

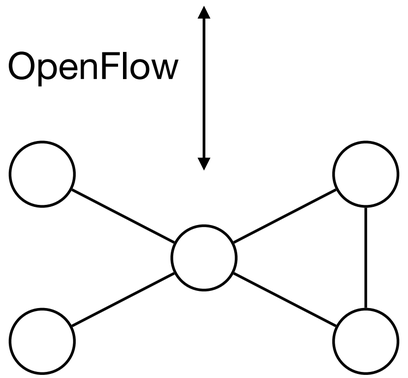
Option 2: Use composition operators (e.g., Frenetic controller) to combine multiple applications

→ Require to use the Frenetic language and runtime system

SDN reality

“Best of breed” control applications are developed by different parties,
using different languages, running on different controllers

MAC learner	Firewall	Gateway	Monitor	IP router
POX	Ryu	Floodlight	ONOS	ODL



How to mix-and-match
different controllers?

CoVisor: a compositional hypervisor for SDN

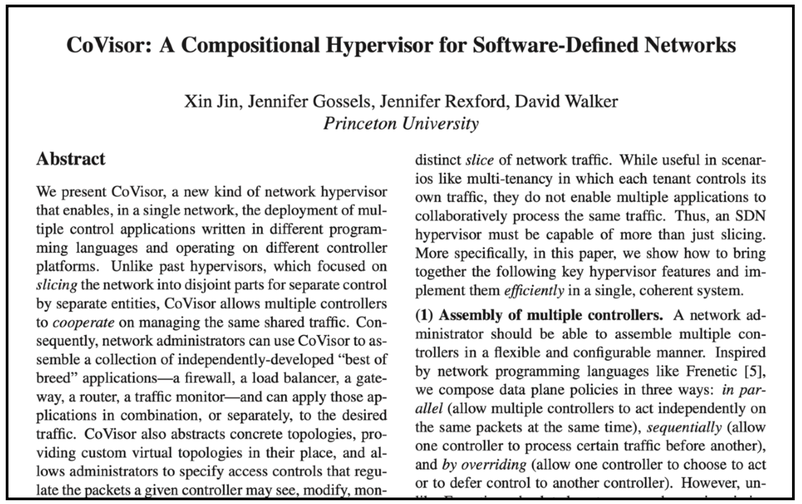
Provide clean interface to compose multiple controllers on the same network

Composition of multiple controllers

- Use composition operators to compose multiple controllers

Constraints on individual controllers

- Visibility: virtual topology to each controller
- Capability: fine-grained access control to each controller



CoVisor: A Compositional Hypervisor for Software-Defined Networks

Xin Jin, Jennifer Gossels, Jennifer Rexford, David Walker
Princeton University

Abstract

We present CoVisor, a new kind of network hypervisor that enables, in a single network, the deployment of multiple control applications written in different programming languages and operating on different controller platforms. Unlike past hypervisors, which focused on *slicing* the network into disjoint parts for separate control by separate entities, CoVisor allows multiple controllers to *cooperate* on managing the same shared traffic. Consequently, network administrators can use CoVisor to assemble a collection of independently-developed "best of breed" applications—a firewall, a load balancer, a gateway, a router, a traffic monitor—and can apply those applications in combination, or separately, to the desired traffic. CoVisor also abstracts concrete topologies, providing custom virtual topologies in their place, and allows administrators to specify access controls that regulate the packets a given controller may see, modify, mon-

distinct *slice* of network traffic. While useful in scenarios like multi-tenancy in which each tenant controls its own traffic, they do not enable multiple applications to collaboratively process the same traffic. Thus, an SDN hypervisor must be capable of more than just slicing. More specifically, in this paper, we show how to bring together the following key hypervisor features and implement them *efficiently* in a single, coherent system.

(1) Assembly of multiple controllers. A network administrator should be able to assemble multiple controllers in a flexible and configurable manner. Inspired by network programming languages like Frenetic [5], we compose data plane policies in three ways: *in parallel* (allow multiple controllers to act independently on the same packets at the same time), *sequentially* (allow one controller to process certain traffic before another), and *by overriding* (allow one controller to choose to act or to defer control to another controller). However, un-

USENIX NSDI 2014

Composition of multiple controllers

Parallel operator (+): two controllers process packets in parallel

Sequential operator (>>): two controllers process packets one after another

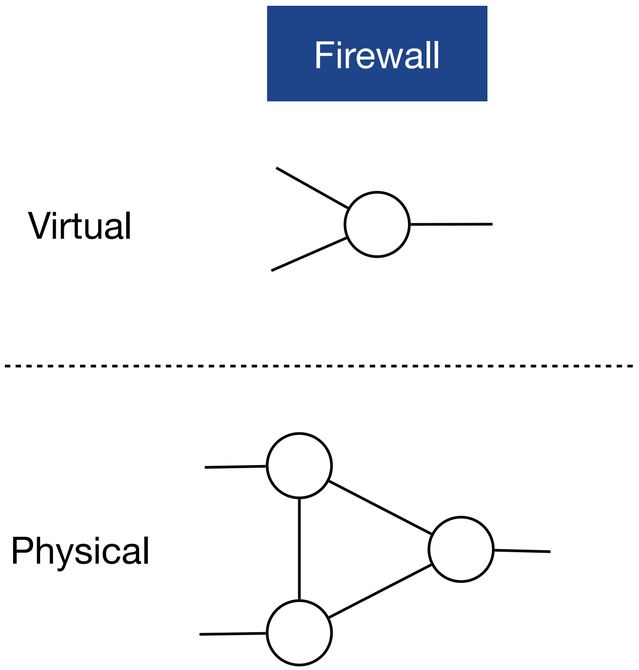
Override operator (▷): one controller chooses to act or defer the processing to another controller

Use multiple operators to compose complex control behaviors

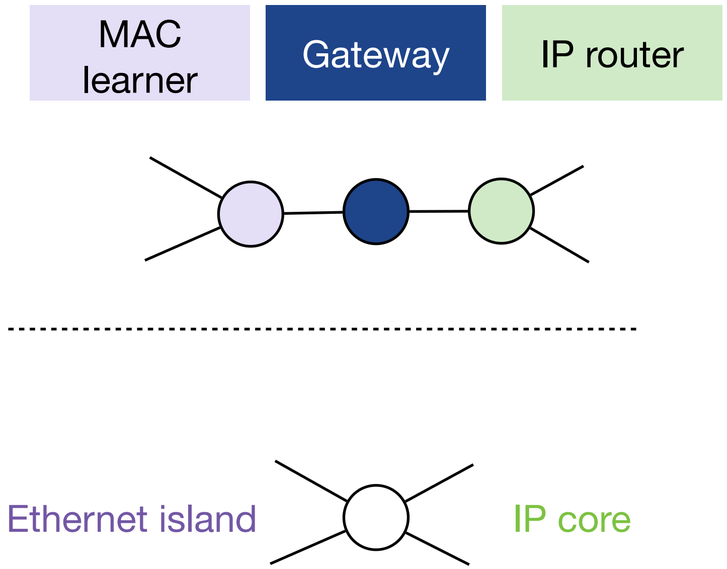
Constraints on topology visibility

Create virtual topology with two primitives: information hiding, controller reuse, composition

Primitive 1: many-to-one



Primitive 2: one-to-many



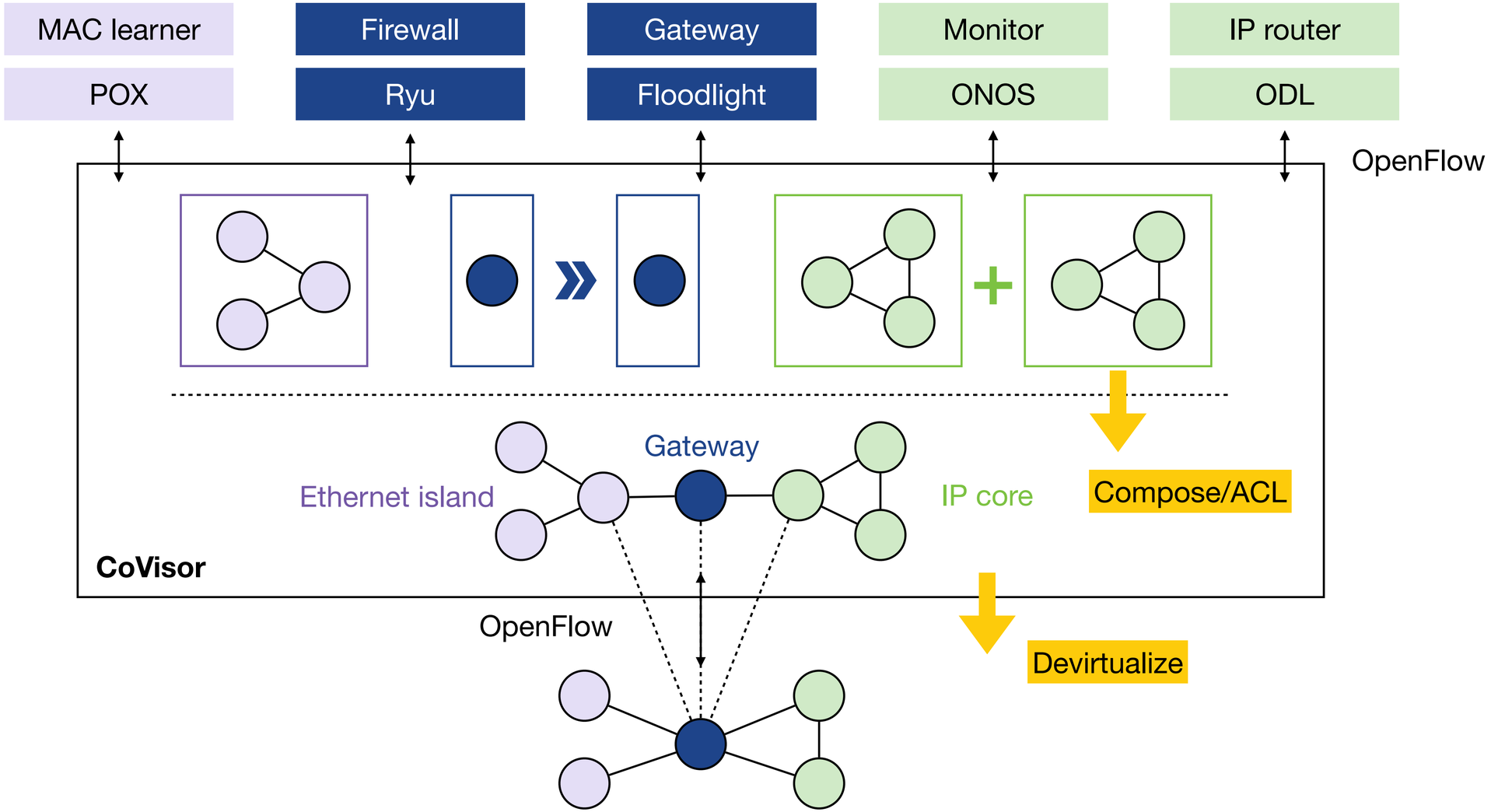
Constraints on packet handling capability

Protect against buggy or malicious third-party control programs

Constraints on **pattern**: header fields, match type
E.g., MAC learner: srcMAC (exact), dstMAC (exact), in_port (exact)

Constraints on **action**: actions to take on matched packets
E.g., MAC learner: forward, drop

CoVisor design overview



Policy composition

Compile all the control policies (lists of rules) from all controllers to the physical network

How to assign priorities to the compiled policies?

?. 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

Naive solution

Assign priorities from top to bottom by decrement of one

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
```

Update overhead

Recompute the entire switch table and assign priorities

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
```

```
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
```

```
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
```

Only two new rules, but three more rules change priorities

Incremental update

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+0=9.  srcip=1.0.0.0/24                   → count
0+7=7.  dstip=2.0.0.0/30                   → fwd(1)
0+0=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(2)
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(2)
0+0=0.  *                                  → drop
```

Only two rule updates

Incremental update

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
```

```
011 001
  3>>1=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
```

Incremental update

Stack priorities for override composition

Special router

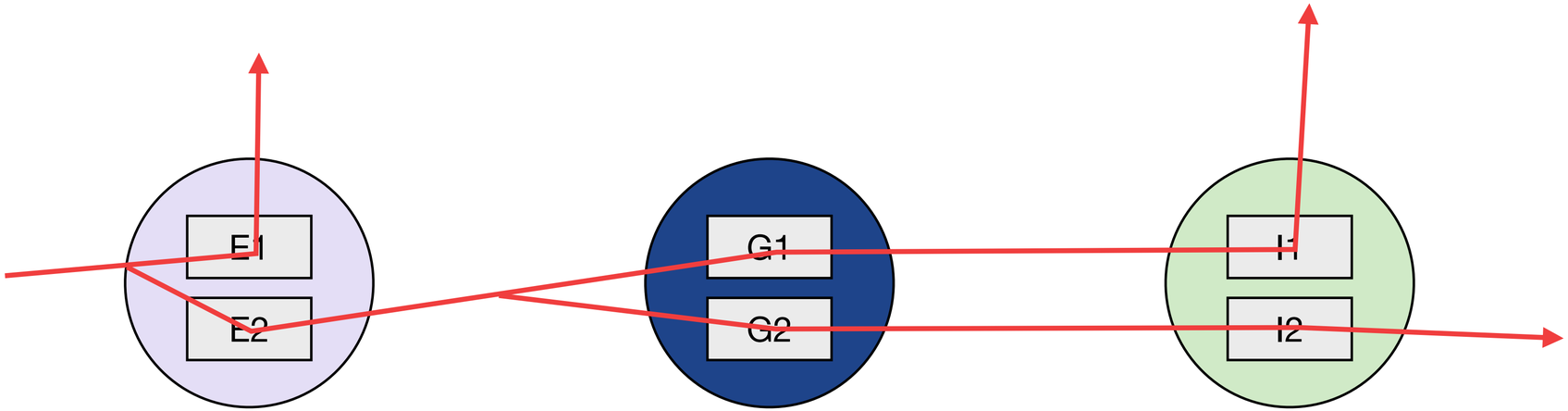
1. srcip=1.0.0.0, dstip=3.0.0.0 → fwd(3)

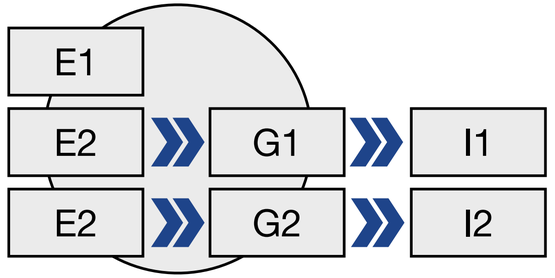
Default router (max priority=8)

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

Compiling one-to-many virtualization



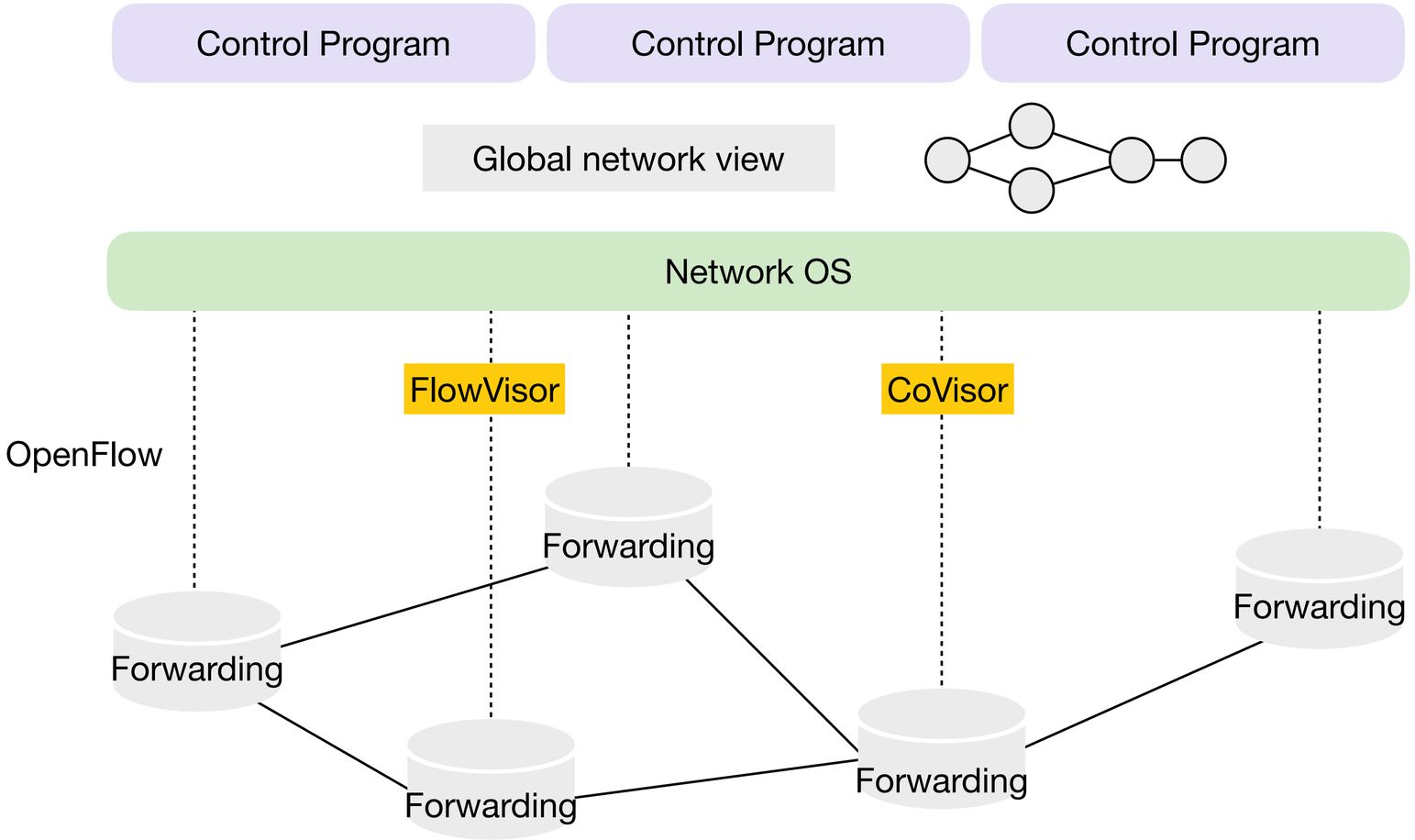


Symbolic path generation

Sequential composition

Priority augmentation

Summary


Control Program
Control Program
Control Program
Global network view
Network OS
FlowVisor
CoVisor
OpenFlow
Forwarding
Forwarding
Forwarding
Forwarding
Forwarding

Next time: network automation