Machine-Verified Network Controllers

Nate Foster
Cornell University

frenetic >>

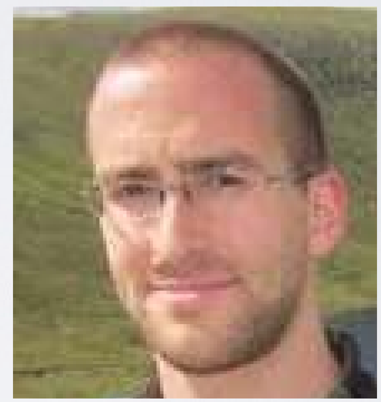
Proof Assistants

Coq

Proof Assistants

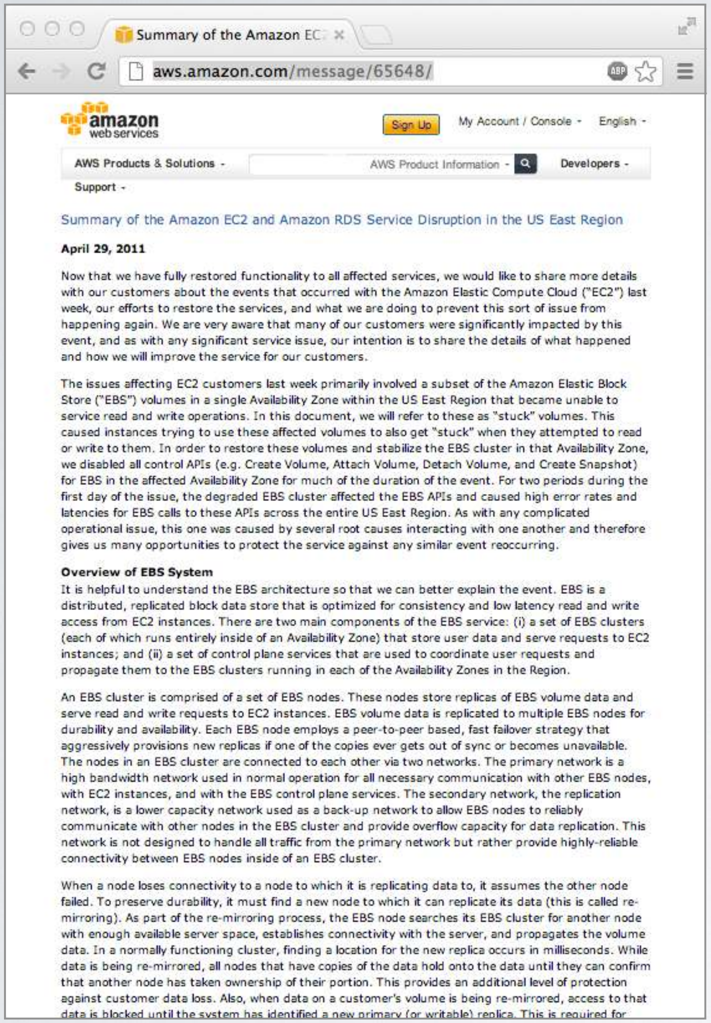
Coq

Arjun Guha
Postdoc → UMass



Mark Reitblatt
PhD student

Networks in Practice



Summary of the Amazon EC

aws.amazon.com/message/65648/

Sign Up My Account / Console English

AWS Products & Solutions AWS Product Information Developers Support

Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region

April 29, 2011

Now that we have fully restored functionality to all affected services, we would like to share more details with our customers about the events that occurred with the Amazon Elastic Compute Cloud ("EC2") last week, our efforts to restore the services, and what we are doing to prevent this sort of issue from happening again. We are very aware that many of our customers were significantly impacted by this event, and as with any significant service issue, our intention is to share the details of what happened and how we will improve the service for our customers.

The issues affecting EC2 customers last week primarily involved a subset of the Amazon Elastic Block Store ("EBS") volumes in a single Availability Zone within the US East Region that became unable to service read and write operations. In this document, we will refer to these as "stuck" volumes. This caused instances trying to use these affected volumes to also get "stuck" when they attempted to read or write to them. In order to restore these volumes and stabilize the EBS cluster in that Availability Zone, we disabled all control APIs (e.g. Create Volume, Attach Volume, Detach Volume, and Create Snapshot) for EBS in the affected Availability Zone for much of the duration of the event. For two periods during the first day of the issue, the degraded EBS cluster affected the EBS APIs and caused high error rates and latencies for EBS calls to these APIs across the entire US East Region. As with any complicated operational issue, this one was caused by several root causes interacting with one another and therefore gives us many opportunities to protect the service against any similar event reoccurring.

Overview of EBS System

It is helpful to understand the EBS architecture so that we can better explain the event. EBS is a distributed, replicated block data store that is optimized for consistency and low latency read and write access from EC2 instances. There are two main components of the EBS service: (i) a set of EBS clusters (each of which runs entirely inside of an Availability Zone) that store user data and serve requests to EC2 instances; and (ii) a set of control plane services that are used to coordinate user requests and propagate them to the EBS clusters running in each of the Availability Zones in the Region.

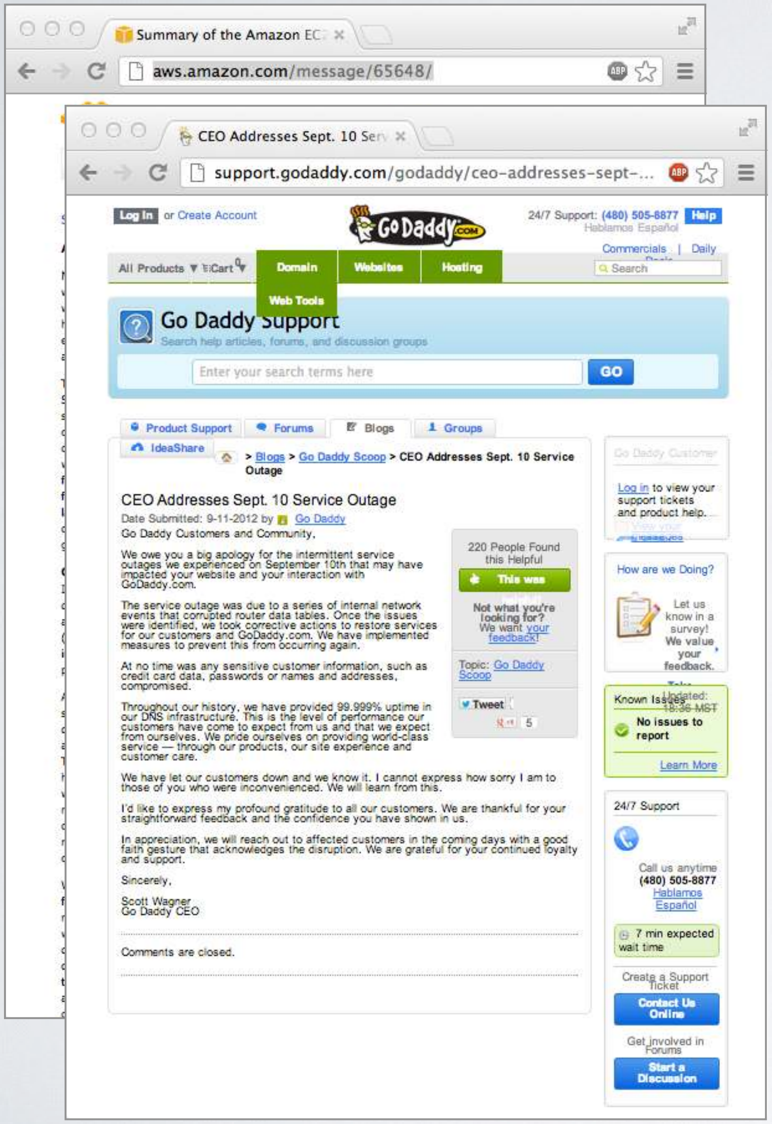
An EBS cluster is comprised of a set of EBS nodes. These nodes store replicas of EBS volume data and serve read and write requests to EC2 instances. EBS volume data is replicated to multiple EBS nodes for durability and availability. Each EBS node employs a peer-to-peer based, fast failover strategy that aggressively provisions new replicas if one of the copies ever gets out of sync or becomes unavailable. The nodes in an EBS cluster are connected to each other via two networks. The primary network is a high bandwidth network used in normal operation for all necessary communication with other EBS nodes, with EC2 instances, and with the EBS control plane services. The secondary network, the replication network, is a lower capacity network used as a back-up network to allow EBS nodes to reliably communicate with other nodes in the EBS cluster and provide overflow capacity for data replication. This network is not designed to handle all traffic from the primary network but rather provide highly-reliable connectivity between EBS nodes inside of an EBS cluster.

When a node loses connectivity to a node to which it is replicating data to, it assumes the other node failed. To preserve durability, it must find a new node to which it can replicate its data (this is called re-mirroring). As part of the re-mirroring process, the EBS node searches its EBS cluster for another node with enough available server space, establishes connectivity with the server, and propagates the volume data. In a normally functioning cluster, finding a location for the new replica occurs in milliseconds. While data is being re-mirrored, all nodes that have copies of the data hold onto the data until they can confirm that another node has taken ownership of their portion. This provides an additional level of protection against customer data loss. Also, when data on a customer's volume is being re-mirrored, access to that data is blocked until the system has identified a new primary (or writable) replica. This is required for

"The trigger for this event was a network configuration change"

—Amazon

Networks in Practice


Summary of the Amazon EC

aws.amazon.com/message/65648/

CEO Addresses Sept. 10 Serv

support.godaddy.com/godaddy/ceo-addresses-sept-...

> Blogs > Go Daddy Scoop > CEO Addresses Sept. 10 Service Outage

CEO Addresses Sept. 10 Service Outage

Date Submitted: 9-11-2012 by Go Daddy

Go Daddy Customers and Community,

We owe you a big apology for the intermittent service outages we experienced on September 10th that may have impacted your website and your interaction with GoDaddy.com.

The service outage was due to a series of internal network events that corrupted router data tables. Once the issues were identified, we took corrective actions to restore services for our customers and GoDaddy.com. We have implemented measures to prevent this from occurring again.

At no time was any sensitive customer information, such as credit card data, passwords or names and addresses, compromised.

Throughout our history, we have provided 99.999% uptime in our DNS infrastructure. This is the level of performance our customers have come to expect from us and that we expect from ourselves. We pride ourselves on providing world-class service — through our products, our site experience and customer care.

We have let our customers down and we know it. I cannot express how sorry I am to those of you who were inconvenienced. We will learn from this.

I'd like to express my profound gratitude to all our customers. We are thankful for your straightforward feedback and the confidence you have shown in us.

In appreciation, we will reach out to affected customers in the coming days with a good faith gesture that acknowledges the disruption. We are grateful for your continued loyalty and support.

Sincerely,

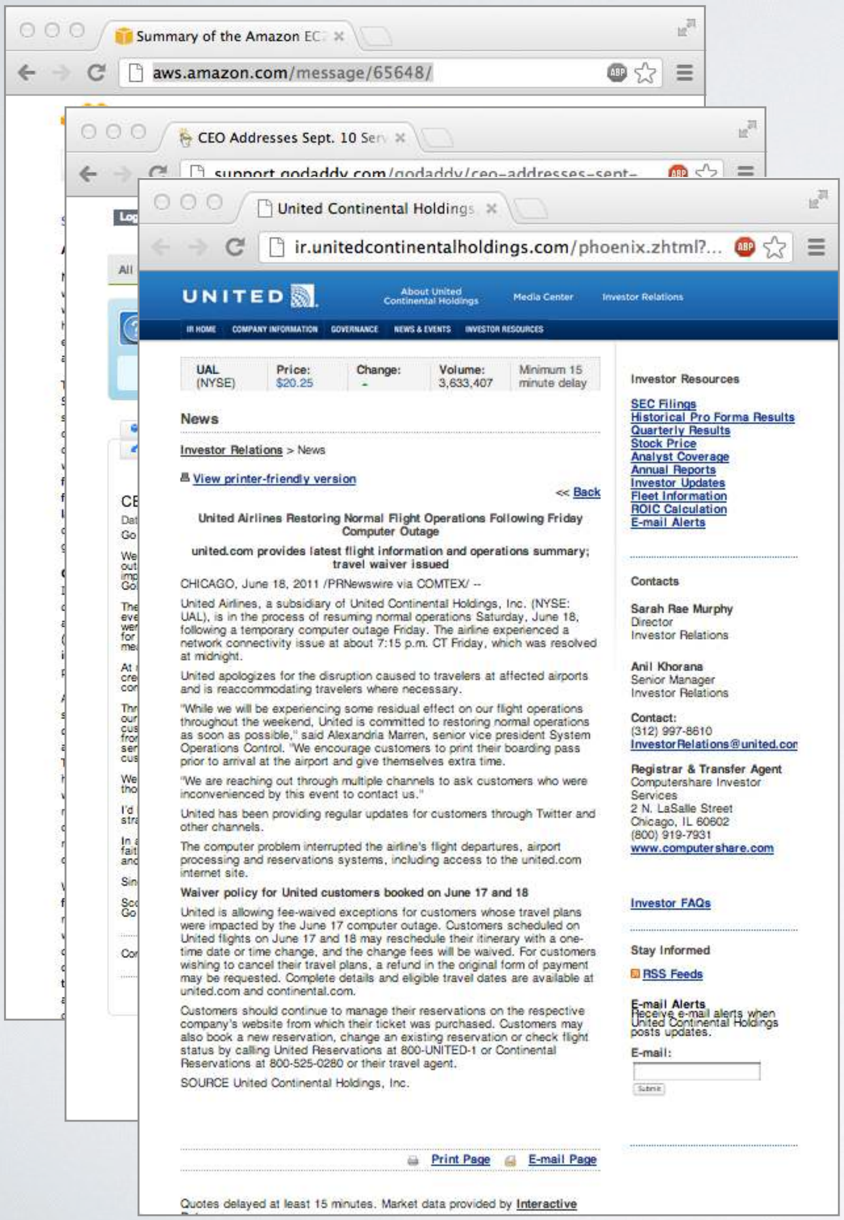
Scott Wagner
Go Daddy CEO

Comments are closed.

"The service outage was due to a series of internal network events that corrupted router data tables"

—GoDaddy

Networks in Practice


Summary of the Amazon EC
aws.amazon.com/message/65648/
CEO Addresses Sept. 10 Serv
support.godaddy.com/godaddy/ceo-addresses-sept-
United Continental Holdings
ir.unitedcontinentalholdings.com/phoenix.zhtml?...

UNITED

About United Continental Holdings | Media Center | Investor Relations

IR HOME | COMPANY INFORMATION | GOVERNANCE | NEWS & EVENTS | INVESTOR RESOURCES

UAL (NYSE)	Price: $20.25	Change: -	Volume: 3,633,407	Minimum 15 minute delay

News

Investor Relations > News

View printer-friendly version

<< Back

United Airlines Restoring Normal Flight Operations Following Friday Computer Outage

united.com provides latest flight information and operations summary; travel waiver issued

CHICAGO, June 18, 2011 /PRNewswire via COMTEX/ --

United Airlines, a subsidiary of United Continental Holdings, Inc. (NYSE: UAL), is in the process of resuming normal operations Saturday, June 18, following a temporary computer outage Friday. The airline experienced a network connectivity issue at about 7:15 p.m. CT Friday, which was resolved at midnight.

United apologizes for the disruption caused to travelers at affected airports and is reaccommodating travelers where necessary.

"While we will be experiencing some residual effect on our flight operations throughout the weekend, United is committed to restoring normal operations as soon as possible," said Alexandria Marren, senior vice president System Operations Control. "We encourage customers to print their boarding pass prior to arrival at the airport and give themselves extra time.

"We are reaching out through multiple channels to ask customers who were inconvenienced by this event to contact us."

United has been providing regular updates for customers through Twitter and other channels.

The computer problem interrupted the airline's flight departures, airport processing and reservations systems, including access to the united.com internet site.

Waiver policy for United customers booked on June 17 and 18

United is allowing fee-waived exceptions for customers whose travel plans were impacted by the June 17 computer outage. Customers scheduled on United flights on June 17 and 18 may reschedule their itinerary with a one-time date or time change, and the change fees will be waived. For customers wishing to cancel their travel plans, a refund in the original form of payment may be requested. Complete details and eligible travel dates are available at united.com and continental.com.

Customers should continue to manage their reservations on the respective company's website from which their ticket was purchased. Customers may also book a new reservation, change an existing reservation or check flight status by calling United Reservations at 800-UNITED-1 or Continental Reservations at 800-525-0280 or their travel agent.

SOURCE United Continental Holdings, Inc.

Print Page | E-mail Page

Quotes delayed at least 15 minutes. Market data provided by Interactive

Investor Resources

SEC Filings
Historical Pro Forma Results
Quarterly Results
Stock Price
Analyst Coverage
Annual Reports
Investor Updates
Fleet Information
ROIC Calculation
E-mail Alerts

Contacts

Sarah Rae Murphy
Director
Investor Relations

Anil Khorana
Senior Manager
Investor Relations

Contact:
(312) 997-8610
InvestorRelations@united.com

Registrar & Transfer Agent
Computershare Investor Services
2 N. LaSalle Street
Chicago, IL 60602
(800) 919-7931
www.computershare.com

Investor FAQs

Stay Informed

RSS Feeds

E-mail Alerts
Receive e-mail alerts when United Continental Holdings posts updates.

E-mail:

"The airline experienced a network connectivity issue..."

—United Airlines

Networks in Practice

"The airline experienced a network connectivity issue..."

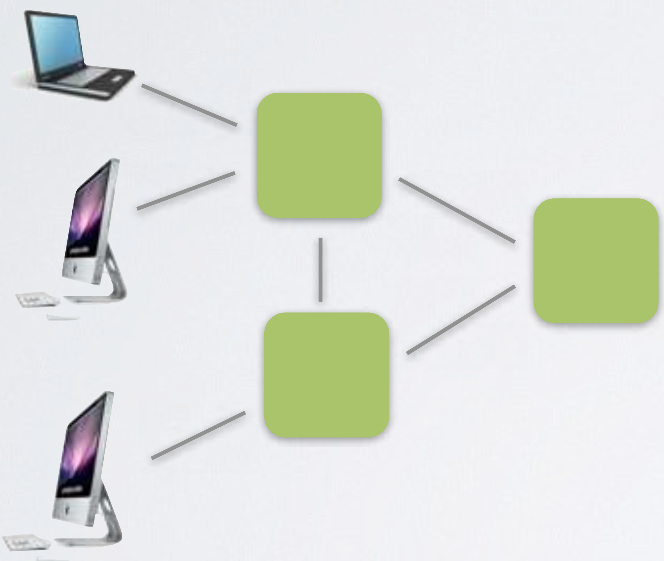
—United Airlines

Networks in Practice

There are hosts...

Networks in Practice

Connected by switches...

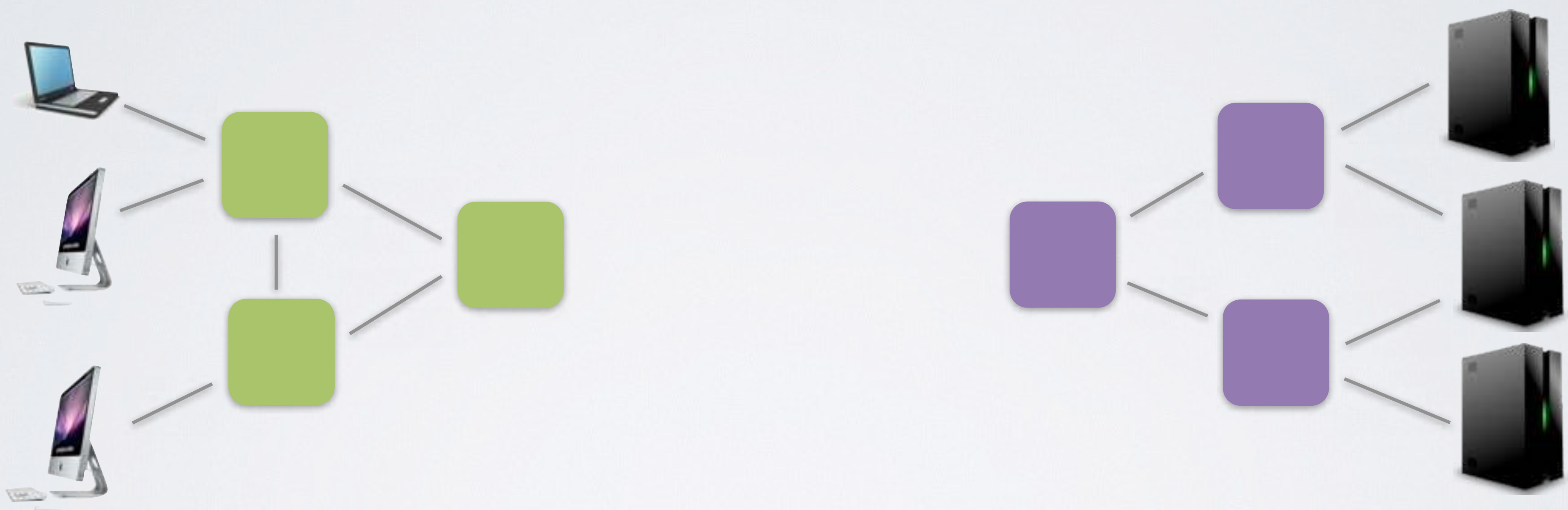


Networks in Practice

There are also servers...

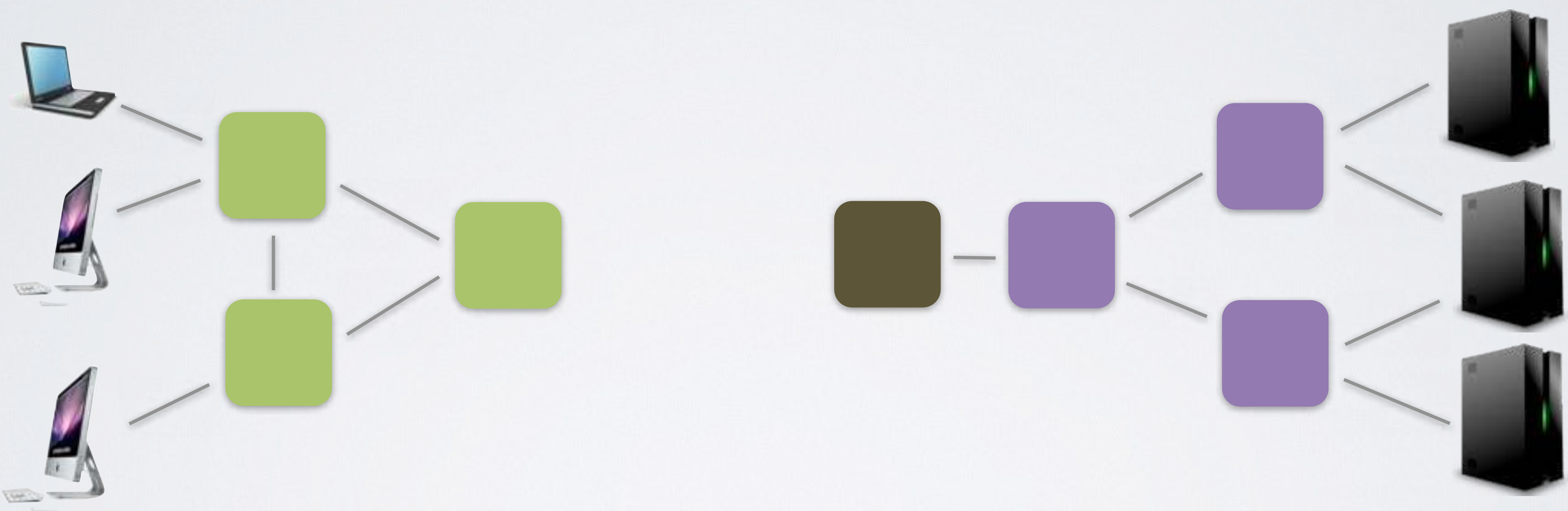
Networks in Practice

Connected by routers...



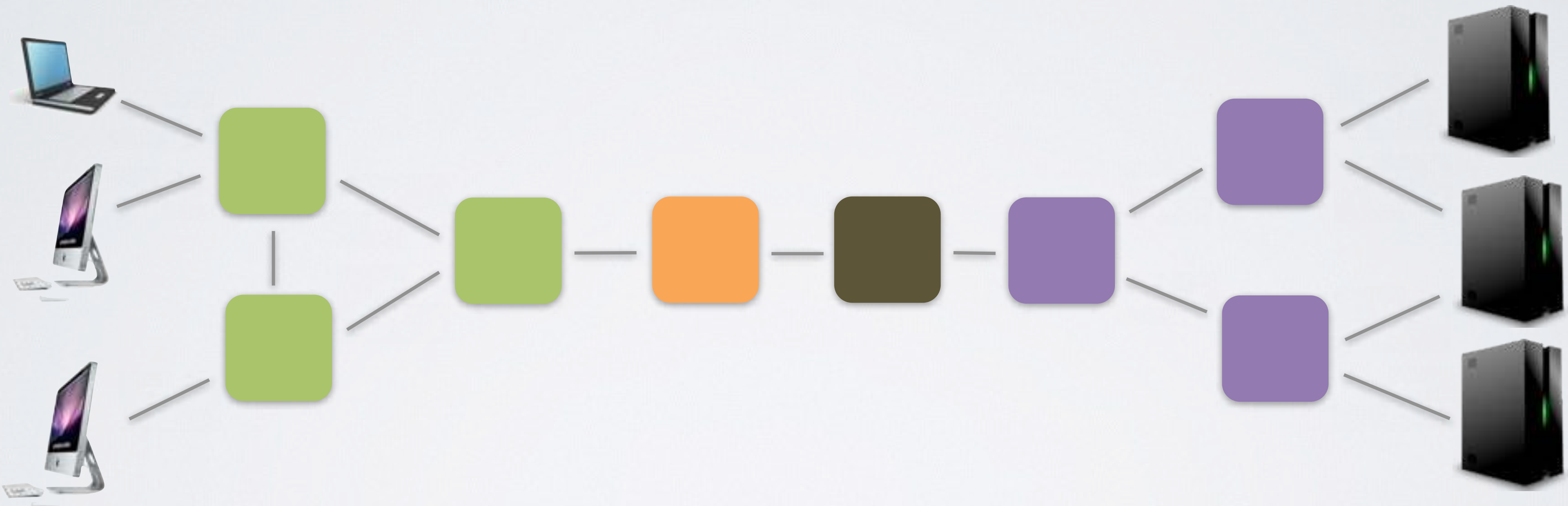
Networks in Practice

And a load balancer...



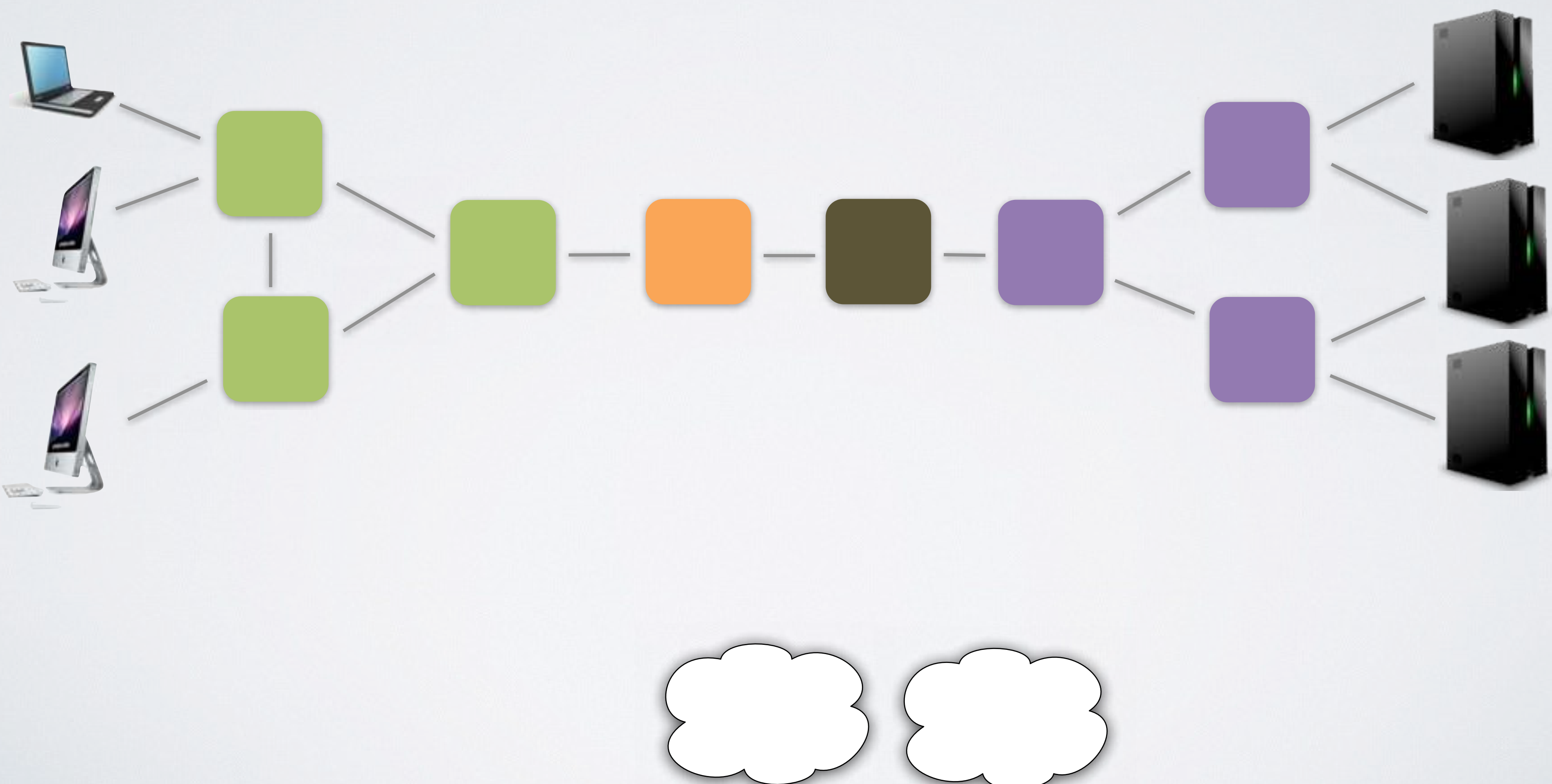
Networks in Practice

And a gateway router...



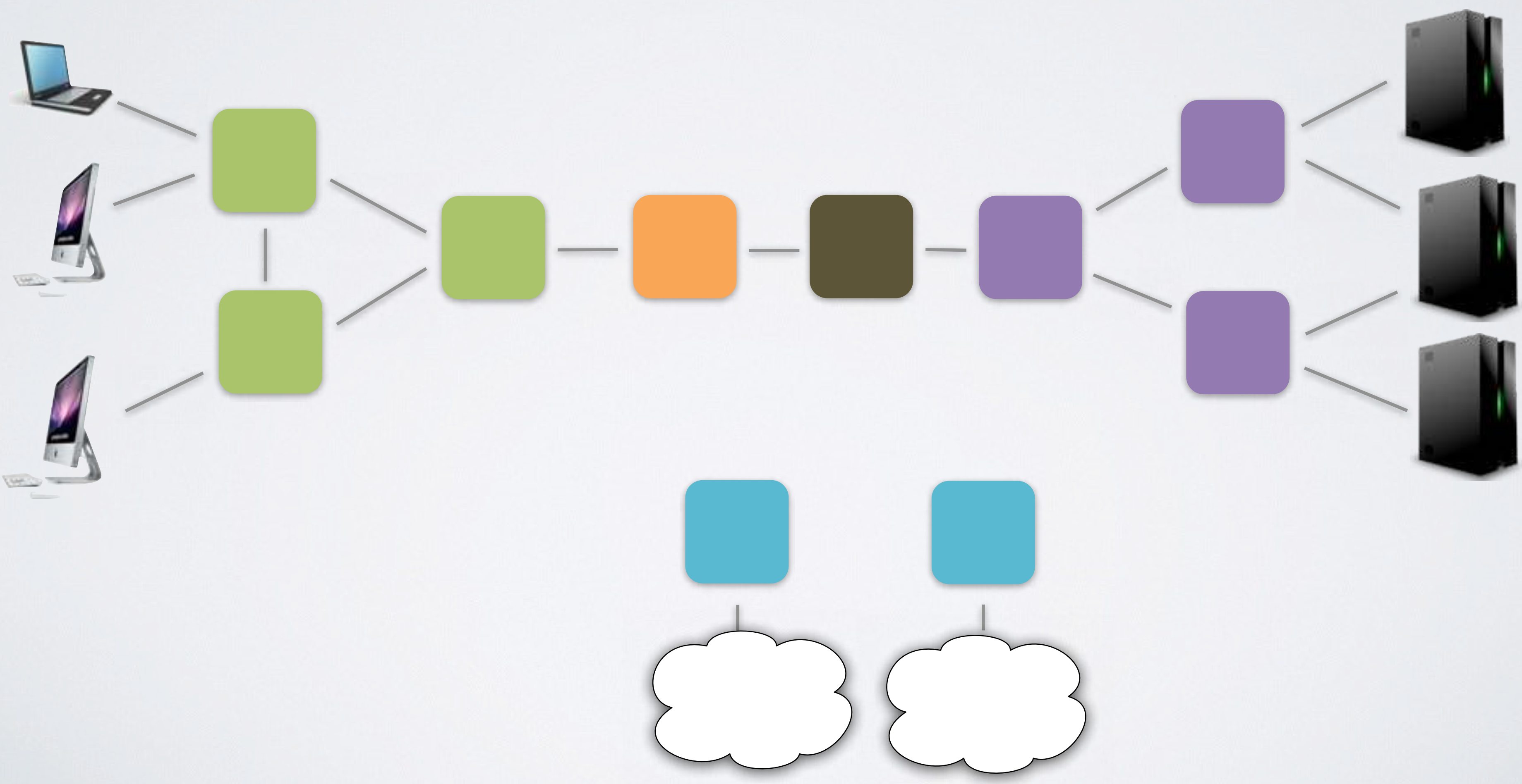
Networks in Practice

There are other ISPs...



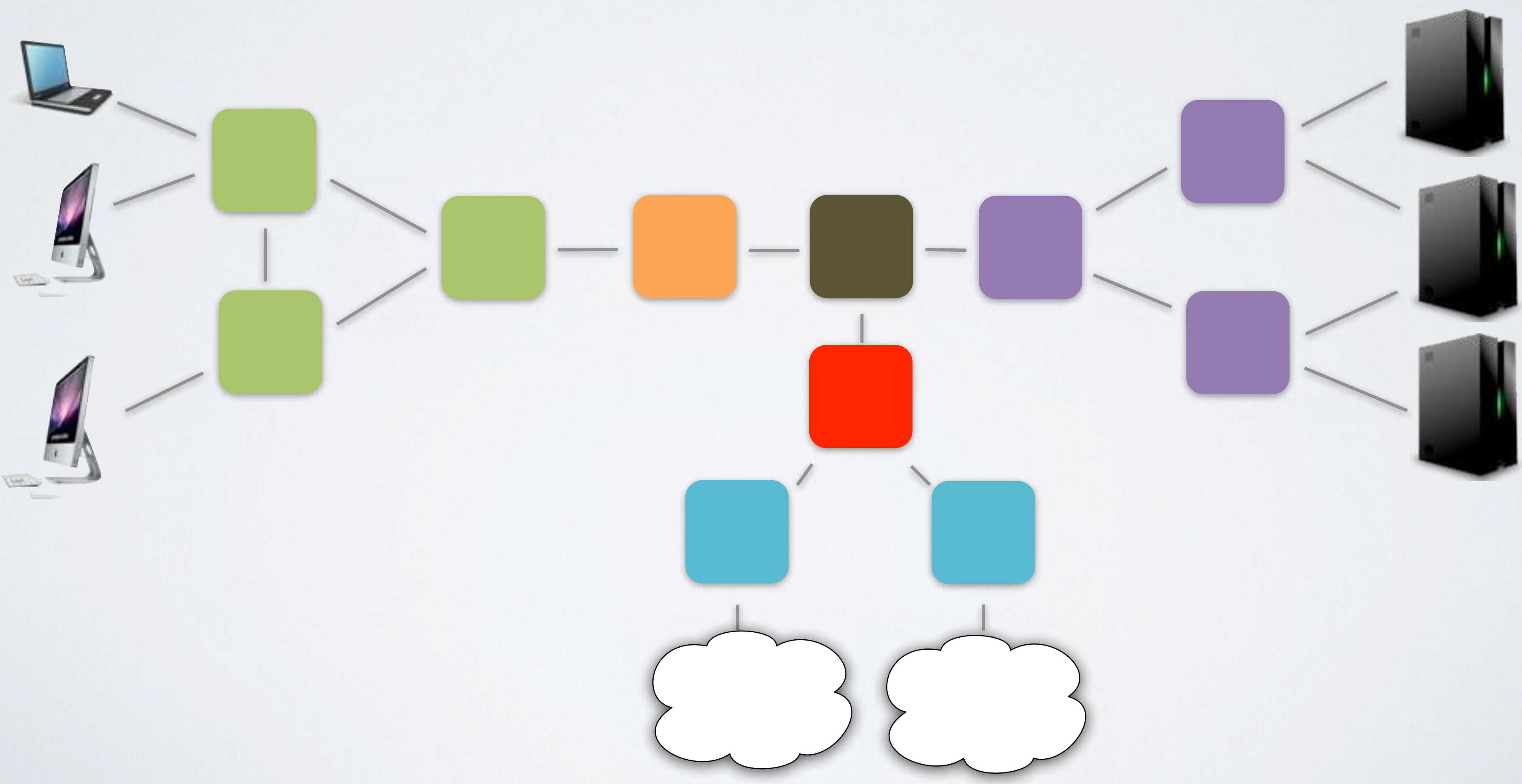
Networks in Practice

So we need to run BGP...



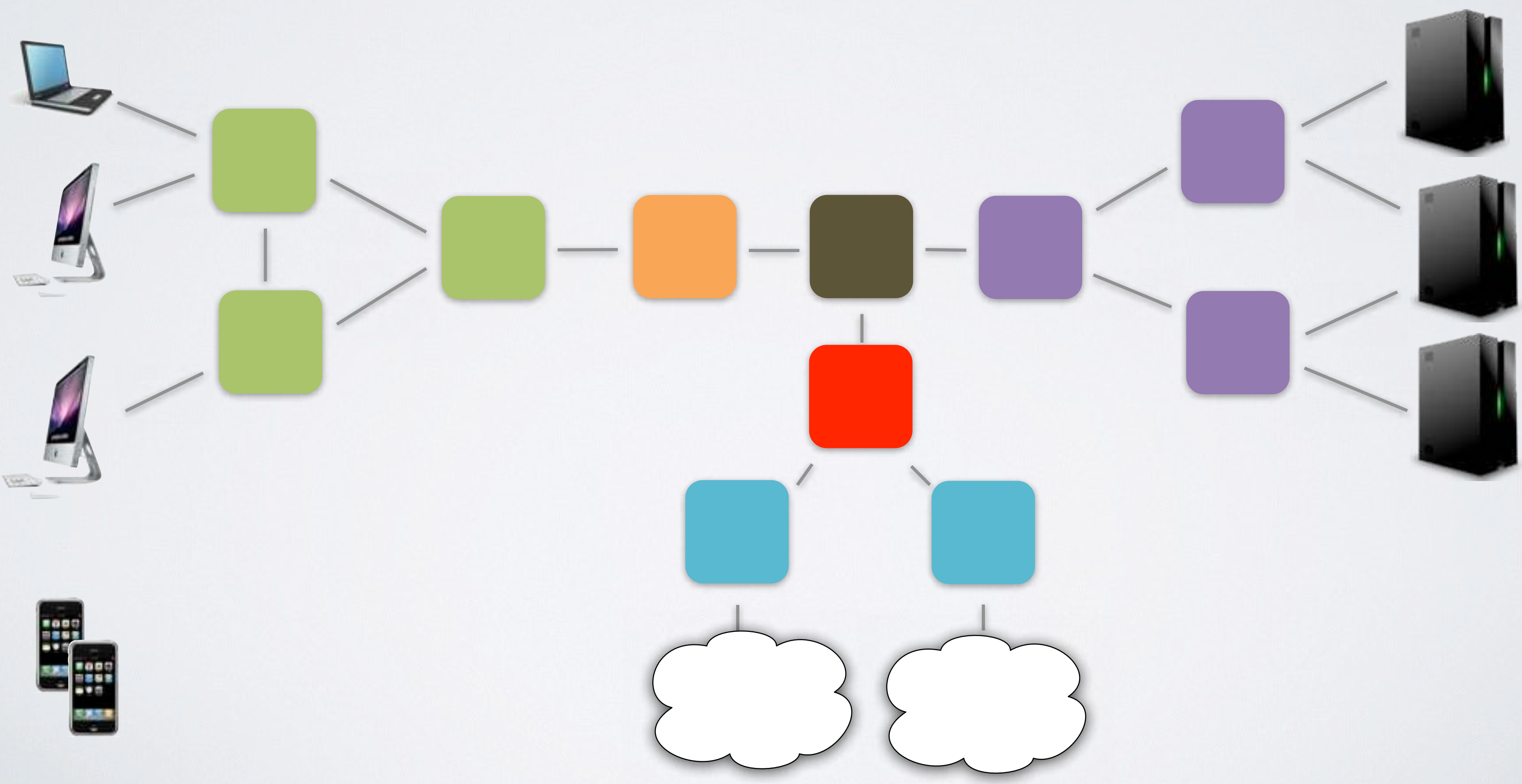
Networks in Practice

And we need a firewall to filter incoming traffic...



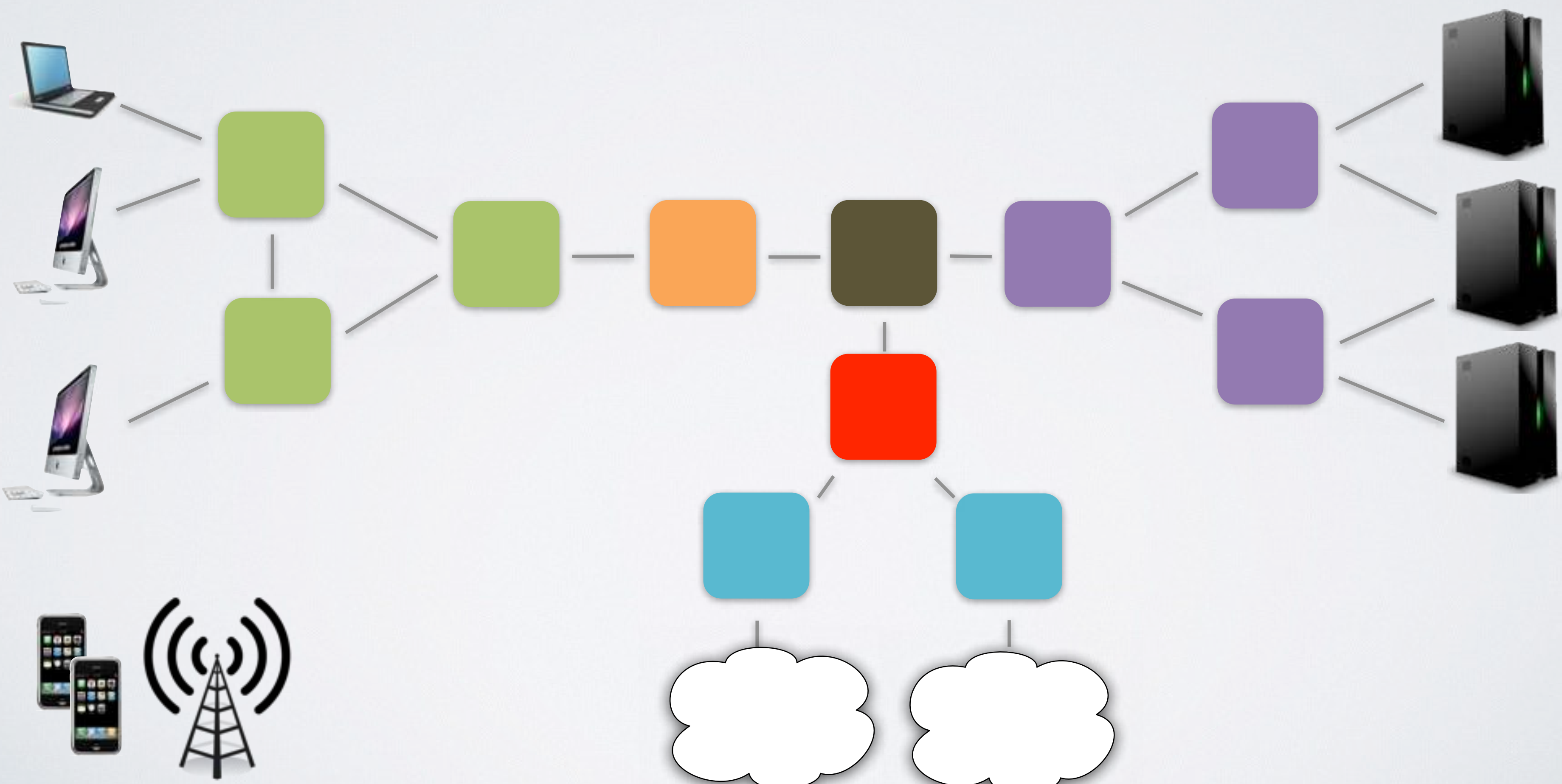
Networks in Practice

There are also wireless hosts...



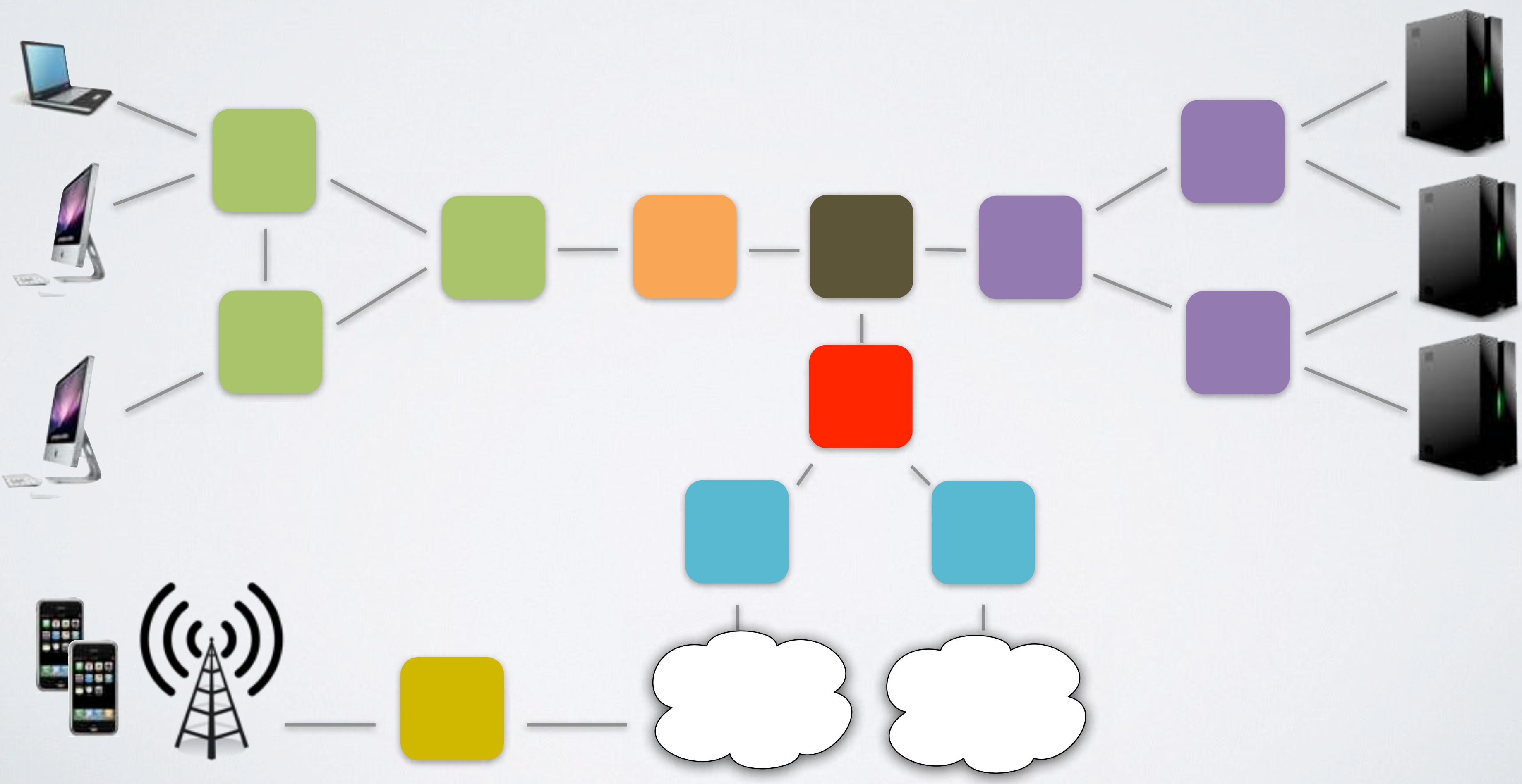
Networks in Practice

So we need wireless gateways...



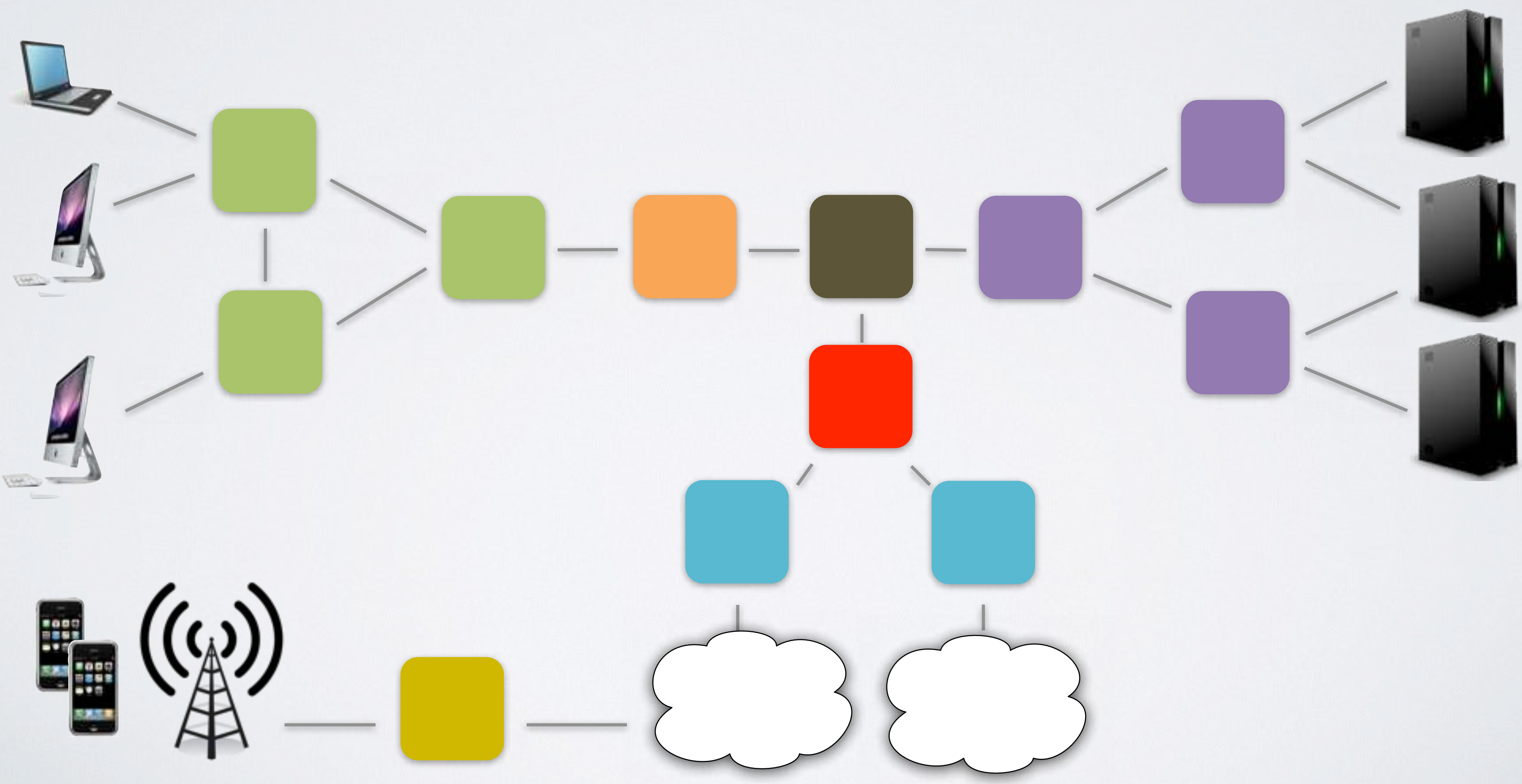
Networks in Practice

And yet more middleboxes for lawful intercept...



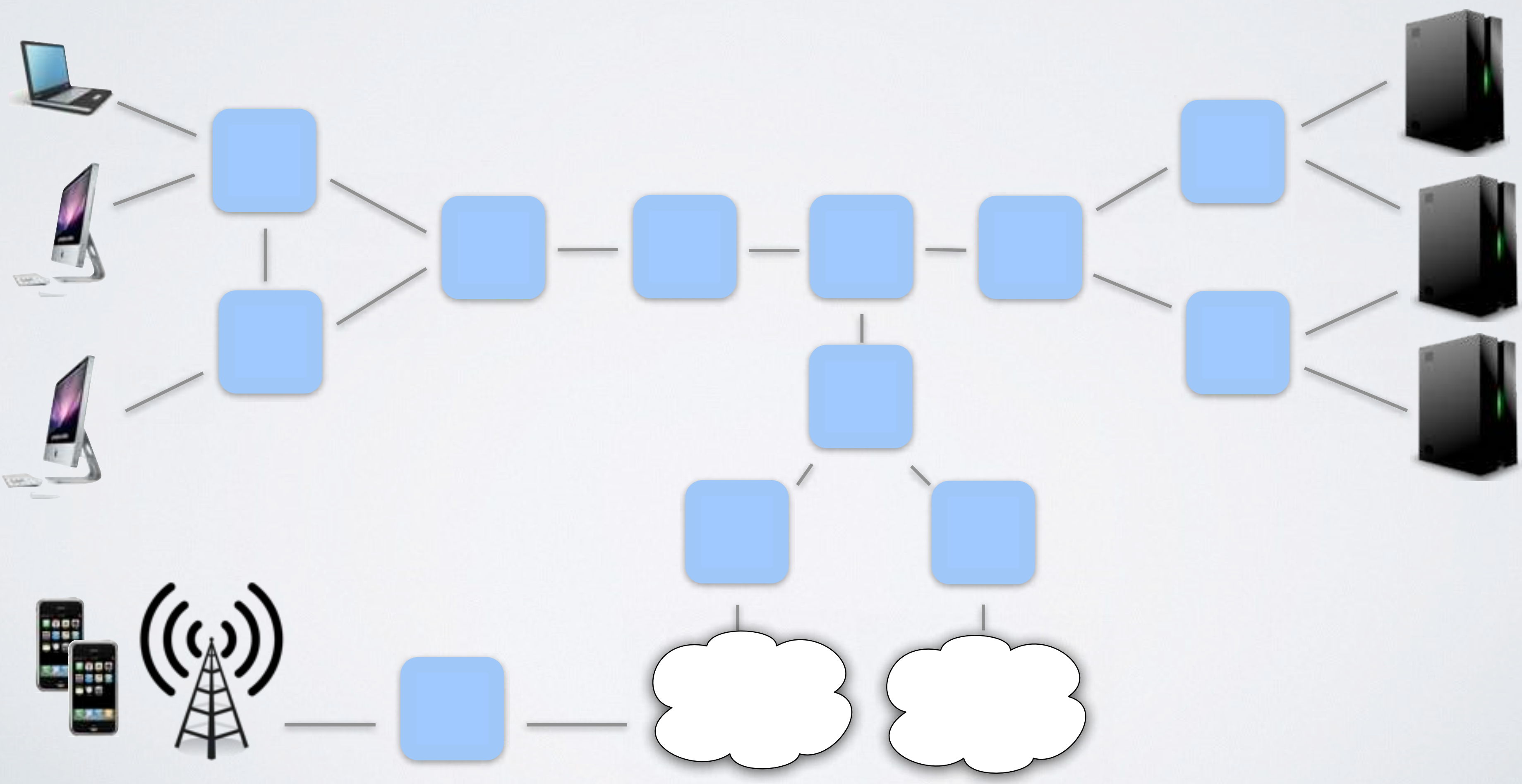
Networks in Practice

Each color represents a different set of control plane protocols and algorithms... this is



Software-Defined Networking

A clean-slate architecture that standardizes features and decouples forwarding from



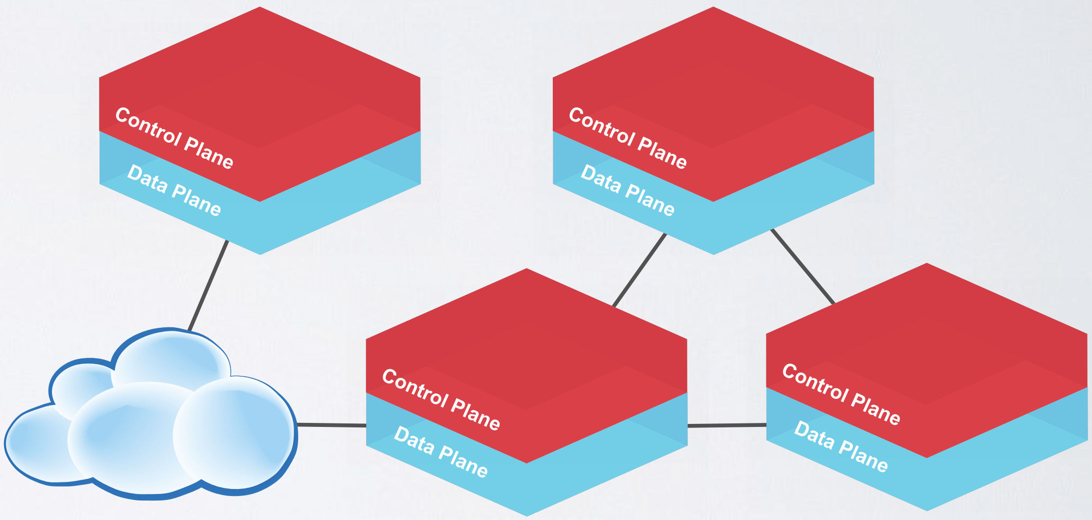
Software-Defined Networking

Essential ingredients

- Decouple control and data planes
- Logically-centralized control

Enables

- Novel functionality
- Formal reasoning



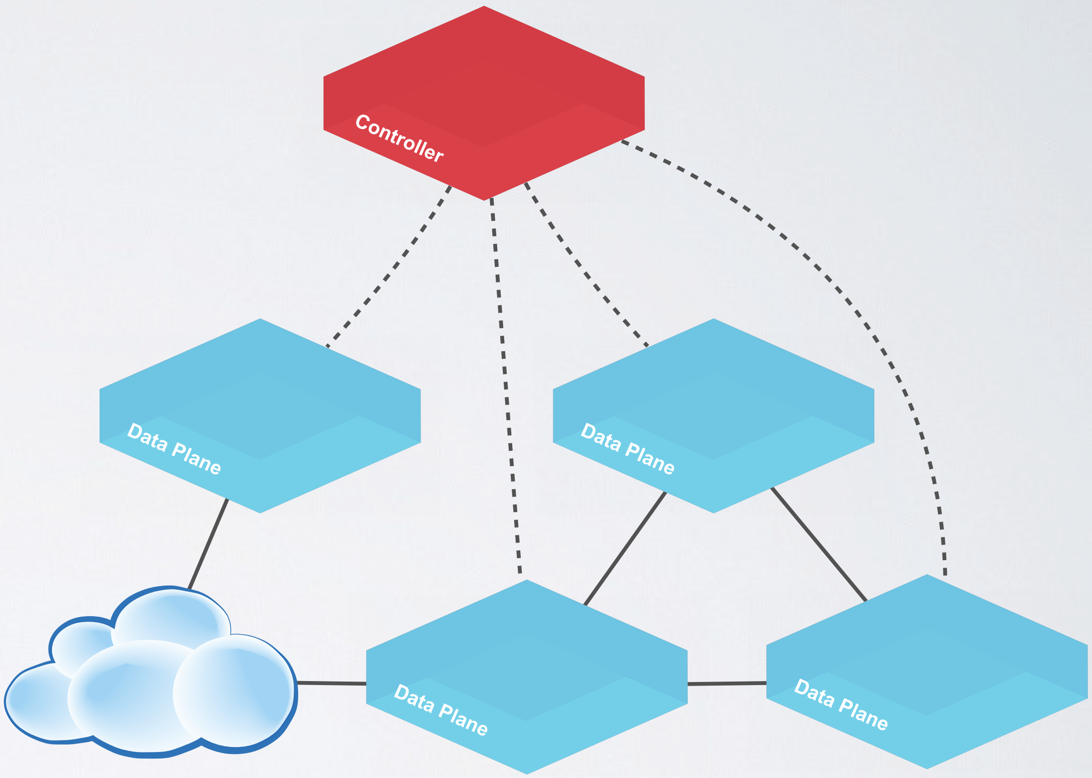
Software-Defined Networking

Essential ingredients

- Decouple control and data planes
- Logically-centralized control

Enables

- Novel functionality
- Formal reasoning



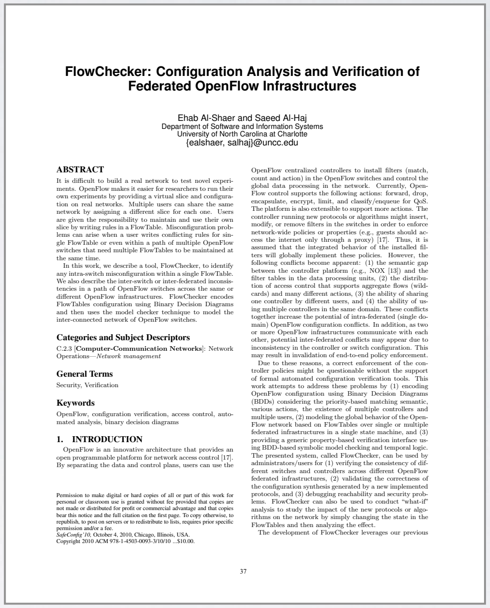
Existing Tools

There is a cottage industry in SDN configuration-checking tools...

Existing Tools

There is a cottage industry in SDN configuration-checking tools...

- FlowChecker [SafeConfig '10]



FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures

Ehab Al-Shaer and Saeed Al-Haj
Department of Software and Information Systems
University of North Carolina at Charlotte
{ealshaer, salhaj}@uncc.edu

ABSTRACT

It is difficult to build a real network to test novel experiments. OpenFlow makes it easier for researchers to run their own experiments by providing a virtual slice and configuration on real networks. Multiple users can share the same network by assigning a different slice for each one. Users are given the responsibility to maintain and use their own slice by writing rules in a FlowTable. Misconfiguration problems can arise when a user writes conflicting rules for single FlowTable or even within a path of multiple OpenFlow switches that need multiple FlowTables to be maintained at the same time.

In this work, we describe a tool, FlowChecker, to identify any intra-switch misconfiguration within a single FlowTable. We also describe the inter-switch or inter-federated inconsistencies in a path of OpenFlow switches across the same or different OpenFlow infrastructures. FlowChecker encodes FlowTables configuration using Binary Decision Diagrams and then uses the model checker technique to model the inter-connected network of OpenFlow switches.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

General Terms

Security, Verification

Keywords

OpenFlow, configuration verification, access control, automated analysis, binary decision diagrams

1. INTRODUCTION

OpenFlow is an innovative architecture that provides an open programmable platform for network access control [17]. By separating the data and control plans, users can use the OpenFlow centralized controllers to install filters (match, count and action) in the OpenFlow switches and control the global data processing in the network. Currently, OpenFlow control supports the following actions: forward, drop, encapsulate, encrypt, limit, and classify/enqueue for QoS. The platform is also extensible to support more actions. The controller running new protocols or algorithms might insert, modify, or remove filters in the switches in order to enforce network-wide policies or properties (e.g., guests should access the internet only through a proxy) [17]. Thus, it is assumed that the integrated behavior of the installed filters will globally implement these policies. However, the following conflicts become apparent: (1) the semantic gap between the controller platform (e.g., NOX [13]) and the filter tables in the data processing units, (2) the distribution of access control that supports aggregate flows (wildcards) and many different actions, (3) the ability of sharing one controller by different users, and (4) the ability of using multiple controllers in the same domain. These conflicts together increase the potential of intra-federated (single domain) OpenFlow configuration conflicts. In addition, as two or more OpenFlow infrastructures communicate with each other, potential inter-federated conflicts may appear due to inconsistency in the controller or switch configuration. This may result in invalidation of end-to-end policy enforcement.

Due to these reasons, a correct enforcement of the controller policies might be questionable without the support of formal automated configuration verification tools. This work attempts to address these problems by (1) encoding OpenFlow configuration using Binary Decision Diagrams (BDDs) considering the priority-based matching semantic, various actions, the existence of multiple controllers and multiple users, (2) modeling the global behavior of the OpenFlow network based on FlowTables over single or multiple federated infrastructures in a single state machine, and (3) providing a generic property-based verification interface using BDD-based symbolic model checking and temporal logic. The presented system, called FlowChecker, can be used by administrators/users for (1) verifying the consistency of different switches and controllers across different OpenFlow federated infrastructures, (2) validating the correctness of the configuration synthesis generated by a new implemented protocols, and (3) debugging reachability and security problems. FlowChecker can also be used to conduct "what-if" analysis to study the impact of the new protocols or algorithms on the network by simply changing the state in the FlowTables and then analyzing the effect.

The development of FlowChecker leverages our previous

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SafeConfig'10, October 4, 2010, Chicago, Illinois, USA.
Copyright 2010 ACM 978-1-4503-0093-3/10/10 ...$10.00.

Existing Tools

There is a cottage industry in SDN configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]

Existing Tools

There is a cottage industry in SDN configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]

Existing Tools

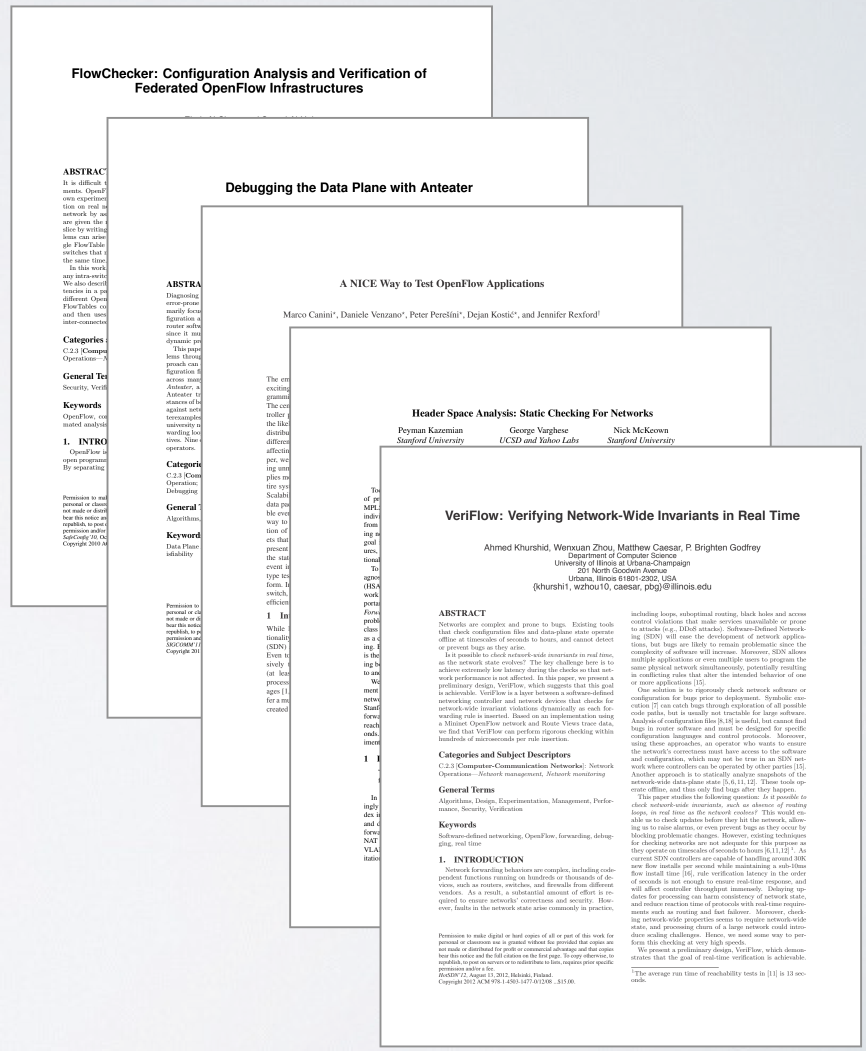
There is a cottage industry in SDN configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]
- Header Space Analysis [NSDI '12]

Existing Tools

There is a cottage industry in SDN configuration-checking tools...

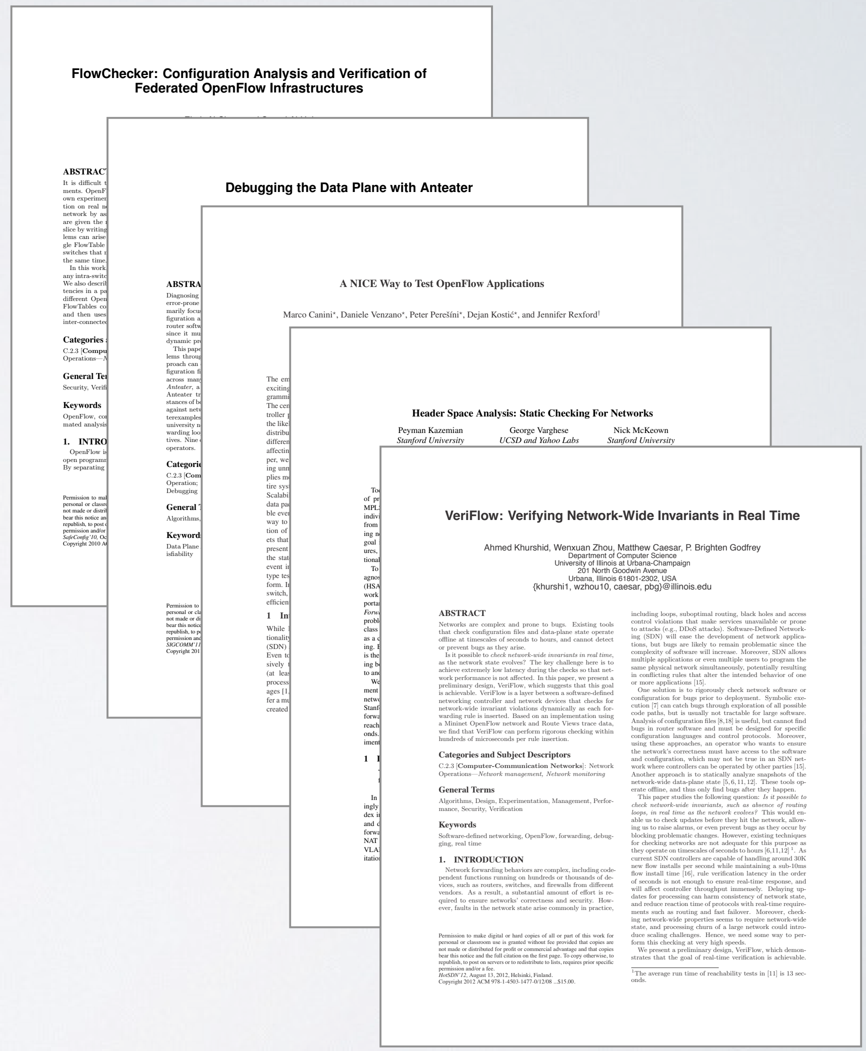
- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]
- Header Space Analysis [NSDI '12]
- VeriFlow [HotSDN '12]
- and many others...



Existing Tools

There is a cottage industry in SDN configuration-checking tools...

- FlowChecker [SafeConfig '10]
- AntEater [SIGCOMM '11]
- NICE [NSDI '12]
- Header Space Analysis [NSDI '12]
- VeriFlow [HotSDN '12]
- and many others...



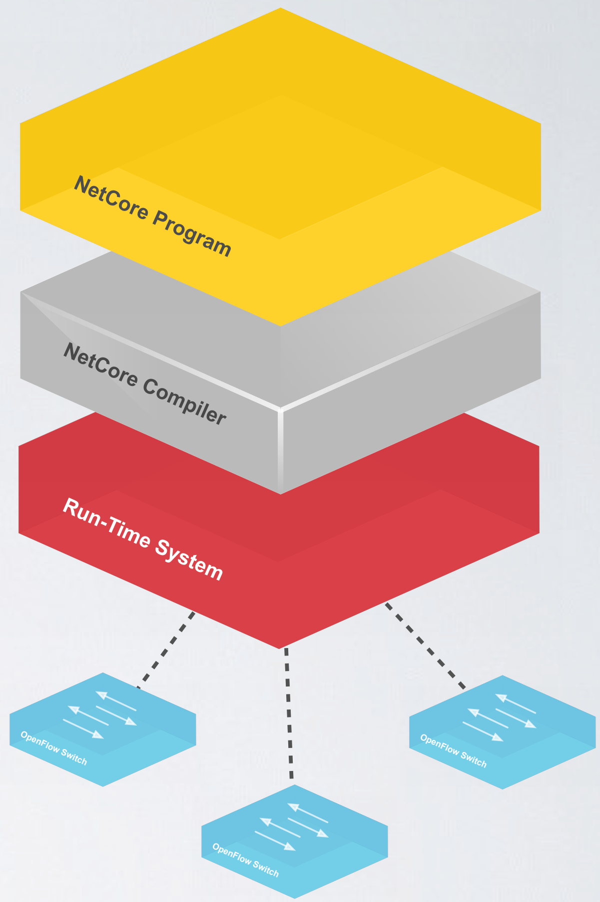
These are all great tools!

But they are expensive to run, and each builds on a custom (typically ad hoc) foundation

Machine-Verified Controllers

Vision

- Develop programs in a high-level language
- Reason at a high level of abstraction
- Use a compiler and run-time system to generate low-level control messages
- Machine-verified proofs of correctness

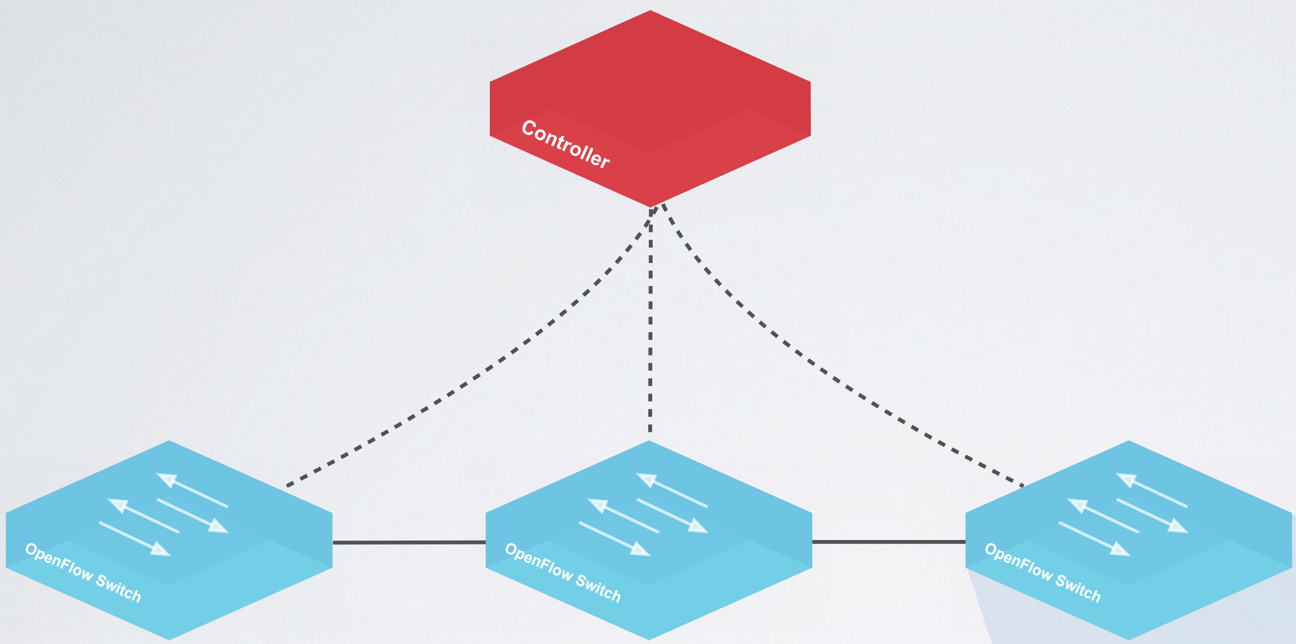


Contributions

- NetCore compiler + optimizer
- Featherweight OpenFlow model
- General framework for establishing run-time system correctness

OVERVIEW

OpenFlow Switches



Forwarding Table: prioritized list of rules

Rule: pattern, actions, and counters

Pattern: prefix match on headers

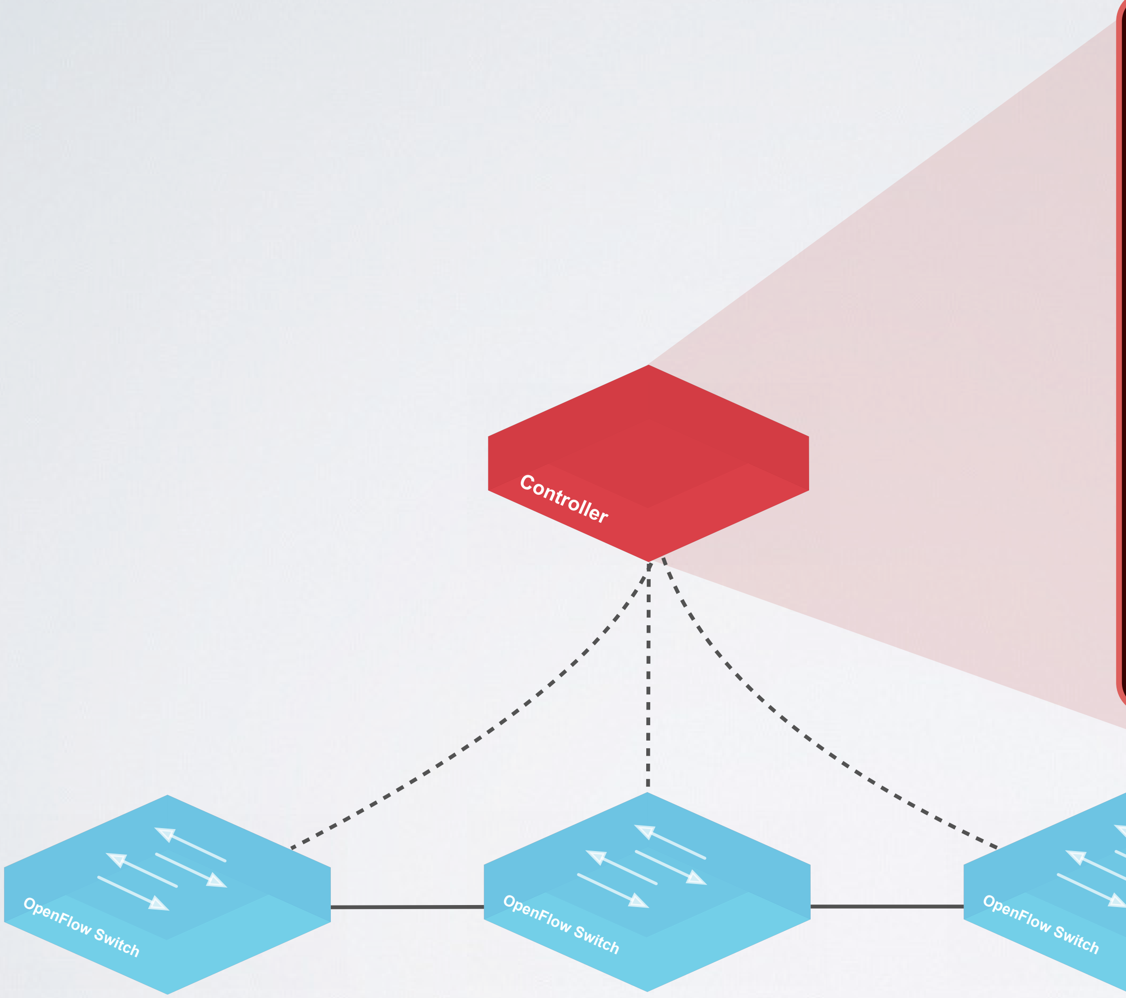
Action: forward or modify

Counters: total bytes and packets processed

OpenFlow

Pattern	Action	Bytes	Packets
1010	Drop	200	10
010*	Forward(2)	100	4
011*	Controller	0	0

Priority ↓



NOX

Network Events

- Topology changes
- **Diverted packets**
- Traffic statistics

Control Messages

- **Modify rules**
- Query counters

Issue #1: Switch-Level Errors

What happens if...

- The controller misses a keep-alive message?
- The controller sends a malformed message?
  - Bad output port
  - Too many actions
  - Inconsistent actions
  - Unsupported actions
- The switches runs out of space for rules?

Any of these can lead to essentially arbitrary behavior

Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*”, ... },
                  actions = [ flood ], ... }
```

Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*”, ... },
                  actions = [ flood ], ... }
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...

Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*”, ... },
                  actions = [ flood ], ... }
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...

...but it actually installs a rule that floods *all* traffic

Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*”, ... },
                  actions = [ flood ], ... }
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...

...but it actually installs a rule that floods *all* traffic

Why? Switches *silently* ignore IP fields unless the Ethernet frame type is IP!

Issue #3: Message Reordering

What happens if the controller sends the following pair of OpenFlow messages to a switch in sequence?

```
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                            srcIPAddress =
"10.0.1.99", ... },
                  priority = 1,
                  actions = [ ] }
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                            srcIPAddress = "10.0.1.*", ... },
                  priority = 2,
                  actions = [ flood ] }
```

The intention is to encode a negation...

Issue #3: Message Reordering

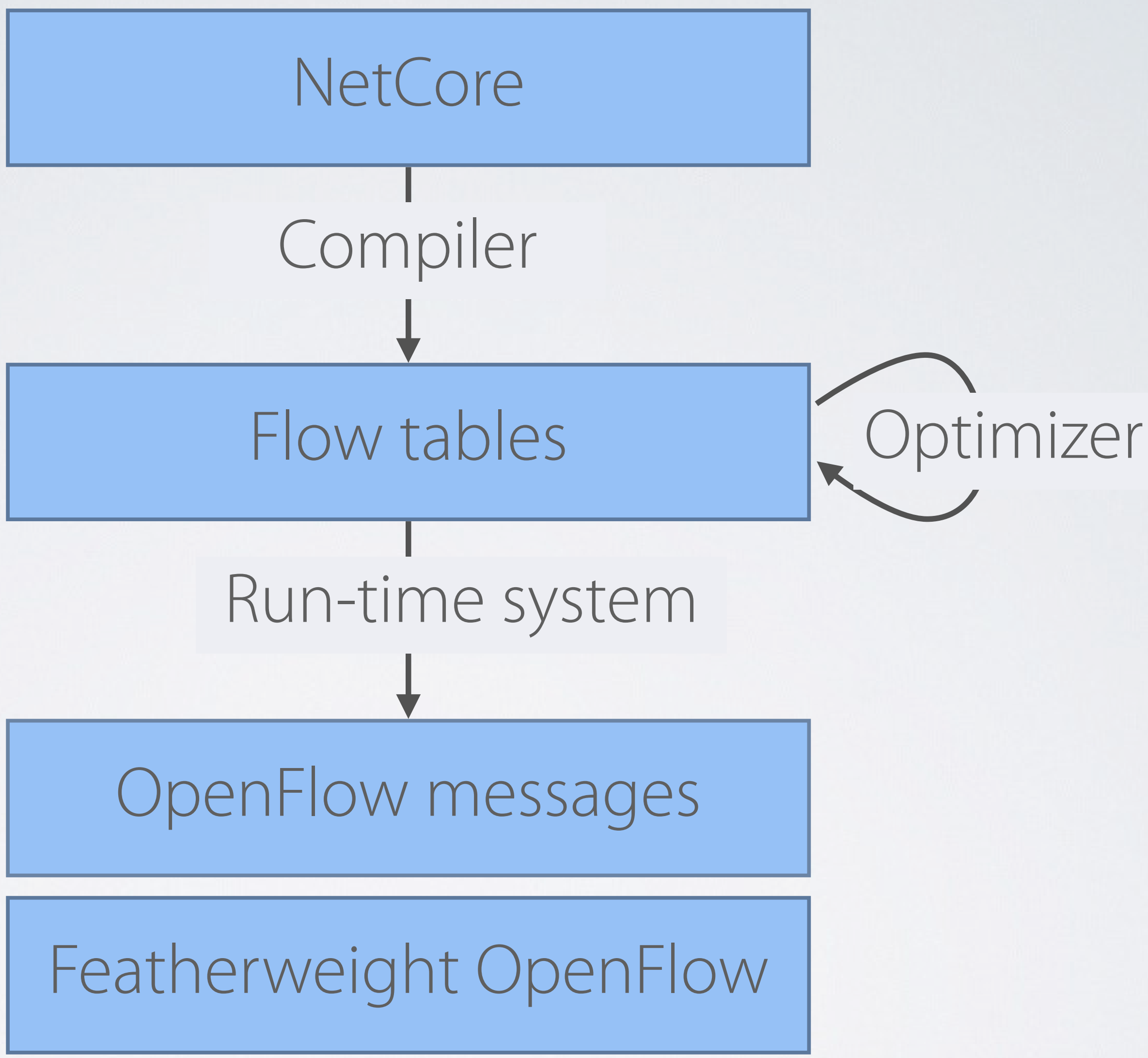
What happens if the controller sends the following pair of OpenFlow messages to a switch in sequence?

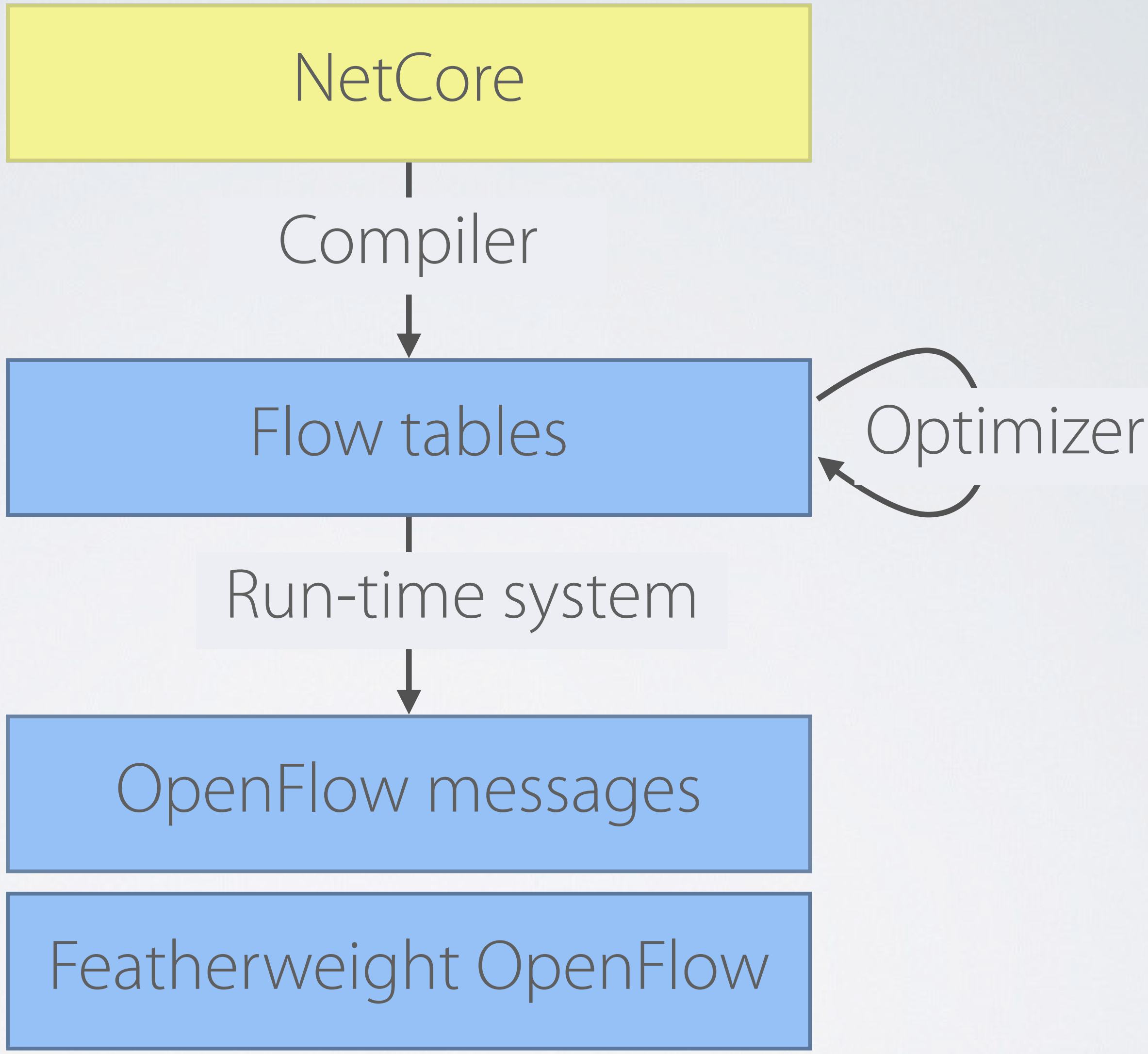
```
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                            srcIPAddress =
“10.0.1.99”, ... },
                  priority = 1,
                  actions = [ ] }
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                            srcIPAddress = “10.0.1.*”, ... },
                  priority = 2,
                  actions = [ flood ] }
```

The intention is to encode a negation...

...but the switch may process these in either order!

MACHINE-VERIFIED CONTROLLERS


NetCore
Compiler
Flow tables
Optimizer
Run-time system
OpenFlow messages
Featherweight OpenFlow


NetCore
Compiler
Flow tables
Optimizer
Run-time system
OpenFlow messages
Featherweight OpenFlow

NetCore

[POPL '12]

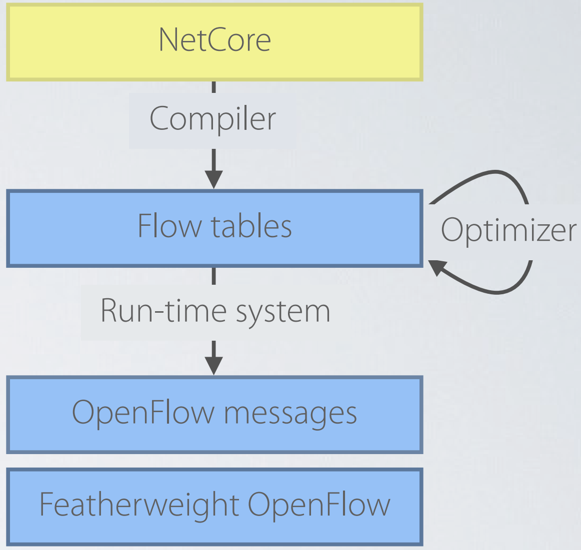
Syntax

```
Inductive pred : Type :=                  (* Predicates *)
  | OnSwitch : Switch -> pred
  | IngressPort : Port -> pred
  | DlSrc : EthernetAddress -> pred
  | DlDst : EthernetAddress -> pred
  | DlVlan : option VLAN -> pred
  | ...
  | And : pred -> pred -> pred
  | Or : pred -> pred -> pred
  | Not : pred -> pred
  | All : pred
  | None : pred.

Inductive PseudoPort : Type :=            (* Psuedo Ports *)
  | PhysicalPort : Port -> PseudoPort
  | AllPorts : PseudoPort.

Inductive act : Type :=                   (* Actions *)
  | FwdMod : Mod -> PseudoPort -> act

Inductive pol : Type :=                   (* Policies *)
  | Policy : pred -> list act -> pol
  | Union : pol -> pol -> pol
  | Restrict : pol -> pred -> pol.
```

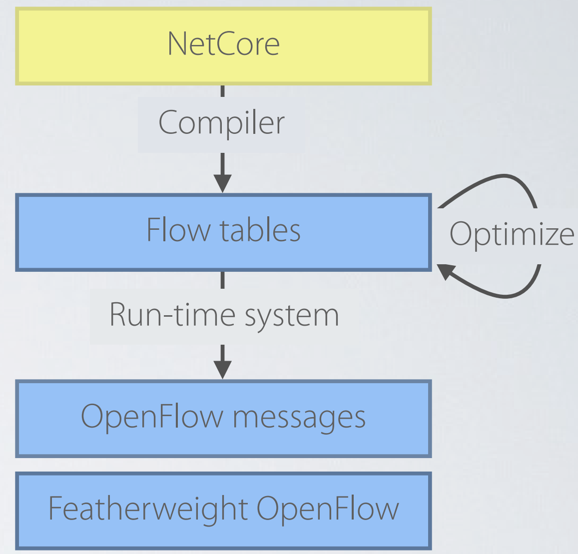


NetCore

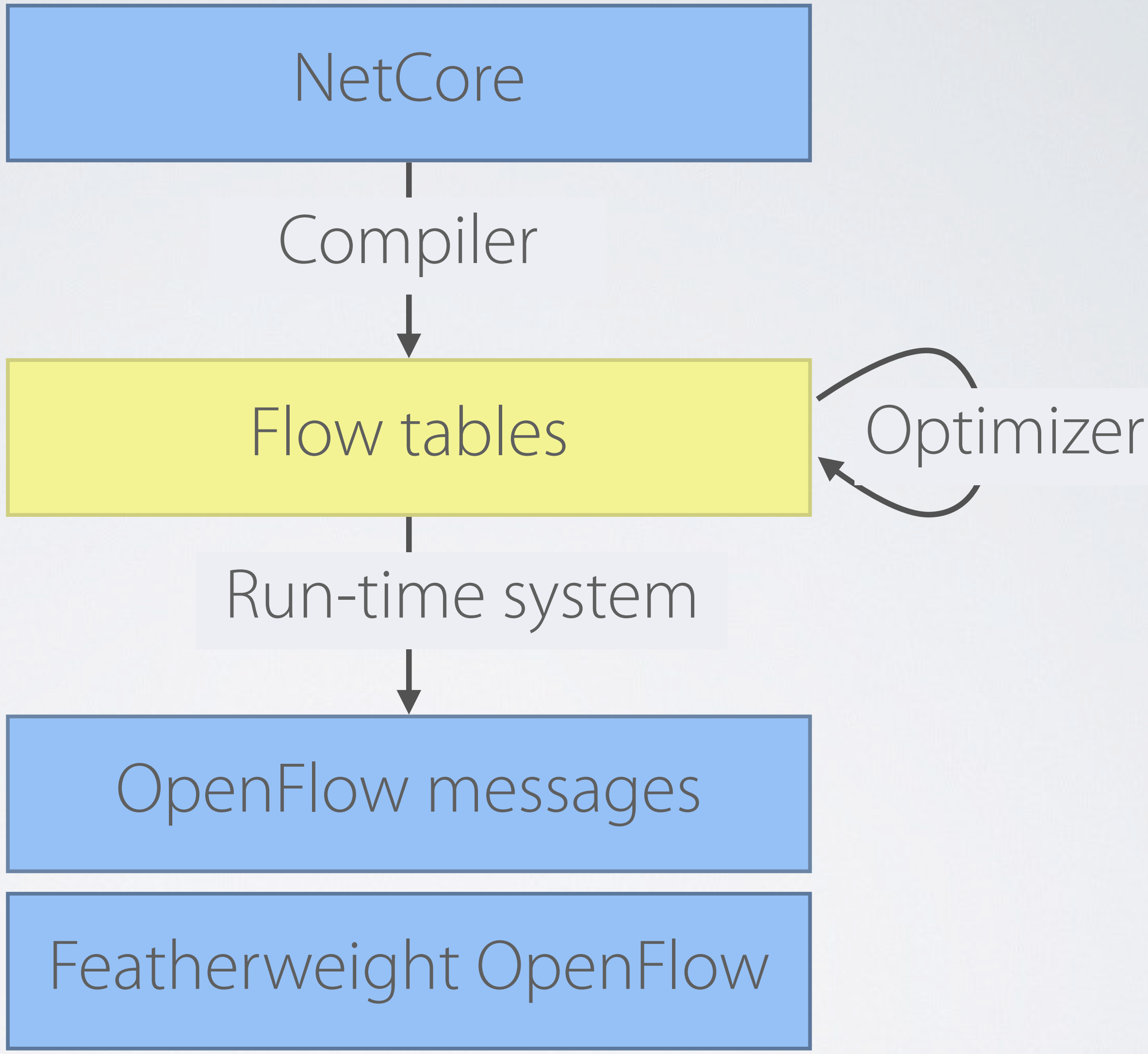
[POPL '12]

Semantics

$$\dfrac{\begin{array}{c} lp = (sw, pt, pk) \\ lps_{out} = pol(sw, pt, pk) \\ S = \{\!|(T(sw, pt_{out}), pk) \mid (pt_{out}, pk) \in lps_{out}|\!\} \end{array}}{\{\!|lp|\!\} \uplus \{\!|lp_1 \cdots lp_n|\!\} \xrightarrow{lp} S \uplus \{\!|lp_1 \cdots lp_n|\!\}}$$



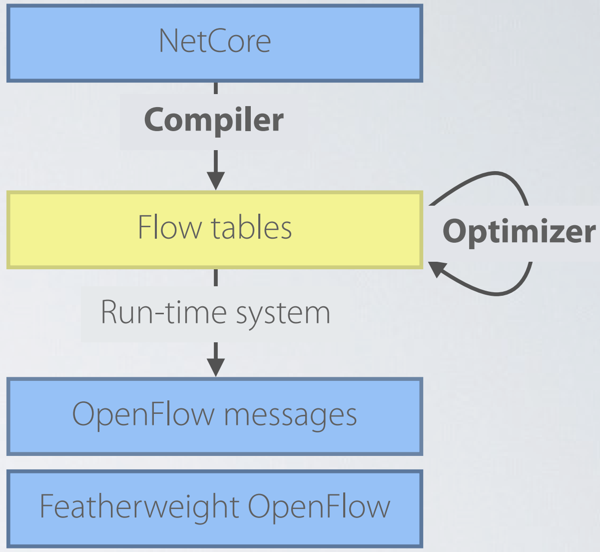
- Models hop-by-hop forwarding behavior of the network
- Abstracts away from the underlying distributed system
- Makes it easy to reason about network-wide properties


NetCore
Compiler
Flow tables
Optimizer
Run-time system
OpenFlow messages
Featherweight OpenFlow

NetCore to Flow Tables

Example

Priority	Pattern	Action
65534	**inPort** = 2, **dlSrc** = `dc:ba:65:43:21`	**Fwd** 2
65533	**inPort** = 2	**Fwd** 3



NetCore compiler

- Key operation: flow table intersection
- Must restrict to "valid" patterns

Optimizer

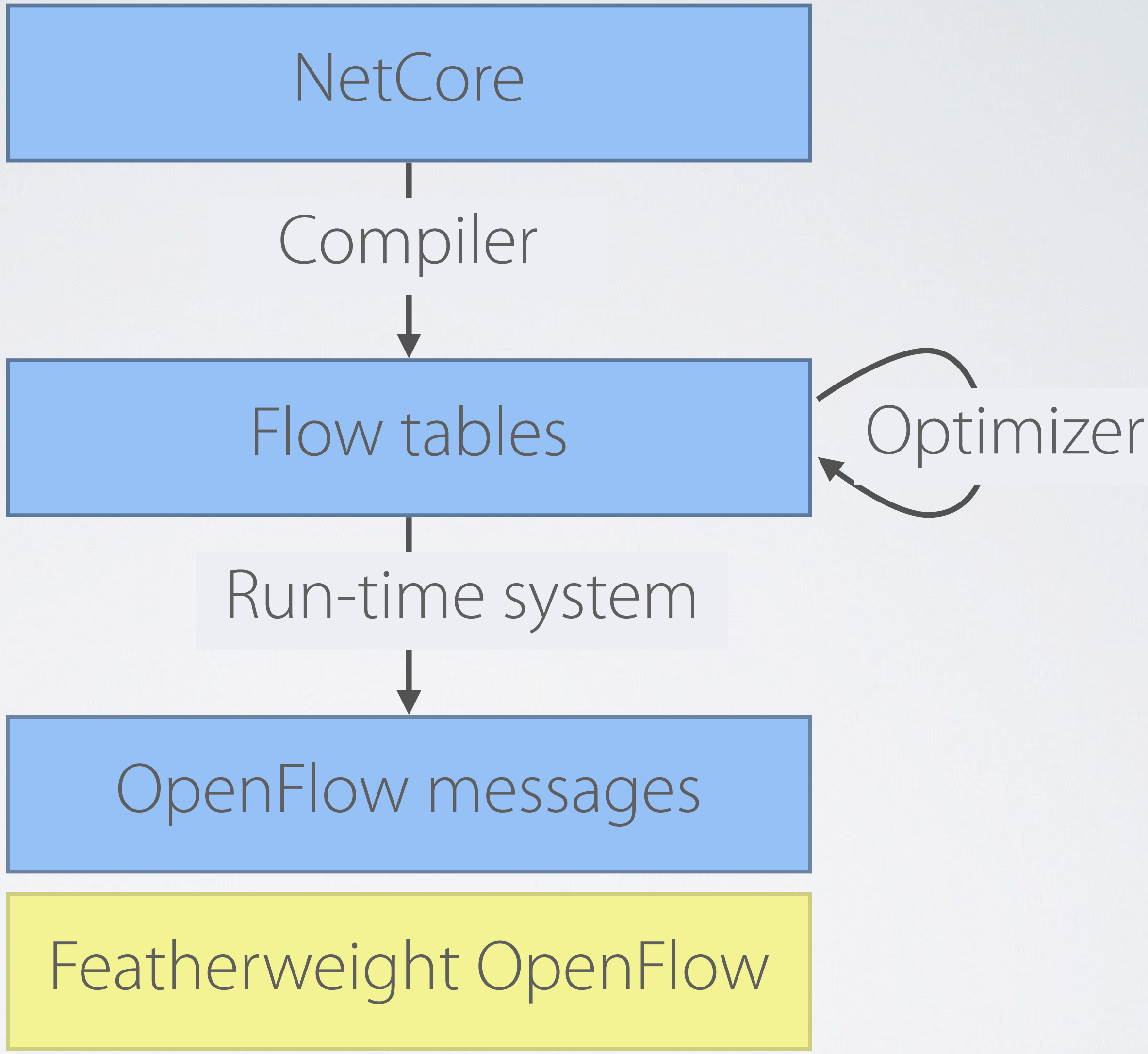
- Optimizer prunes (many) redundant rules
- Based on simple algebra of operations

Correctness Theorem

NetCore ~ FlowTable

Valid Patterns

```
Inductive ValidPattern : Pattern -> Prop :=
  | SupportedIPPatternValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos
                              tpSrc tpDst inPort nwProto,
      In nwProto SupportedL4Protos ->
      ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x800)
                              dlVlan dlVlanPcp
                              nwSrc nwDst (WildcardExact nwProto)
                              nwTos tpSrc tpDst inPort)
  | UnsupportedIPPatternValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos
                                inPort nwProto,
      ~ In nwProto SupportedL4Protos ->
      ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x800)
                              dlVlan dlVlanPcp
                              nwSrc nwDst (WildcardExact nwProto)
                              nwTos WildcardAll WildcardAll inPort)
  | ARPPacketValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst inPort,
      ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x806)
                              dlVlan dlVlanPcp
                              nwSrc nwDst WildcardAll
                              WildcardAll WildcardAll WildcardAll inPort)
  | UnknownDlTypPatternValid : forall dlSrc dlDst dlTyp dlVlan dlVlanPcp inPort,
      ValidPattern (MkPattern dlSrc dlDst dlTyp
                              dlVlan dlVlanPcp
                              WildcardAll WildcardAll WildcardAll
                              WildcardAll WildcardAll WildcardAll inPort)
  | EmptyPatternValid :
      ValidPattern Pattern_empty.
```


NetCore
Compiler
Flow tables
Optimizer
Run-time system
OpenFlow messages
Featherweight OpenFlow

OpenFlow Specification

42 pages...

...of informal English text

...and C struct definitions

Featherweight OpenFlow

Syntax

Devices	Switch	S	$::= \mathbb{S}(sw, pts, RT, inp.outp, inm, out$
	Controller	C	$::= \mathbb{C}(\sigma, f_{in}, f_{out})$
	Link	L	$::= \mathbb{L}(loc_{src}, pks, loc_{dst})$
	OpenFlow Link to Controller	M	$::= \mathbb{M}(sw, SMS, CMS)$
Packets and Locations	Packet	pk	$::= abstract$
	Switch ID	sw	$\in \mathbb{N}$
	Port ID	pt	$\in \mathbb{N}$
	Location	loc	$\in sw \times pt$
	Located Packet	lp	$\in loc \times pk$
Controller Components	Controller state	σ	$::= abstract$
	Controller input relation	f_{in}	$\in sw \times CM \times \sigma \rightsquigarrow \sigma$
	Controller output relation	f_{out}	$\in \sigma \rightsquigarrow sw \times SM \times \sigma$
Switch Components	Rule table	RT	$::= abstract$
	Rule table Interpretation	$[\![RT]\!]$	$\in lp \rightarrow \{\!\mid lp_1 \cdots lp_n \mid\!\} \times \{\!\mid CM_1 \cdots C$
	Rule table modifier	ΔRT	$::= abstract$
	Rule table modifier interpretation	apply	$\in \Delta RT \rightarrow RT \rightarrow \Delta RT$
	Ports on switch	pts	$\in \{pt_1 \cdots pt_n\}$
	Consumed packets	inp	$\in \{\!\mid lp_1 \cdots lp_n \mid\!\}$
	Produced packets	$outp$	$\in \{\!\mid lp_1 \cdots lp_n \mid\!\}$
	Messages from controller	inm	$\in \{\!\mid SM_1 \cdots SM_n \mid\!\}$
	Messages to controller	$outm$	$\in \{\!\mid CM_1 \cdots CM_n \mid\!\}$
Link Components	Endpoints	loc_{src}, loc_{dst}	$\in loc$ where $loc_{src} \neq loc_{dst}$
	Packets from loc_{src} to loc_{dst}	pks	$\in [pk_1 \cdots pk_n]$
Controller Link	Message queue from controller	SMS	$\in [SM_1 \cdots SM_n]$
	Message queue to controller	CMS	$\in [CM_1 \cdots CM_n]$
Abstract OpenFlow Protocol	Message from controller	SM	$::= \textbf{FlowMod}\ \Delta RT \mid \textbf{PktOut}\ pt\ p$
	Message to controller	CM	$::= \textbf{PktIn}\ pt\ pk \mid \textbf{BarrierReply}\ n$

Semantics

$$\frac{(outp', outm') = [\![RT]\!](lp)}{\mathbb{S}(sw, pts, RT, \{\!\mid lp \mid\!\} \uplus inp, outp, inm, outm) \xrightarrow{lp} \mathbb{S}(sw, pts, RT, inp, outp' \uplus outp, inm, outm' \uplus outm)}\ (\textsc{Pkt-Process})$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, \{\!\mid (sw, pt, pk) \mid\!\} \uplus outp, inm, outm) \mid \mathbb{L}((sw, pt), pks, loc') \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{L}((sw, pt), [pk] +\!\!+ pks, loc') \end{array}}\ (\textsc{Send-Wire})$$

$$\frac{}{\begin{array}{ll} & \mathbb{L}(loc, pks +\!\!+ [pk], (sw, pt)) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \xrightarrow{(sw,pt,pk)} & \mathbb{L}(loc, pks, (sw, pt)) \mid \mathbb{S}(sw, pts, RT, \{\!\mid (sw, pt, pk) \mid\!\} \uplus inp, outp, inm, outm) \end{array}}\ (\textsc{Recv-Wire})$$

$$\frac{RT' = \text{apply}(\Delta RT, RT)}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\mid \textbf{FlowMod}\ \Delta RT \mid\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT', inp, outp, inm, outm)}\ (\textsc{Switch-FlowMod})$$

$$\frac{pt \in pts}{\mathbb{S}(sw, pts, RT, inp, outp, \{\!\mid \textbf{PktOut}\ pt\ pk \mid\!\} \uplus inm, outm) \longrightarrow \mathbb{S}(sw, pts, RT, inp, \{\!\mid (sw, pt, pk) \mid\!\} \uplus outp, inm, outm)}\ (\textsc{Switch-PktOut})$$

$$\frac{f_{out}(\sigma) \rightsquigarrow (sw, SM, \sigma')}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, [SM] +\!\!+ SMS, CMS)}\ (\textsc{Ctrl-Send})$$

$$\frac{f_{in}(sw, \sigma, CM) \rightsquigarrow \sigma'}{\mathbb{C}(\sigma, f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS +\!\!+ [CM]) \longrightarrow \mathbb{C}(\sigma', f_{in}, f_{out}) \mid \mathbb{M}(sw, SMS, CMS)}\ (\textsc{Ctrl-Recv})$$

$$\frac{SM \neq \textbf{BarrierRequest}\ n}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [SM], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \{\!\mid SM \mid\!\} \uplus inm, outm) \end{array}}\ (\textsc{Switch-Recv-Ctrl})$$

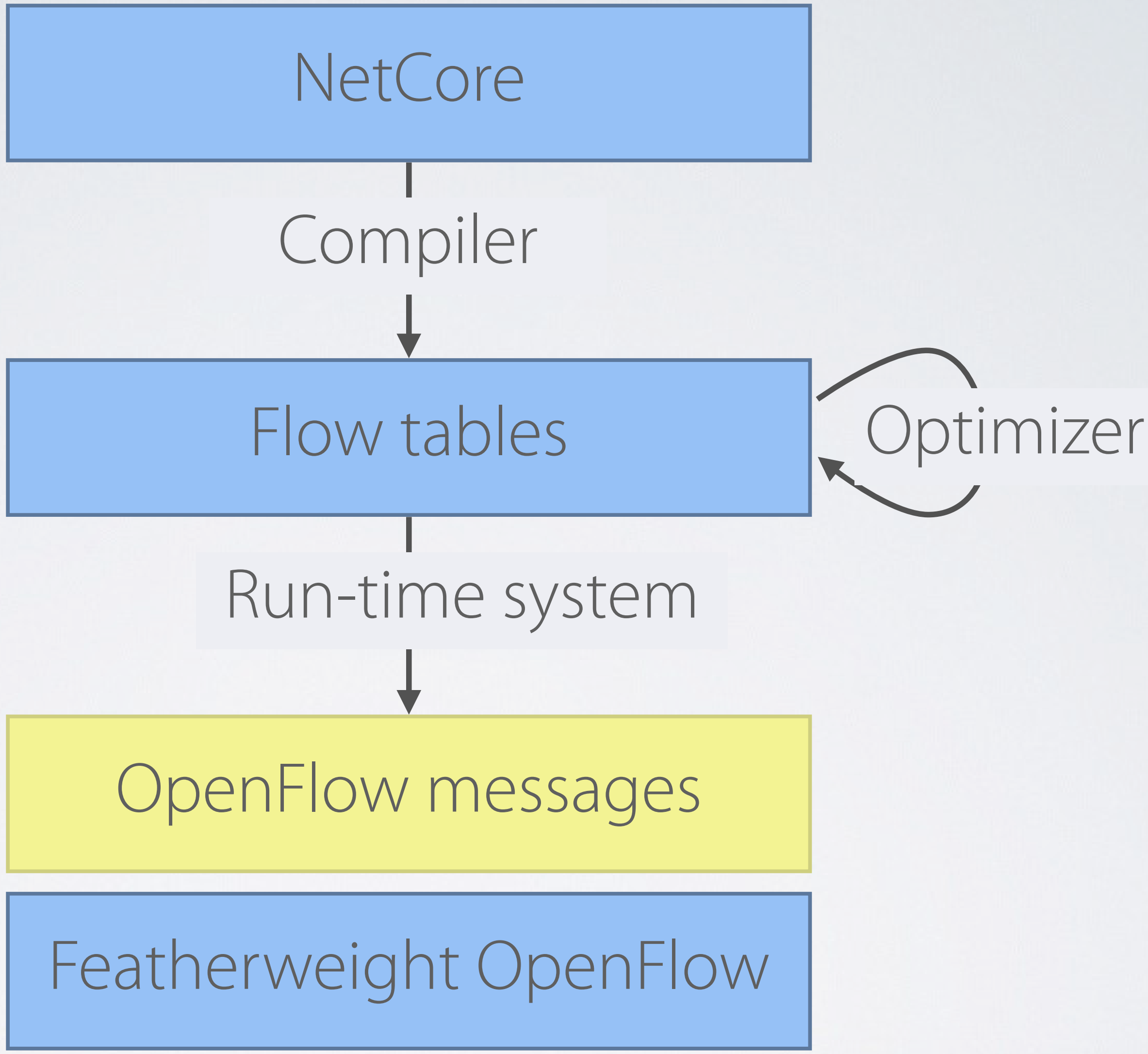
$$\frac{}{\begin{array}{ll} & \mathbb{M}(sw, SMS +\!\!+ [\textbf{BarrierRequest}\ n], CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, outm) \\ \longrightarrow & \mathbb{M}(sw, SMS, CMS) \mid \mathbb{S}(sw, pts, RT, inp, outp, \emptyset, \{\!\mid \textbf{BarrierReply}\ n \mid\!\} \uplus outm) \end{array}}\ (\textsc{Switch-Recv-Barrier})$$

$$\frac{}{\begin{array}{ll} & \mathbb{S}(sw, pts, RT, inp, outp, inm, \{\!\mid CM \mid\!\} \uplus outm) \mid \mathbb{M}(sw, SMS, CMS) \\ \longrightarrow & \mathbb{S}(sw, pts, RT, inp, outp, inm, outm) \mid \mathbb{M}(sw, SMS, [CM] +\!\!+ CMS) \end{array}}\ (\textsc{Switch-Send-Ctrl})$$

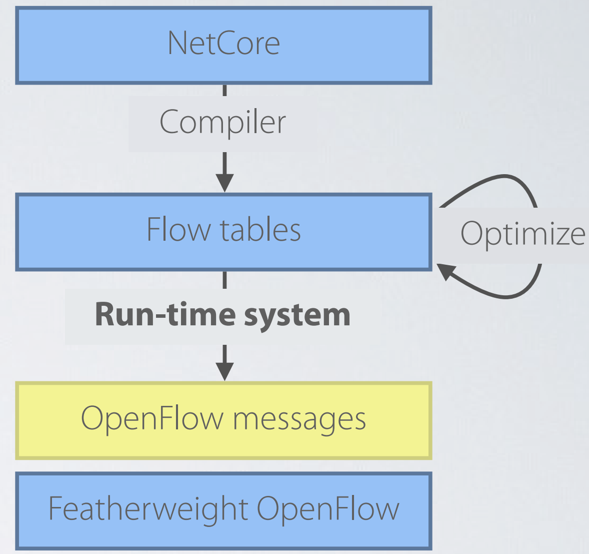
Key judgments:

- Controller in: $(sw, CM, \sigma) \rightsquigarrow \sigma'$
- Controller out: $\sigma \rightsquigarrow (sw, SM, \sigma')$
- Network step: $M \rightarrow M'$

Models *all* essential asynchrony


NetCore
Compiler
Flow tables
Optimizer
Run-time system
OpenFlow messages
Featherweight OpenFlow

Run-Time System



Invariants

- Maintain a sound approximation of overall flow table each switch
- Eventually process all diverted packets

Theorem

FlowTable ≈ Featherweight OpenFlow

Run-time instances

- Trivial: processes all packets on controller
- Proactive: installs rules, falls back to Trivial when out of space
- Full: like Proactive, but also installs exact-match rules

Safe Wires

```
Inductive SafeWire : SF -> SF -> SF -> list CM -> Prop :=
| SafeWire_nil : forall lb ub,
    extends ub lb ->
    SafeWire lb ub lb nil
| SafeWire_cons_FlowMod : forall lb ub sf sft lst,
    SafeWire lb ub sf lst ->
    extends ub (apply_SFT sft sf) ->
    SafeWire lb ub (apply_SFT sft sf) (FlowMod sft :: lst)
| SafeWire_cons_PktOut : forall lb ub sf pt pk lst,
    SafeWire lb ub sf lst ->
    SafeWire lb ub sf (PktOut pt pk :: lst)
| SafeWire_cons_BarrierRequest : forall lb ub sf n lst,
    SafeWire lb ub sf lst ->
    SafeWire lb ub sf (BarrierRequest n :: lst).
```

Implementation

Source

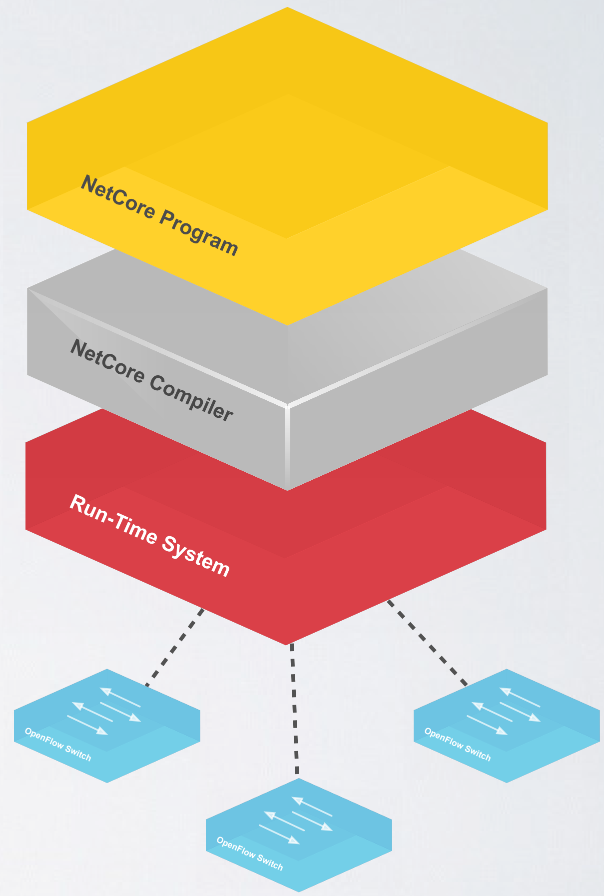
- ~8,000 lines of Coq
- ~1,500 lines of Haskell

Components

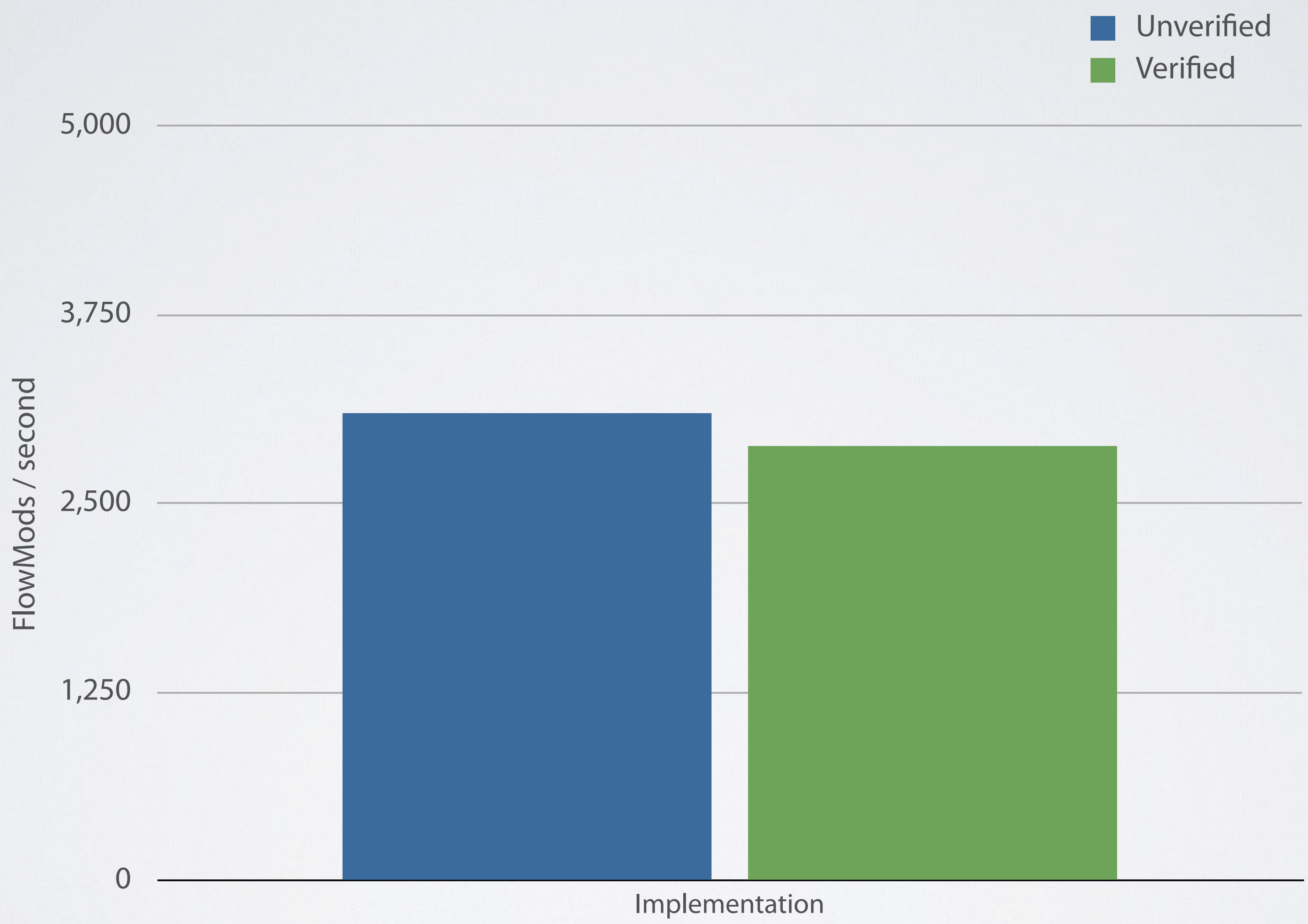
- NetCore compiler and optimizer
- Flow tables
- Featherweight OpenFlow
- Run-time system instances
- Proofs of correctness

Status

- Extracts to Haskell source code
- Compiles against Nettle libraries
- Running on "production" traffic in the lab



Performance



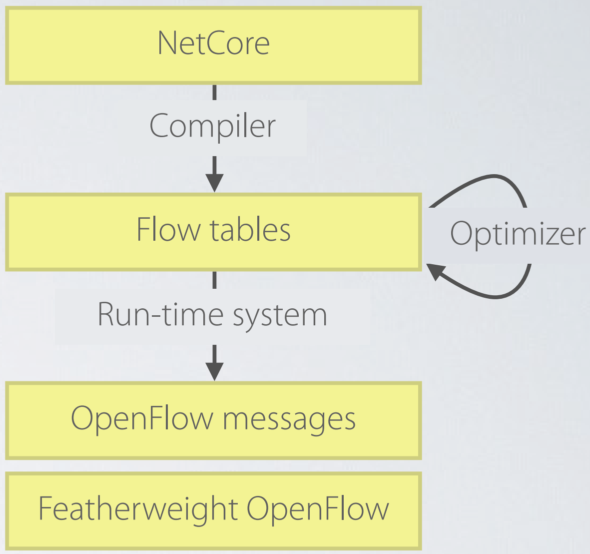
Conclusion

Networks are critical infrastructure...

...developed using 1970s-era techniques

Software-defined networks are an architecture that could be used to put networks on a solid foundation

Machine-verified controllers based on NetCore a first step in this direction



A Grand Collaboration: Languages + Networking

Frenetic Cornell

Shrutarshi Basu (PhD)
Nate Foster (Faculty)
Arjun Guha (Postdoc)
Stephen Gutz (Undergrad)
Mark Reitblatt (PhD)
Robert Soulé (Postdoc)
Alec Story (Undergrad)

Frenetic Princeton

Chris Monsanto (PhD)
Joshua Reich (Postdoc)
Jen Rexford (Faculty)
Cole Schlesinger (PhD)
Dave Walker (Faculty)
Naga Praveen Katta (PhD)

frenetic >>

`http://frenetic-lang.org`