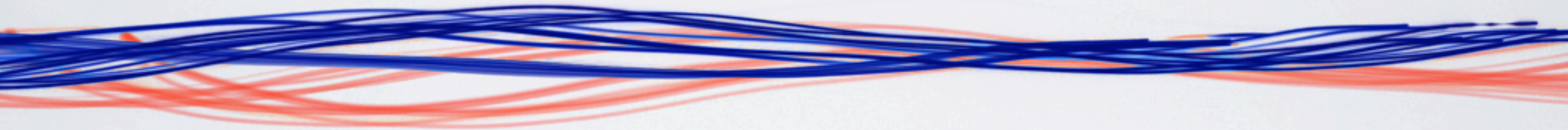
Consistent Updates in Software-Defined Networks



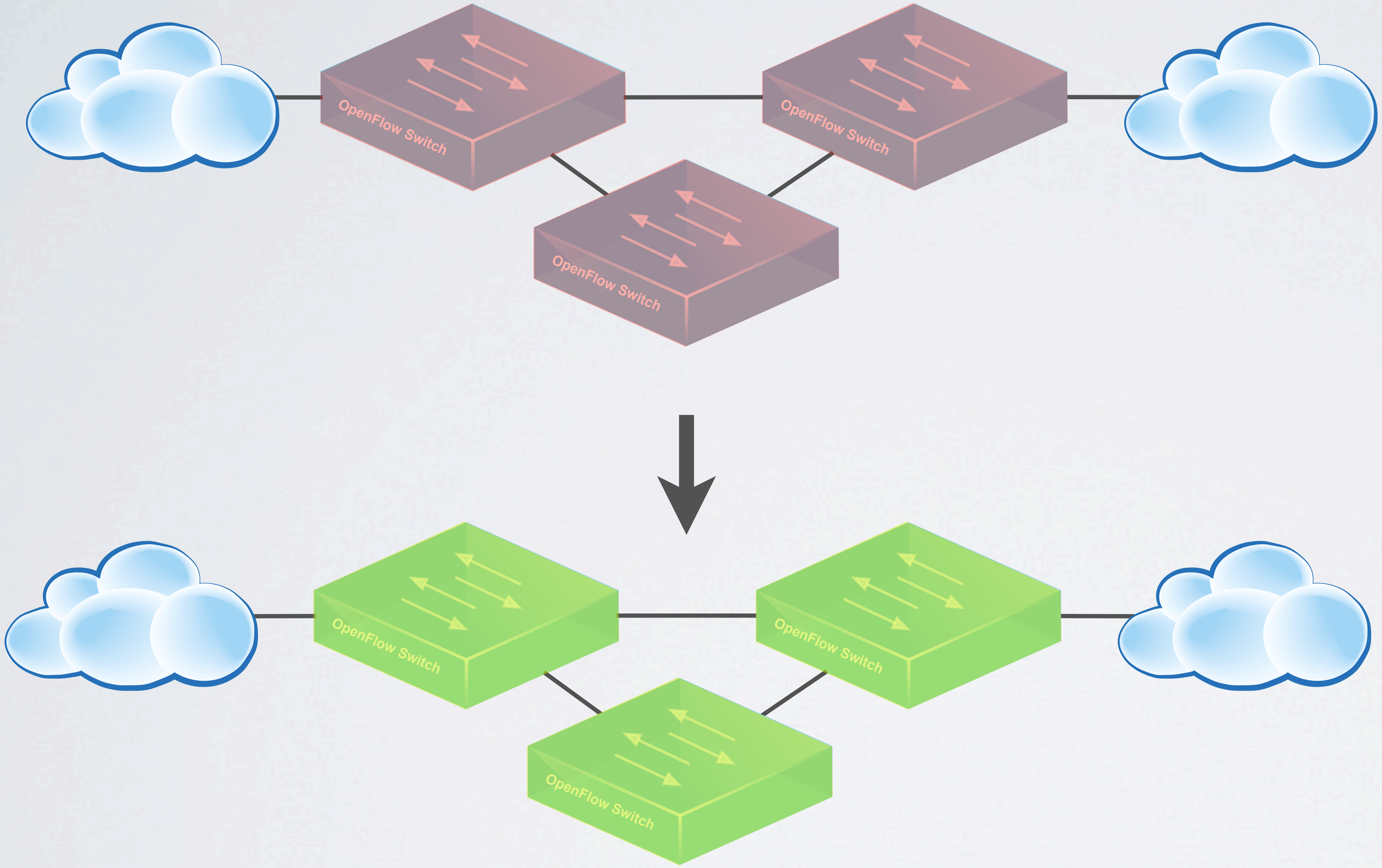
CORNELL UNIVERSITY FOUNDED A.D. 1865

Nate Foster
Mark Reitblatt

Cole Schlesinger
Jennifer Rexford
David Walker

frenetic >>

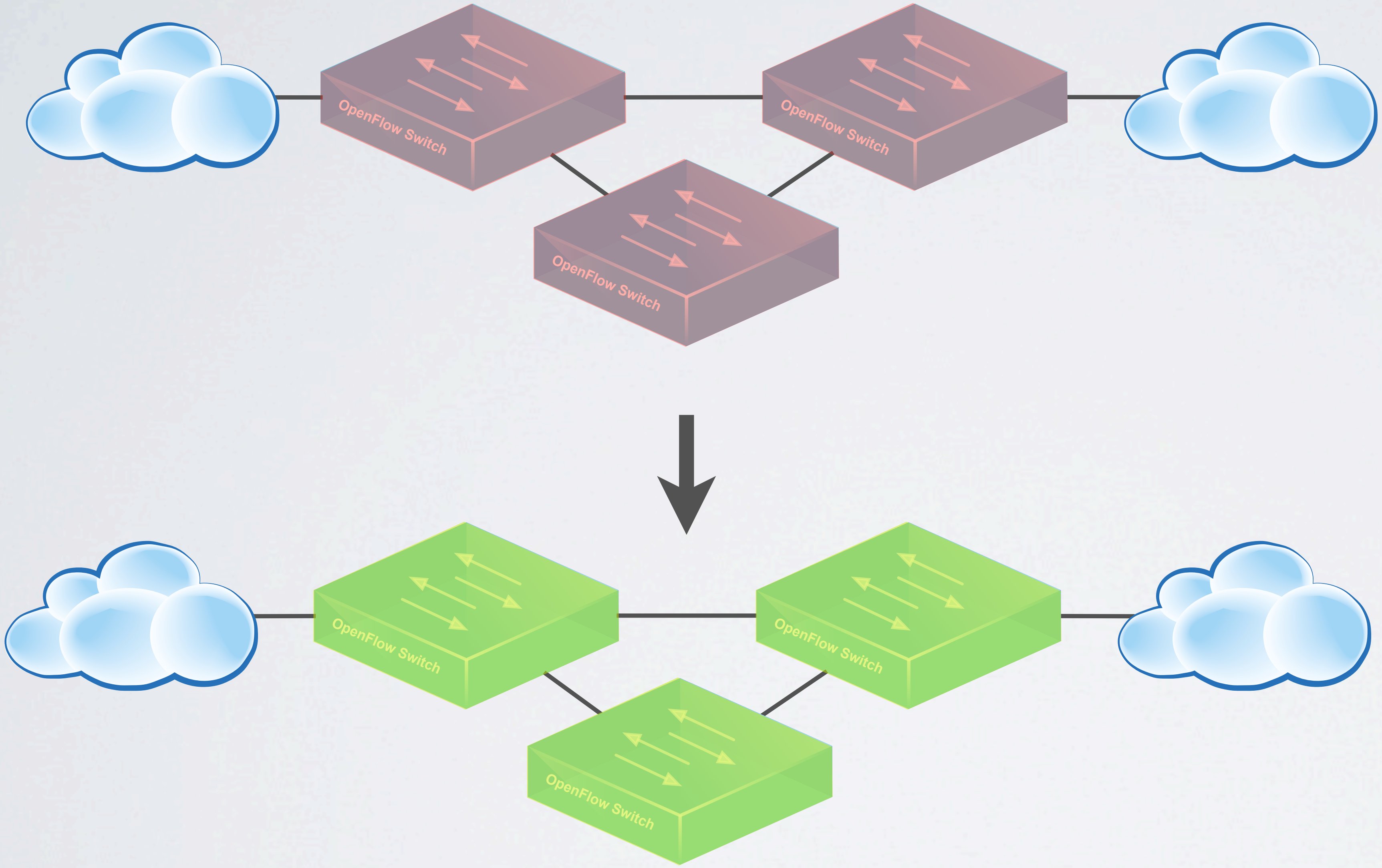
Network Updates



Network Updates

- Routine maintenance
- Unexpected failures
- Traffic engineering
- Updated ACL

Network Updates



Network Updates
- Routine maintenance
- Unexpected failures
- Traffic engineering
- Updated ACL

Desired Invariants
- No lost packets
- No broken connections
- No forwarding loops
- No security holes

amazon webservices™

At 12:47 AM PDT on April 21st, a network change was performed as part of our normal scaling activities...

During the change, one of the steps is to shift traffic off of one of the redundant routers...

The traffic shift was executed incorrectly and the traffic was routed onto the lower capacity redundant network.

This led to a “re-mirroring storm”...

During this re-mirroring storm, the volume of connection attempts was extremely high and nodes began to fail, resulting in more volumes left needing to re-mirror. This added more requests to the re-mirroring storm...

The trigger for this event was a **network configuration change**.

At 12:47 AM PDT on April 21st, a network change was performed as part of our normal scaling activities...

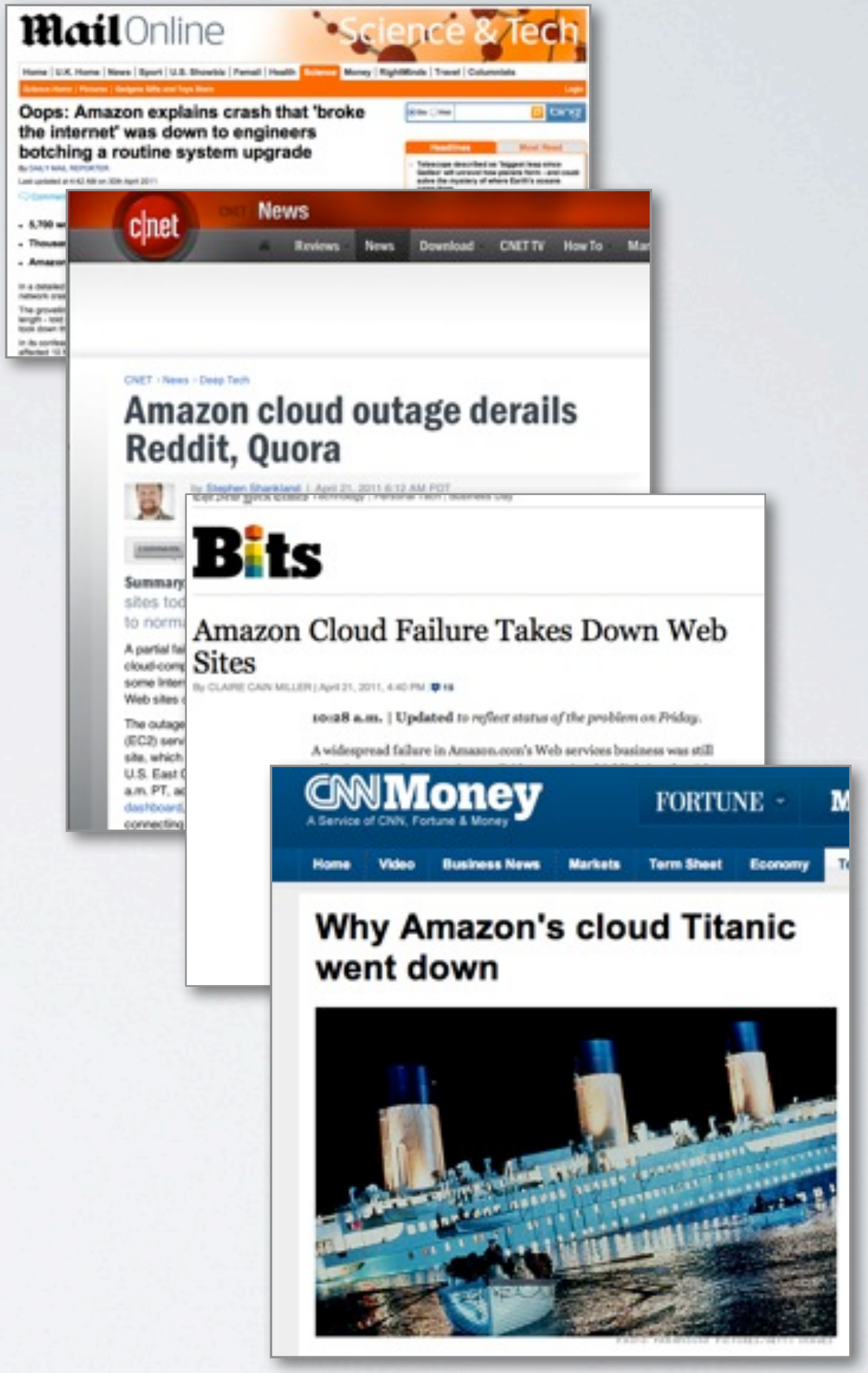
During the change, one of the steps is to shift traffic off of one of the redundant routers...

The traffic shift was executed incorrectly and the traffic was routed onto the lower capacity redundant network.

This led to a “re-mirroring storm”...

During this re-mirroring storm, the volume of connection attempts was extremely high and nodes began to fail, resulting in more volumes left needing to re-mirror. This added more requests to the re-mirroring storm...

The trigger for this event was a **network configuration change**.



Oops: Amazon explains crash that 'broke the internet' was down to engineers botching a routine system upgrade

CNET › News › Deep Tech

Amazon cloud outage derails Reddit, Quora

by Stephen Shankland | April 21, 2011 6:12 AM PDT

Amazon Cloud Failure Takes Down Web Sites

By CLAIRE CAIN MILLER | April 21, 2011, 4:40 PM

10:28 a.m. | Updated to reflect status of the problem on Friday.

A widespread failure in Amazon.com's Web services business was still

Why Amazon's cloud Titanic went down

Prior Work

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John[*] *Ethan Katz-Bassett*[*] *Arvind Krishnamurthy*[*] *Thomas Anderson*[*]
Arun Venkataramani[†]

Abstract

Internet routing protocols (BGP, OSPF, RIP) have traditionally favored responsiveness over consistency. A router applies a received update immediately to its forwarding table before propagating the update to other routers, including those that potentially depend upon the outcome of the update. Responsiveness comes at the cost of routing loops and blackholes—a router A thinks its route to a destination is via B but B disagrees. By favoring responsiveness (a liveness property) over consistency (a safety property), Internet routing has lost both.

Our position is that consistent state in a distributed system makes its behavior more predictable and securable. To this end, we present consensus routing, a consistency-first approach that cleanly separates safety and liveness using two logically distinct modes of packet delivery: a *stable* mode where a route is adopted only after all dependent routers have agreed upon it, and a *transient* mode that heuristically forwards the small fraction of packets that encounter failed links. Somewhat surprisingly, we find that consensus routing improves overall availability when used in conjunction with existing transient mode heuristics such as backup paths, deflections, or detouring. Experiments on the Internet's AS-level topology show that consensus routing eliminates nearly all transient disconnectivity in BGP.

1 Introduction

Internet routing, especially interdomain routing, has traditionally favored responsiveness, i.e., how quickly the network reacts to changes, over consistency, i.e., ensuring that packets traverse adopted routes. A router applies a received update immediately to its forwarding table before propagating the update to other routers, including those that potentially depend upon the outcome of the update. Responsiveness comes at the cost of availability: a router A thinks its route to a destination is via B but B disagrees, either because 1) B's old route to the destination is via A, causing loops, or 2) B does not have a current route to the destination, causing blackholes. BGP updates are known to cause up to 30% packet-loss for two minutes or more after a routing change, even though usable physical routes exist [26]. Further, transient loops account for 90% of all packet loss according to a Sprint network study [16]. Even a recovering link can cause unavailability lasting tens of seconds due to an inconsistent view among routers in a single autonomous system [44].

Our position is that the lack of consistency is at the root of bigger problems in Internet routing beyond availability. First, protocol behavior is complex and unpredictable as routers by design operate upon inconsistent distributed state, e.g., by forwarding packets along loops. There is no indicator of when, if at all, the network converges to a consistent state. Second, unpredictable behavior makes the system more vulnerable to misconfiguration or abuse, as it is difficult to distinguish between expected behavior and misbehavior. Third, unpredictable behavior stifles innovation in the long term, e.g., network operators are reluctant to adopt protocol optimizations such as interdomain traffic engineering [1] because they have to worry about its poorly understood side-effects. Perhaps most tellingly, despite a decade of research investigating the complex dynamics of interdomain routing, the *goal of a simple, practical routing protocol that allows general routing policies and achieves high availability* has remained elusive.

Our primary contribution, consensus routing, achieves the above goal. The key insight is to recognize consistency as a safety property and responsiveness as a liveness property and systematically separate the two design concerns, thereby borrowing an old lesson from distributed system design. Consistency safety means that a router forwards a packet strictly along the path adopted by the upstream routers unless the packet encounters a failed link. Liveness means that the system reacts quickly to failures or policy changes. Separating safety and liveness improves end-to-end availability, and, perhaps more importantly, makes system behavior simple to describe and understand.

Consensus routing achieves this separation using two logically distinct modes of packet delivery: 1) A *stable mode* ensures that a route is adopted only after all dependent routers have agreed upon a consistent view of global state. Every epoch, routers participate in a distributed snapshot and consensus protocol to determine whether or not updates are *complete*, i.e., they have been processed by every router that depends on the update. The output of the consensus serves as an explicit indicator that routers may adopt a consistent set of routes processed before the snapshot. 2) A *transient mode* ensures high availability when a packet encounters a router that does not possess a stable route, either because the corresponding link failed

---

[*] Dept. of Computer Science, Univ. of Washington, Seattle.
[†] University of Massachusetts Amherst.

Prior Work

Consensus Routing: The Internet as a Distributed System

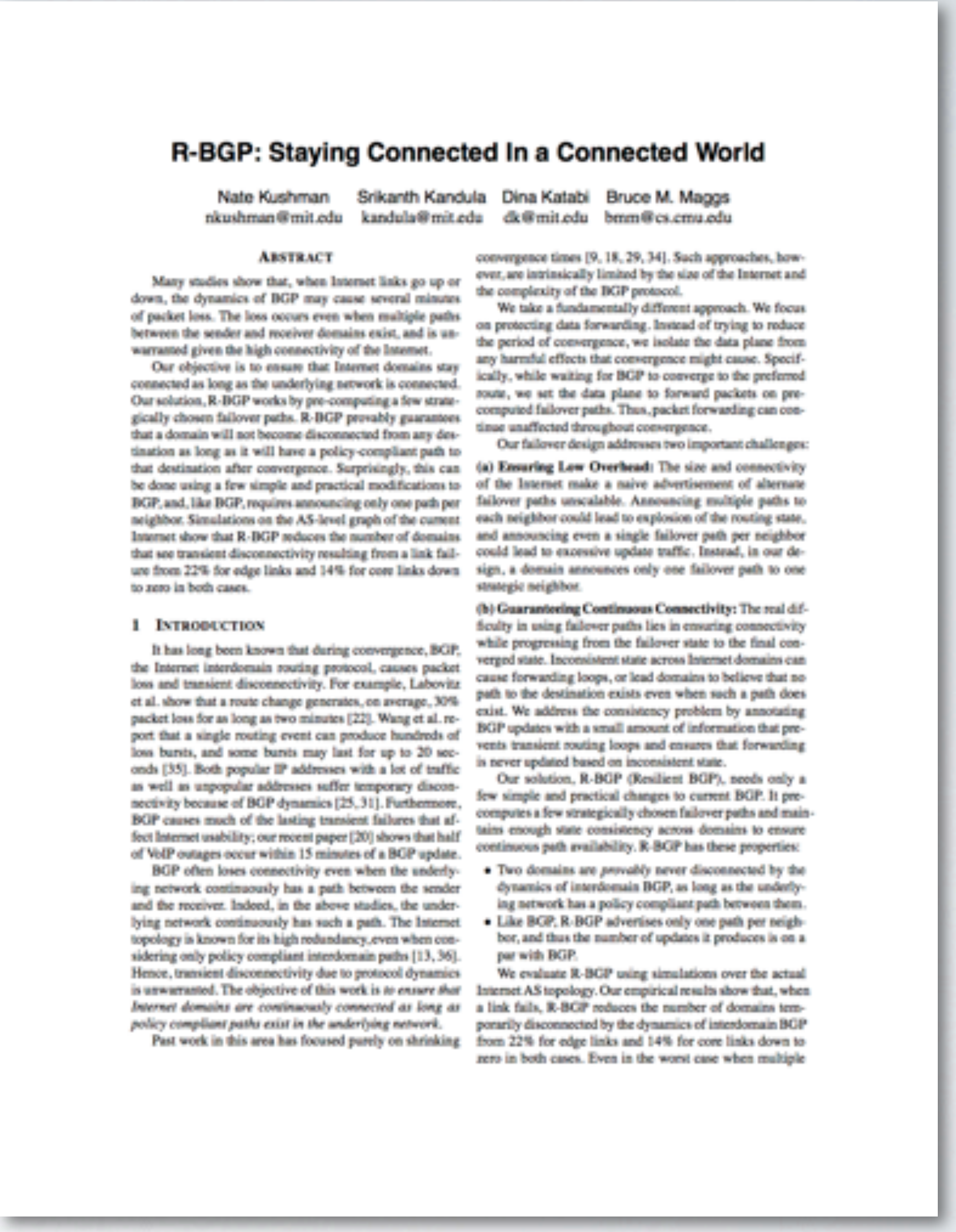
John P. John   Ethan Katz-Bassett   Arvind Krishnamurthy   Thomas Anderson
Arun Venkataramani

Abstract

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John[1] Ethan Katz-Bassett[1] Arvind Krishnamurthy[1] Thomas Anderson[1]
Arun Venkataramani[2]



R-BGP: Staying Connected In a Connected World

Nate Kushman nkushman@mit.edu
Srikanth Kandula kandula@mit.edu
Dina Katabi dk@mit.edu
Bruce M. Maggs bmm@cs.cmu.edu

ABSTRACT

Many studies show that, when Internet links go up or down, the dynamics of BGP may cause several minutes of packet loss. The loss occurs even when multiple paths between the sender and receiver domains exist, and is unwarranted given the high connectivity of the Internet.

Our objective is to ensure that Internet domains stay connected as long as the underlying network is connected. Our solution, R-BGP works by pre-computing a few strategically chosen failover paths. R-BGP provably guarantees that a domain will not become disconnected from any destination as long as it will have a policy-compliant path to that destination after convergence. Surprisingly, this can be done using a few simple and practical modifications to BGP, and, like BGP, requires announcing only one path per neighbor. Simulations on the AS-level graph of the current Internet show that R-BGP reduces the number of domains that see transient disconnectivity resulting from a link failure from 22% for edge links and 14% for core links down to zero in both cases.

1 INTRODUCTION

It has long been known that during convergence, BGP, the Internet interdomain routing protocol, causes packet loss and transient disconnectivity. For example, Labovitz et al. show that a route change generates, on average, 30% packet loss for as long as two minutes [22]. Wang et al. report that a single routing event can produce hundreds of loss bursts, and some bursts may last for up to 20 seconds [35]. Both popular IP addresses with a lot of traffic as well as unpopular addresses suffer temporary disconnectivity because of BGP dynamics [25, 31]. Furthermore, BGP causes much of the lasting transient failures that affect Internet usability; our recent paper [20] shows that half of VoIP outages occur within 15 minutes of a BGP update.

BGP often loses connectivity even when the underlying network continuously has a path between the sender and the receiver. Indeed, in the above studies, the underlying network continuously has such a path. The Internet topology is known for its high redundancy, even when considering only policy compliant interdomain paths [13, 36]. Hence, transient disconnectivity due to protocol dynamics is unwarranted. The objective of this work is *to ensure that Internet domains are continuously connected as long as policy compliant paths exist in the underlying network.*

Past work in this area has focused purely on shrinking convergence times [9, 18, 29, 34]. Such approaches, however, are intrinsically limited by the size of the Internet and the complexity of the BGP protocol.

We take a fundamentally different approach. We focus on protecting data forwarding. Instead of trying to reduce the period of convergence, we isolate the data plane from any harmful effects that convergence might cause. Specifically, while waiting for BGP to converge to the preferred route, we set the data plane to forward packets on pre-computed failover paths. Thus, packet forwarding can continue unaffected throughout convergence.

Our failover design addresses two important challenges:

(a) Ensuring Low Overhead: The size and connectivity of the Internet make a naive advertisement of alternate failover paths unscalable. Announcing multiple paths to each neighbor could lead to explosion of the routing state, and announcing even a single failover path per neighbor could lead to excessive update traffic. Instead, in our design, a domain announces only one failover path to one strategic neighbor.

(b) Guaranteeing Continuous Connectivity: The real difficulty in using failover paths lies in ensuring connectivity while progressing from the failover state to the final converged state. Inconsistent state across Internet domains can cause forwarding loops, or lead domains to believe that no path to the destination exists even when such a path does exist. We address the consistency problem by annotating BGP updates with a small amount of information that prevents transient routing loops and ensures that forwarding is never updated based on inconsistent state.

Our solution, R-BGP (Resilient BGP), needs only a few simple and practical changes to current BGP. It precomputes a few strategically chosen failover paths and maintains enough state consistency across domains to ensure continuous path availability. R-BGP has these properties:

- Two domains are *provably* never disconnected by the dynamics of interdomain BGP, as long as the underlying network has a policy compliant path between them.
- Like BGP, R-BGP advertises only one path per neighbor, and thus the number of updates it produces is on a par with BGP.

We evaluate R-BGP using simulations over the actual Internet AS topology. Our empirical results show that, when a link fails, R-BGP reduces the number of domains temporarily disconnected by the dynamics of interdomain BGP from 22% for edge links and 14% for core links down to zero in both cases. Even in the worst case when multiple

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John Ethan Katz-Bassett Arvind Krishnamurthy Thomas Anderson
Arun Venkataramani

R-BGP: Staying Connected In a Connected World

Nate Kushman Srikanth Kandula Dina Katabi Bruce M. Maggs

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, Arun Venkataramani

R-BGP: Staying Connected In a Connected World

Nate Kushman, Srikanth Kandula, Dina Katabi, Bruce M. Maggs

Graceful Network State Migrations

Saqib Raza, *Member, IEEE*, Yuanbo Zhu, and Chen-Nee Chuah, *Senior Member, IEEE*

Abstract—A significant fraction of network events (such as topology or route changes) and the resulting performance degradation stem from premeditated network management and operational tasks. This paper introduces a general class of *Graceful Network State Migration* (GNSM) problems, where the goal is to discover the optimal sequence of operations that progressively transition the network from its initial to a desired *final* state while minimizing the overall performance disruption. We investigate two specific GNSM problems: (a) Link Weight Reassignment Scheduling (LWRS) studies the optimal ordering of link weight updates to migrate from an existing to a new link weight assignment, and (b) Link Maintenance Scheduling (LMS) looks at how to schedule link deactivations and subsequent reactivations for maintenance purposes. LWRS and LMS are both combinatorial optimization problems. We use dynamic programming to find the optimal solutions when the problem size is small, and leverage Ants Colony Optimization to get near-optimal solutions for large problem sizes. Our simulation study reveals that judiciously ordering network operations can achieve significant performance gains. Our GNSM solution framework is generic and applies to similar problems with different operational contexts, underlying network protocols or mechanisms, and performance metrics.

I. INTRODUCTION

The Internet has been an enabling technology for mission-critical applications and services such as Voice over IP, VPNs, e-commerce applications, and multimedia streaming. Such applications rely upon consistent Quality of Service (QoS) provisioning by Internet Service Providers (ISPs), with five-nines availability (99.999% uptime) becoming the norm rather than the exception. The end-to-end perceived QoS can potentially be affected due to the dynamic nature of networks. For instance, network topology may change due to transient router/link outages or long-term network engineering. Furthermore, protocol configuration parameters may be altered to migrate from one setting to another. Ideally, QoS guarantees should persist across such dynamic conditions.

Some of these dynamic changes are *inadvertent* e.g., ones due to faulty interfaces, router crashes, and accidental fiber cuts. However, other changes ensue from deliberate and *premeditated* actions of network operators (e.g., routine maintenance). A failure characterization study of an IP backbone [18] observed that planned maintenance activities account for more than 20% of transient failures. Other studies [8] have also observed the prevalence of such planned maintenance activities. Premeditated network tasks also include network upgrade activities such as adding new routers or overhauling link capacity. Another example of a premeditated network task is migrating an existing OSPF [20] or IS-IS [25][1] link weight assignment to a new assignment that has been optimized based on the most up-to-date traffic matrix estimates.

In the case of premeditated tasks, network operators have the prerogative to decide the sequence of *atomic* operations that comprise such a task. This paper[2] introduces a general class of problems referred to as *Graceful Network State Migration* (GNSM) problems, which typically involve migrating a network from its *initial* state to a *final* state by executing a series of *atomic* operations. Each of these operations may cause some *performance disruption* that is a function of the network's changed state. The GNSM problem is to discover the sequence of operations that progressively transition the network to the final state while minimizing the overall disruption. This paper looks at two specific *GNSM* problems, as described below.

A. Link Weight Reassignment Scheduling

Setting link weights is the primary tool used by network operators to control network load distribution and traffic engineering [9–11,22]. Link weights are optimized based on an estimate of the traffic matrix. They are usually not modified in response to short-term fluctuations in the traffic matrix. However, the estimated traffic matrix may change significantly over a longer period of time, prompting network operators to re-optimize and reset link weights. In such a case, network operators need to migrate from one weight setting to another. The sequence in which the link weights are changed determines the disruption to network traffic during this migration process.

Fig. 1. Example Network

We illustrate this with the help of a toy example. Fig. 1 gives a network with the arc labels representing IGP link weights. Suppose all links have capacity c, and traffic demands between node pairs (a, g) and (b, g) are both $\frac{1}{4}c$. The traffic demand between all other node pairs is 0. The link weights depicted in Fig. 1 are optimal for such a traffic matrix given the objective of minimizing the maximum link utilization (MLU). Now suppose that the traffic demand between node pair (b, g) increases to $\frac{3}{4}c$. Shortest path routing using Equal Cost Multi-Path (ECMP) yields a new optimal weight setting that corresponds to all weights being 1 [10]. This means that three links weights ($w(c, e)$, $w(c, f)$, and $w(d, f)$) have to be

---

[1] Most common intra-domain (IGP) protocols.

[2] This paper is an extended version of our previous work [28].

Prior Work

Consensus Routing: The Internet as a Distributed System

R-BGP: Staying Connected In a Connected World

Nate Kushman Srikanth Kandula Dina Katabi Bruce M. Maggs

Graceful Network State Migrations

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John Ethan Katz-Bassett Arvind Krishnamurthy Thomas Anderson
Arun Venkataramani

R-BGP: Staying Connected In a Connected World

Nate Kushman Srikanth Kandula Dina Katabi Bruce M. Maggs

Graceful Network State Migrations

Dynamic Route Computation Considered Harmful

Matthew Caesar
University of Illinois at
Urbana-Champaign
caesar@cs.illinois.edu

Martín Casado
Nicira Networks Inc.
casado@nicira.com

Teemu Koponen
Nicira Networks Inc.
koponen@nicira.com

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Scott Shenker
University of California at
Berkeley
shenker@cs.berkeley.edu

ABSTRACT

This paper advocates a different approach to reduce routing convergence—side-stepping the problem by avoiding it in the first place! Rather than recomputing paths after temporary topology changes, we argue for a separation of timescale between *offline* computation of multiple diverse paths and *online* spreading of load over these paths. We believe decoupling failure recovery from path computation leads to networks that are inherently more efficient, more scalable, and easier to manage.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design

Keywords

Internet architecture, routing, convergence, protocols

1 Introduction

The traditional task assigned to routing is very clear: routing calculates paths based on the current view of the network topology. When the network changes, whether due to a transient failure[1] or a permanent change in the topology, routing recomputes these paths. In fact, the reactive nature of routing—dynamically computing paths in response to failures to ensure connectivity—is typically seen as crucial to the Internet's resilience. We disagree.

In fact, there are several problems with using routing to respond to failure. First, computing a new set of paths with a distributed routing algorithm can be slow, leading to noticeable outages. Attempts to speed up convergence often risk scalability (by increasing the frequency of message exchanges) or involve *ad hoc* modifications (such as route-flap damping and tuning MRAI timers). Moreover, this recomputation can be for naught, since equipment failures and planned maintenance can be extremely transient—often completing before routing has reconverged—leading to another recomputation once the link is restored. In addition, route recomputation involves a complicated distributed algorithm that: is hard to understand and debug; is often the source of bugs, vulnerabilities, and misconfigurations; can lead to "update storms" if not properly tuned; and can only scalably compute a limited range of routing options (e.g., it is difficult to compute various sets of disjoint paths with a scalable distributed algorithm). Lastly, in some cases (such as spanning tree and BGP), the recomputation of routes doesn't just fix broken paths, it also disrupts working ones.

In this paper, we argue that greater path diversity can, and should, reduce our reliance on failure-driven (i.e., "real-time") recomputation of routes. We are motivated by two trends in network and system design that improve an application's ability to tolerate failures:

- Multipath routing lets edge nodes (whether end hosts or ASes) circumvent failures by explicitly directing traffic over a different path.
- Data centers often replicate at the application layer, allowing a load balancer to direct flows to another server after a (host or network) failure.

We believe these trends should change the end-point's relationship with the network from "I'll accept whatever best-effort service you give me" to "I'll take action to improve my service". In the early days of the Internet, the emphasis was on end-points passively "adapting" to the current level

---

This article is an editorial note submitted to CCR. It has NOT been peer reviewed. Authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

[1] The term "failure" is overly narrow, since planned link maintenance is a common source of short-term service interruption, but for convenience we will use the term failure for all temporary outages.

Prior Work

Consensus Routing: The Internet as a Distributed System

R-BGP: Staying Connected In a Connected World

Graceful Network State Migrations

Dynamic Route Computation Considered Harmful

Prior Work

Consensus Routing: The Internet as a Distributed System

John P. John Ethan Katz-Bassett Arvind Krishnamurthy Thomas Anderson

Arun Venkataramani

Abstract

R-BGP: Staying Connected In a Connected World

Nate Kushman Srikanth Kandula Dina Katabi Bruce M. Maggs

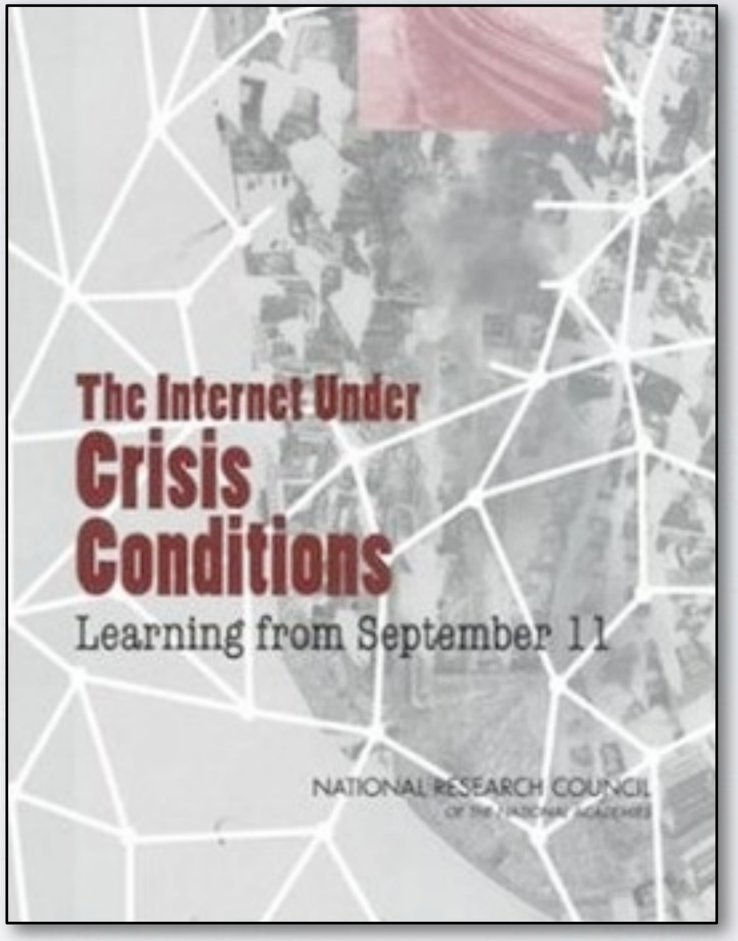
Graceful Network State Migrations

Dynamic Route Computation Considered Harmful

Matthew Caesar Martin Casado Teemu Koponen

Jennifer Rexford Scott Shenker

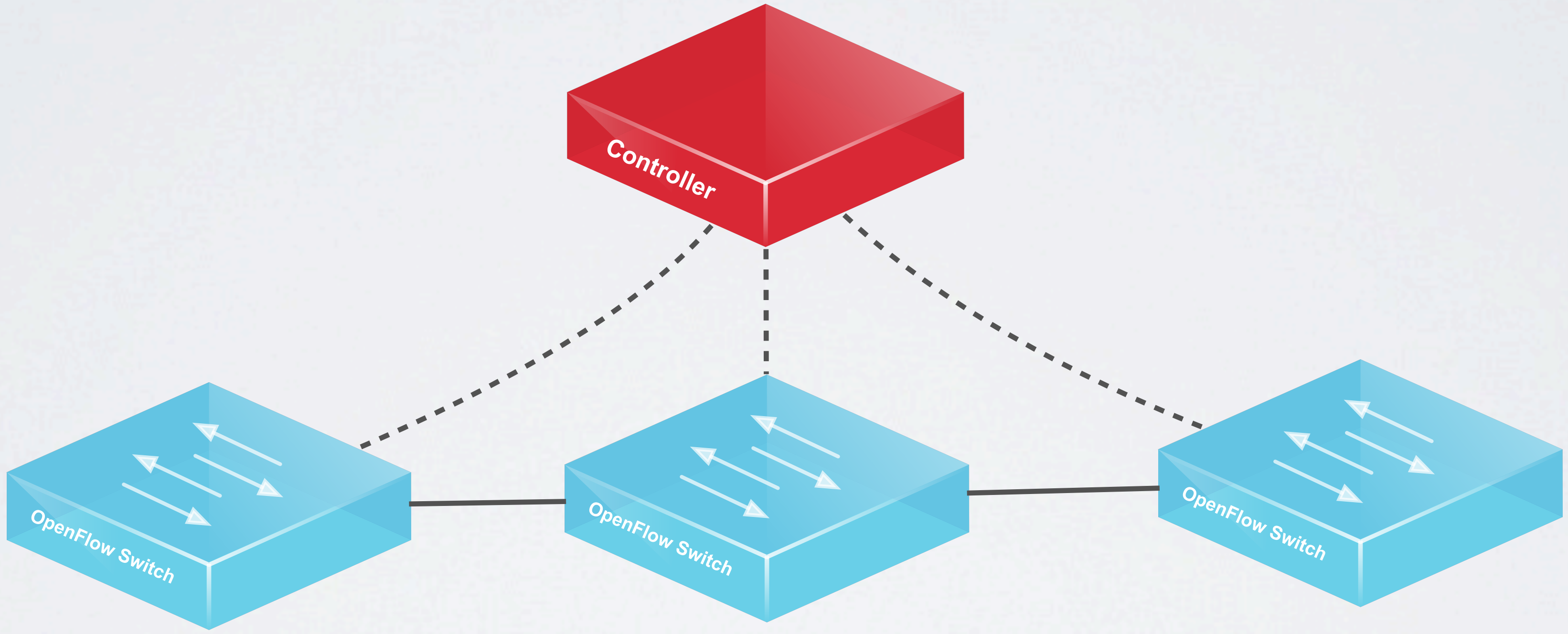
ABSTRACT



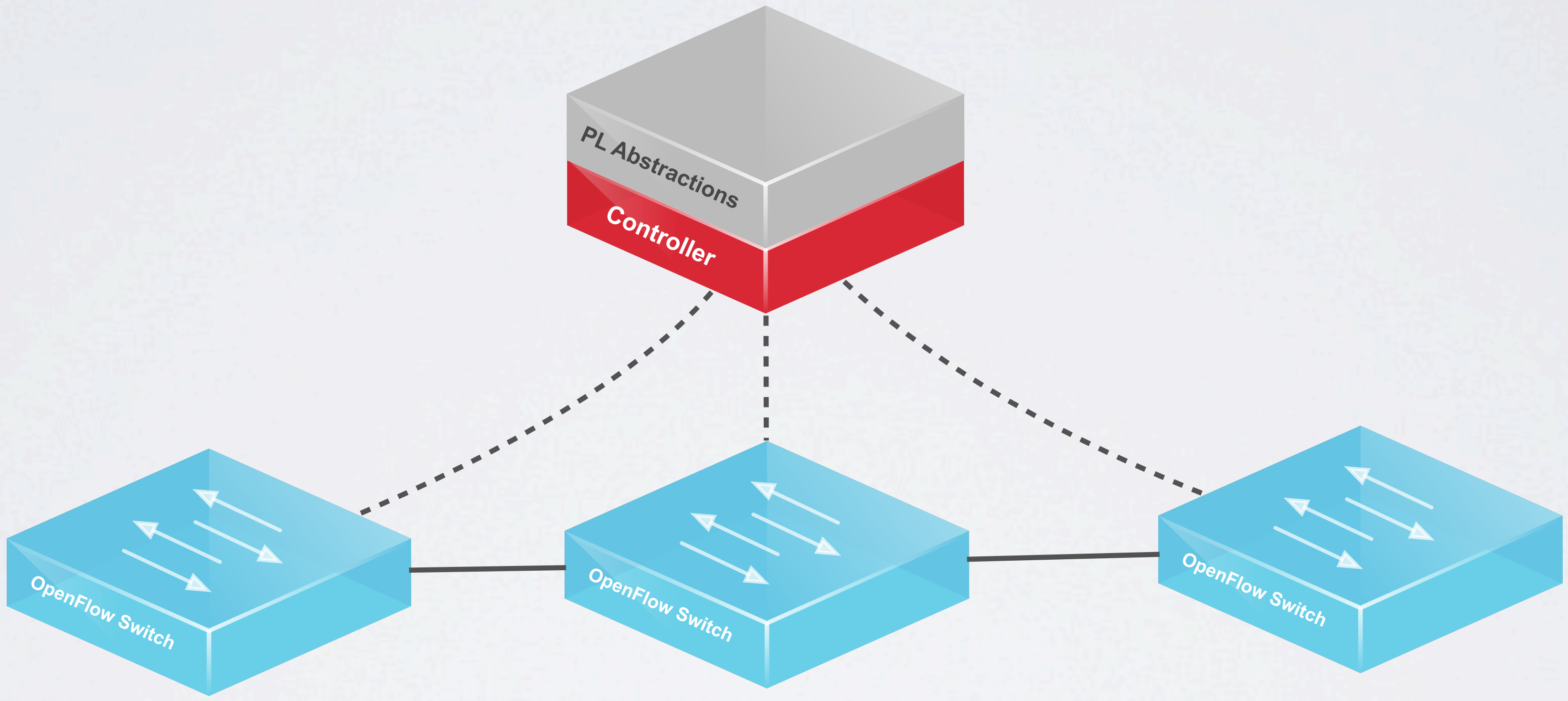
The Internet Under
Crisis
Conditions
Learning from September 11

NATIONAL RESEARCH COUNCIL
OF THE NATIONAL ACADEMIES

Software Abstractions

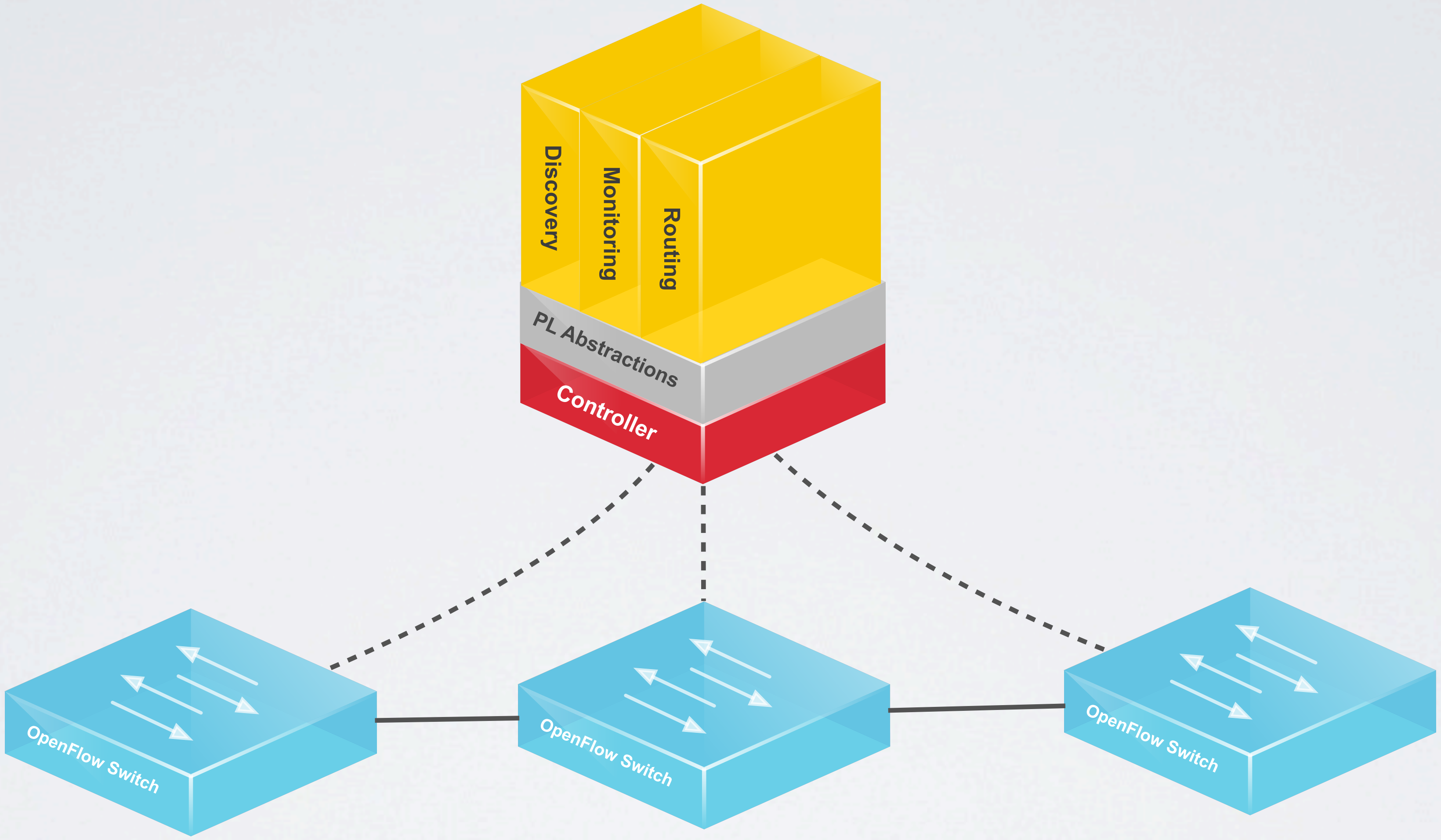


Software Abstractions



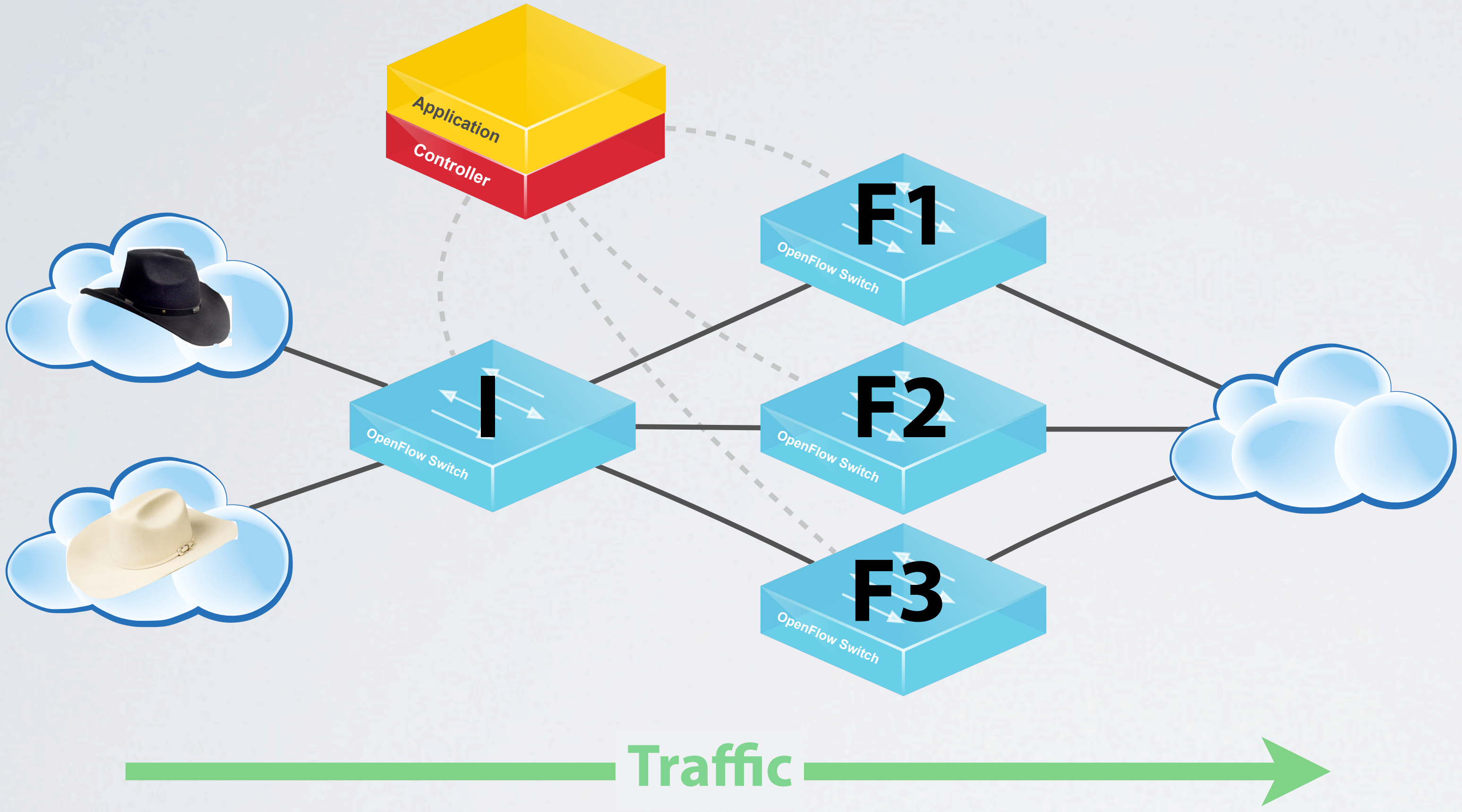
By designing the right software abstractions, we can solve the network update problem once and for all!

Software Abstractions



By designing the right software abstractions, we can solve the network update problem once and for all!

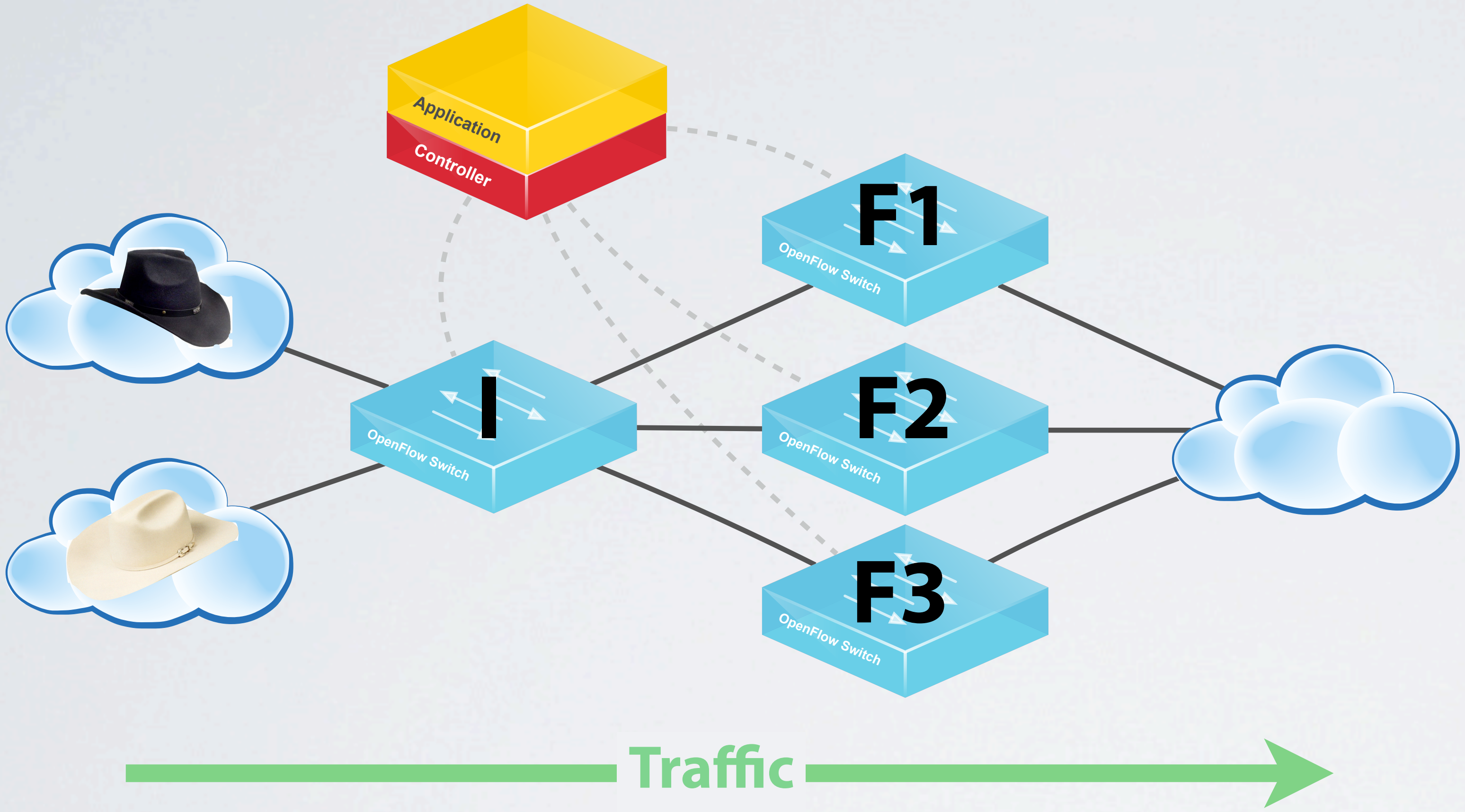
Example: Distributed Access Control



Security Policy

Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Example: Distributed Access Control



Security Policy

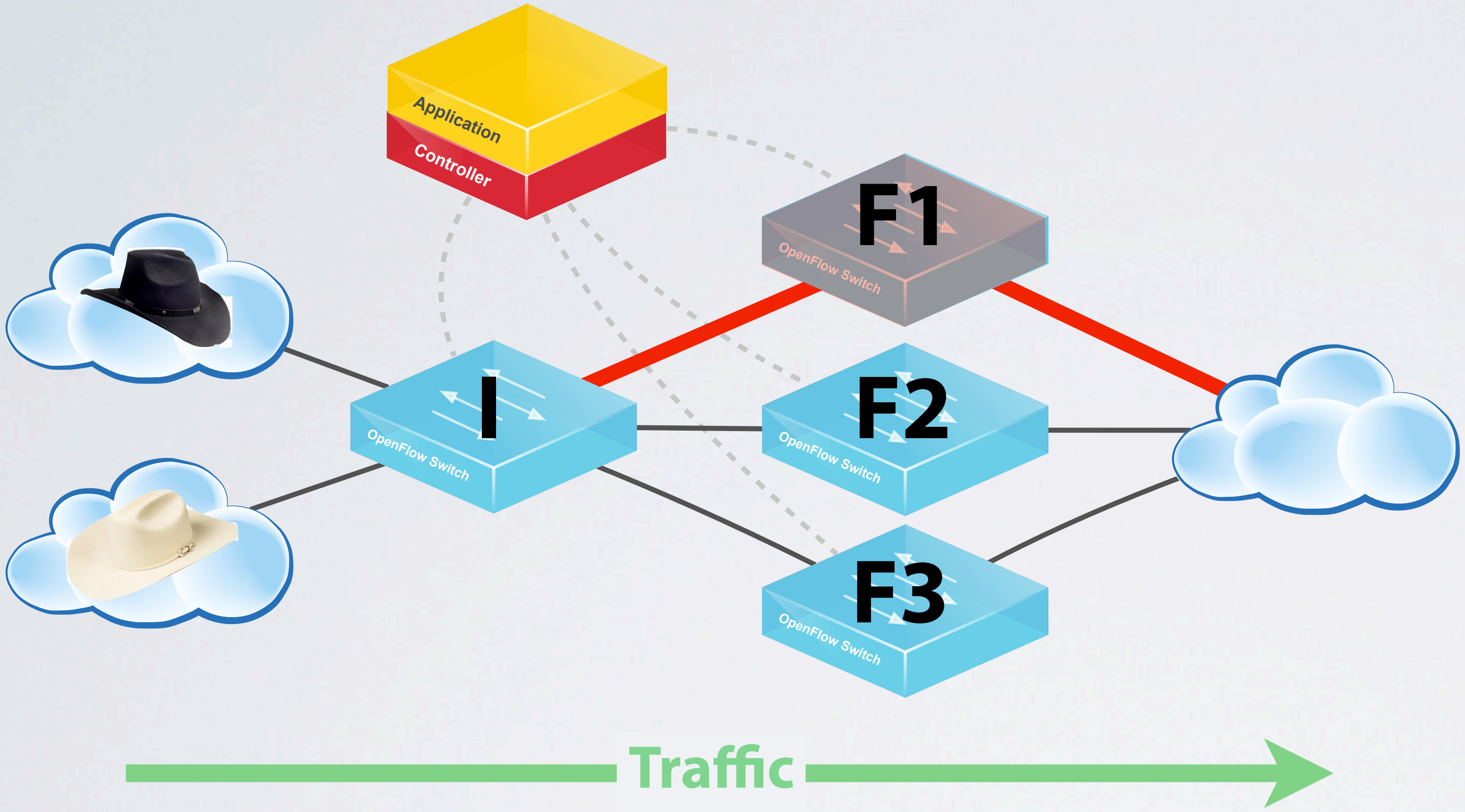
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A

Process black-hat traffic on F1

Process white-hat traffic on {F2,F3}

Example: Distributed Access Control



Security Policy

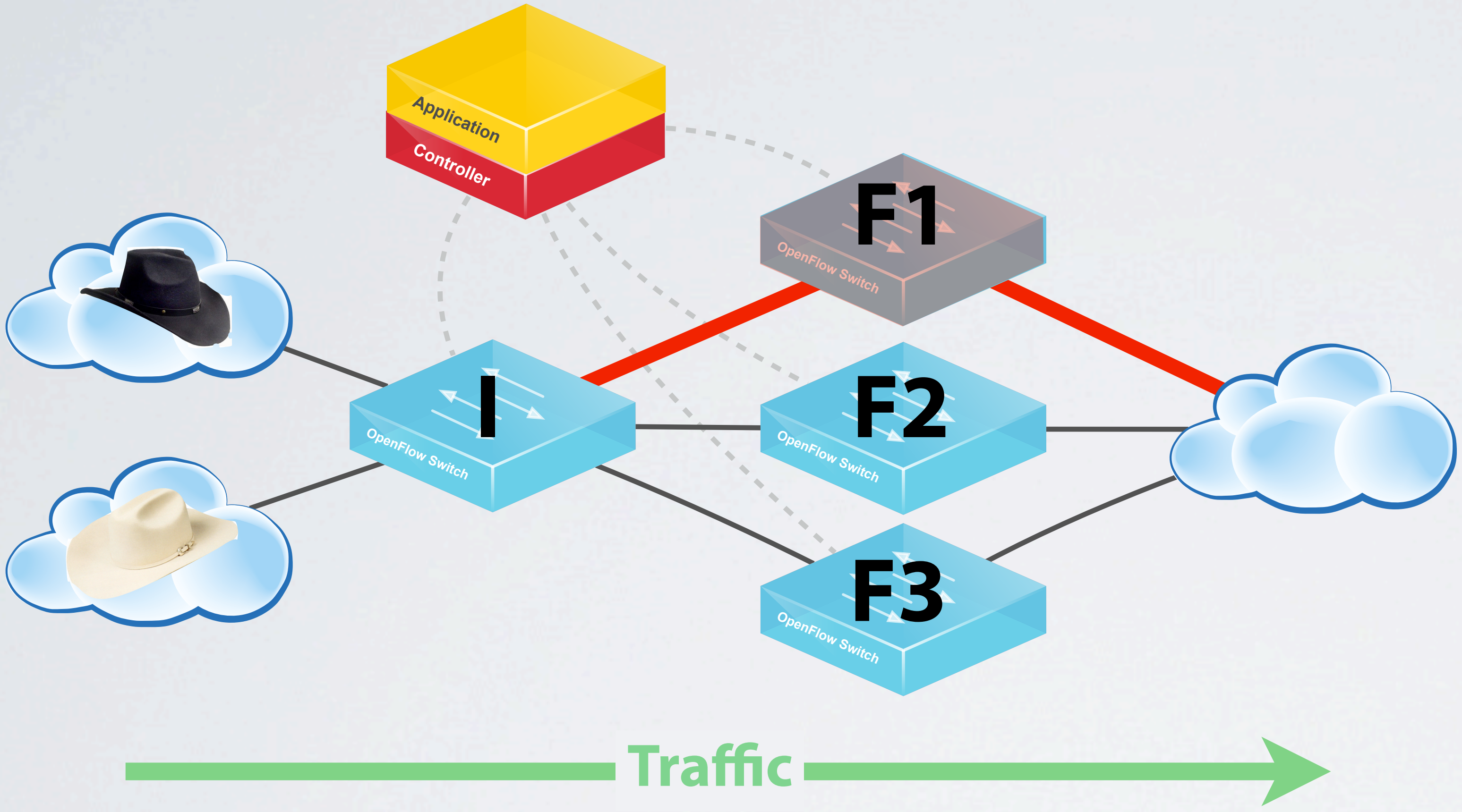
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A

Process black-hat traffic on F1

Process white-hat traffic on {F2,F3}

Example: Distributed Access Control



Security Policy

Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow

Configuration A

Process black-hat traffic on F1

Process white-hat traffic on {F2,F3}

? →

Configuration B

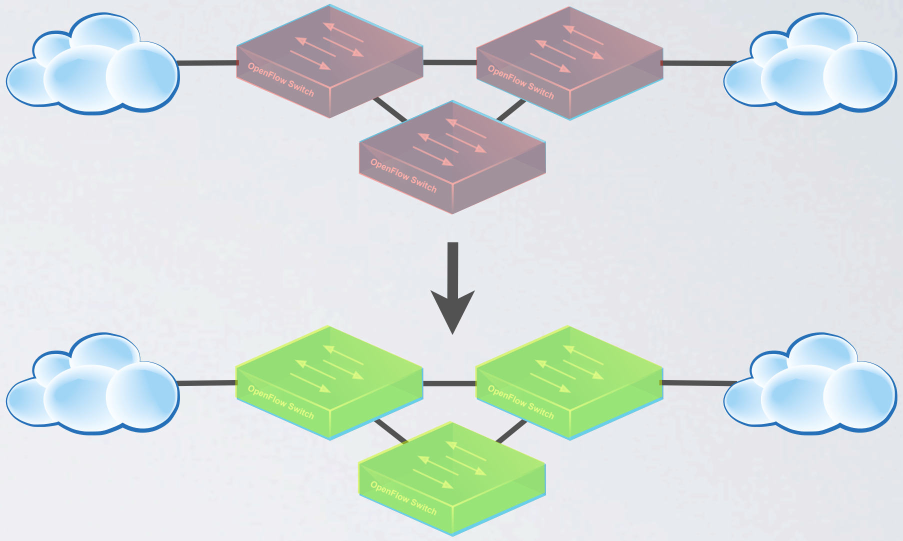
Process black-hat traffic on {F1,F2}

Process white-hat traffic on F3

Abstractions for Network Update

Challenge

- The network is a distributed system
- Can only update one element at a time



Our Approach

- Provide programmers with constructs for updating the entire network at once

```
update(config, topo)
```

- Design semantics to ensure "reasonable" behavior
- Engineer efficient implementation mechanisms
  - Compiler constructs low-level update protocols
  - Automatically applies optimizations

Consistent Updates in Action

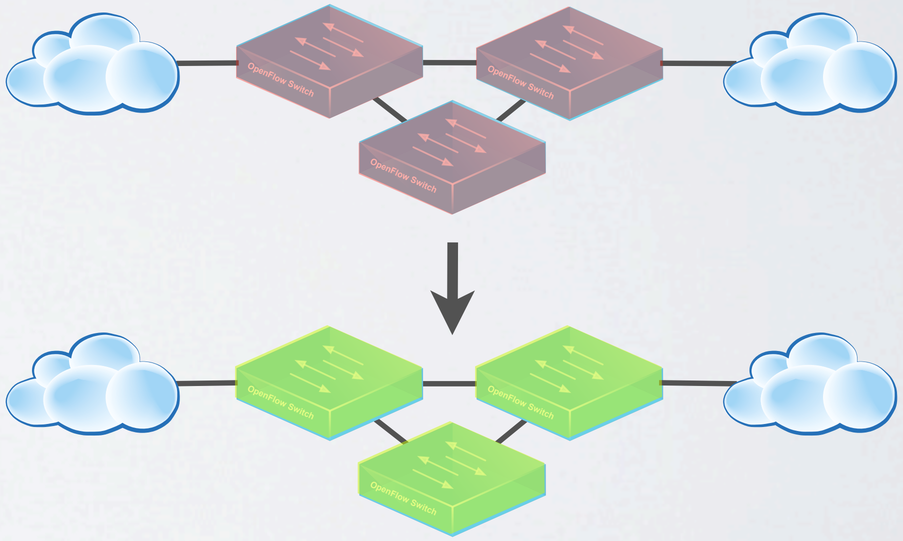
```
# Configuration A
I_



F1_

F2_
F3_
co
```

```
# Configuration B
I_configB = [Rule({IN_PORT:1},[forward(5)]),
             Rule({IN_PORT:2},[forward(6)]),
             Rule({IN_PORT:3},[forward(7)]),
             Rule({IN_PORT:4},[forward(7)])])
F1_configB = [Rule({TP_DST:80}, [forward(2)]),
              Rule({TP_DST:22}, [])])
F2_configB = [Rule({TP_DST:80}, [forward(2)]),
              Rule({TP_DST:22}, [])])
F3_configB = [Rule({},[forward(2)])]
configB = {I:SwitchConfiguration(I_configB),
           F1:SwitchConfiguration(F1_configB),
           F2:SwitchConfiguration(F2_configB),
           F3:SwitchConfiguration(F3_configB)}
```

```
# Main Function
topo = Topo(...)
update(configA, topo)
...wait for traffic load to shift...
update(configB, topo)
```

Security Policy

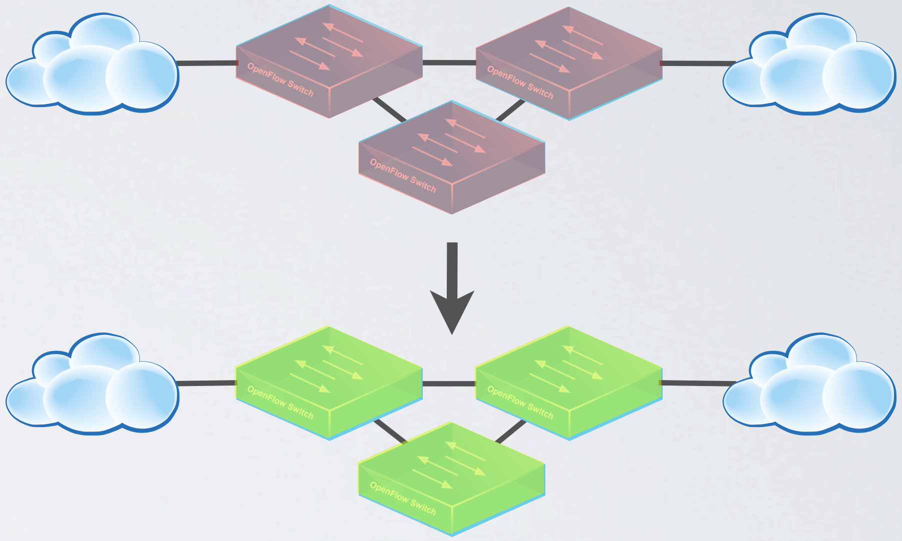
Src	Traffic	Action
	Web	Allow
	Non-web	Drop
	Any	Allow



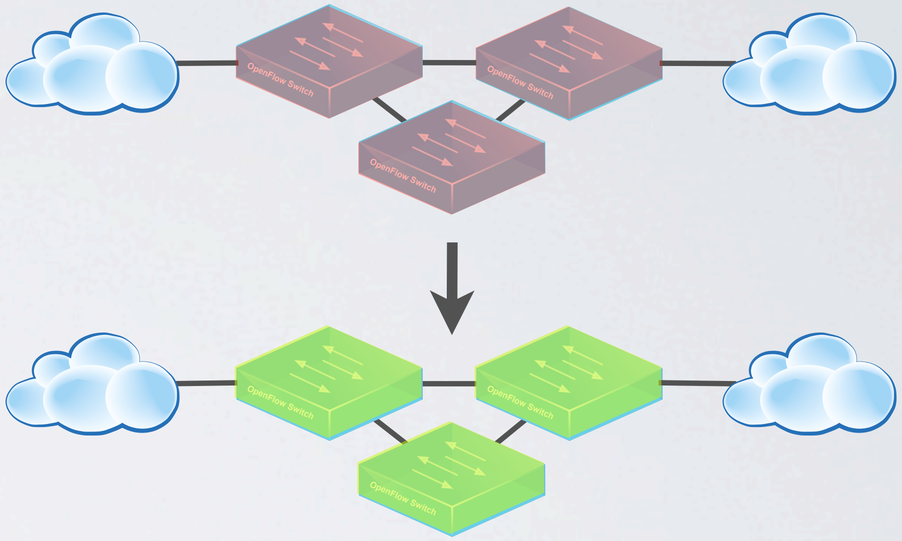
Semantics of Network Updates

Atomic Updates

- Seem sensible...
- but costly to implement
- and difficult to reason about effects on packets already in-flight



Semantics of Network Updates



Atomic Updates

- Seem sensible...
- but costly to implement
- and difficult to reason about effects on packets already in-flight

Per-Packet Consistent Updates

Every packet processed with old or new configuration, but not a mixture of the two

Implementation Mechanisms

Two-phase commit

- Construct versioned internal and edge configurations
- Phase 1: Install internal configuration
- Phase 2: Install edge configuration

Pure Extension

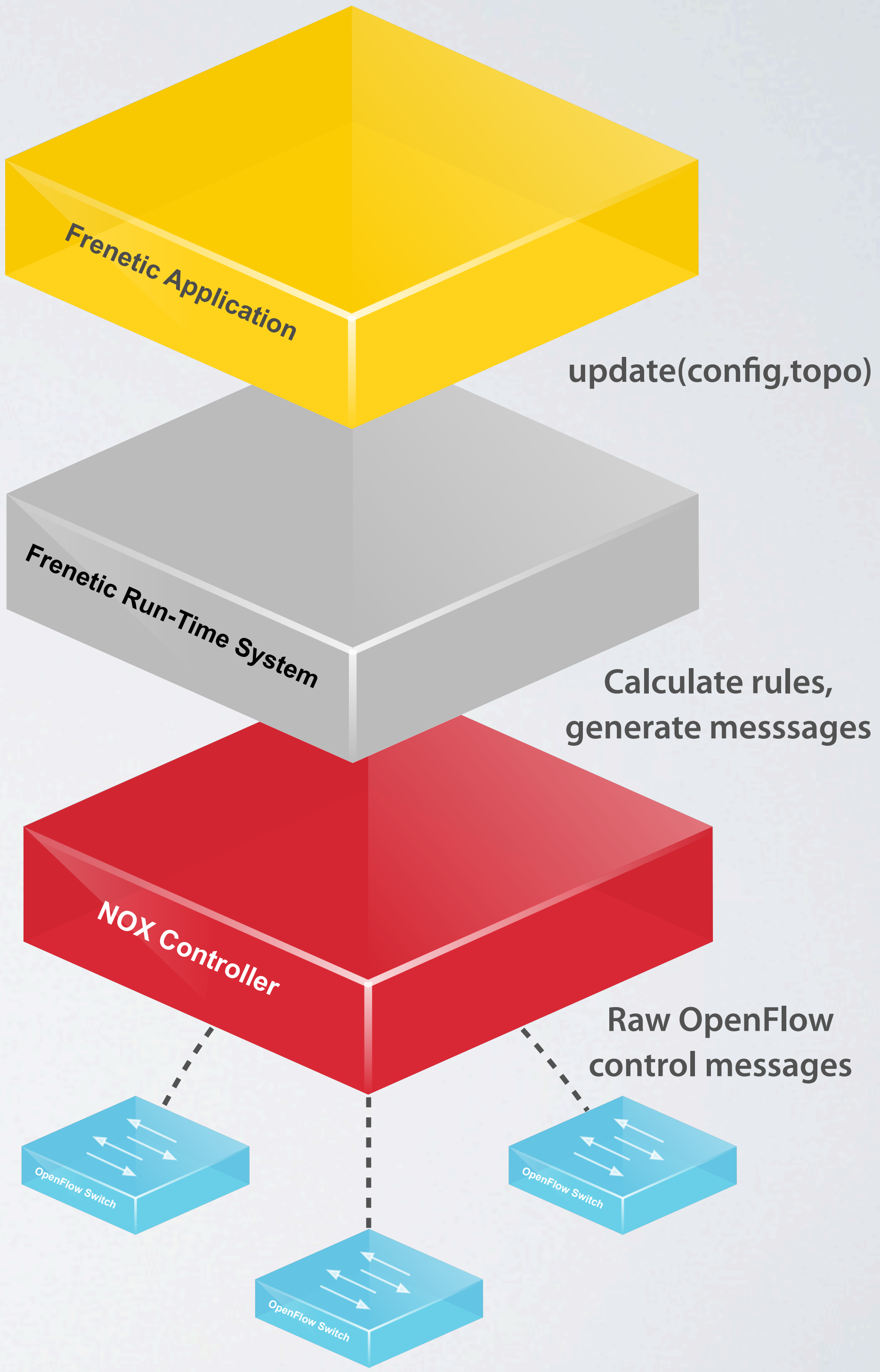
- Update strictly adds paths

Pure Retraction

- Update strictly removes paths

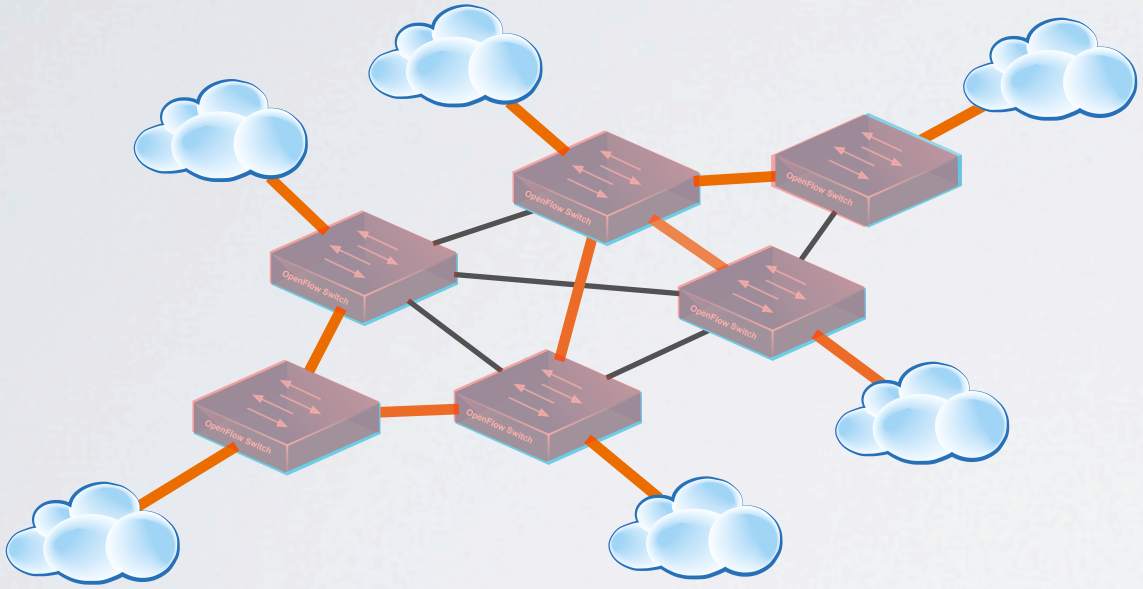
Slice Update

- Update affects a small number of switches

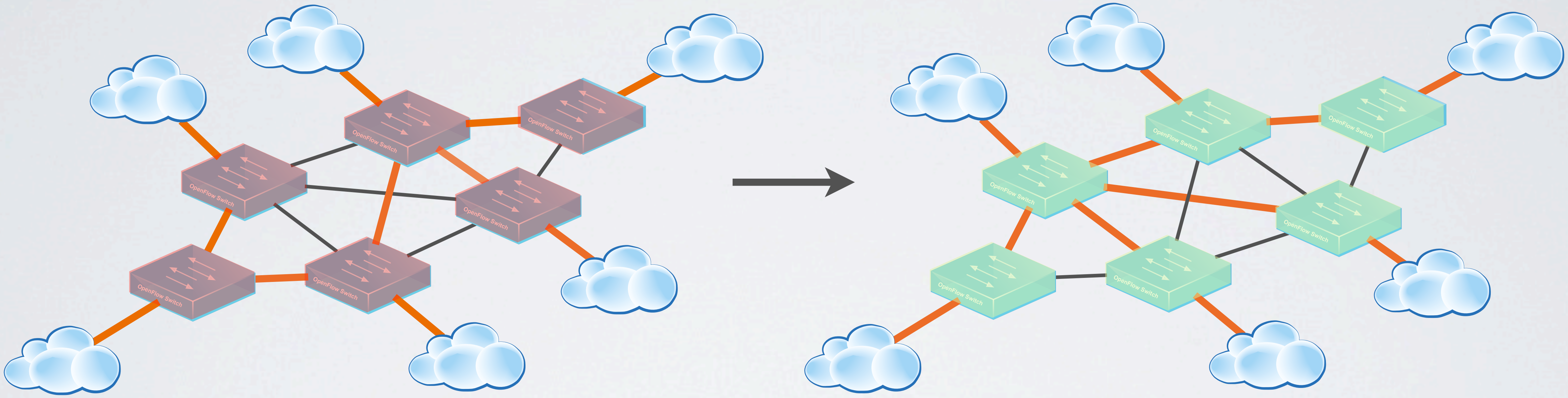


(Ask me for a demo!)

(Ask me for a demo!)



(Ask me for a demo!)



Formal Verification

Global Config

Update

Packet Queue

$$\langle C, Q \rangle \xrightarrow{u} \langle C', Q' \rangle$$

Formal Verification

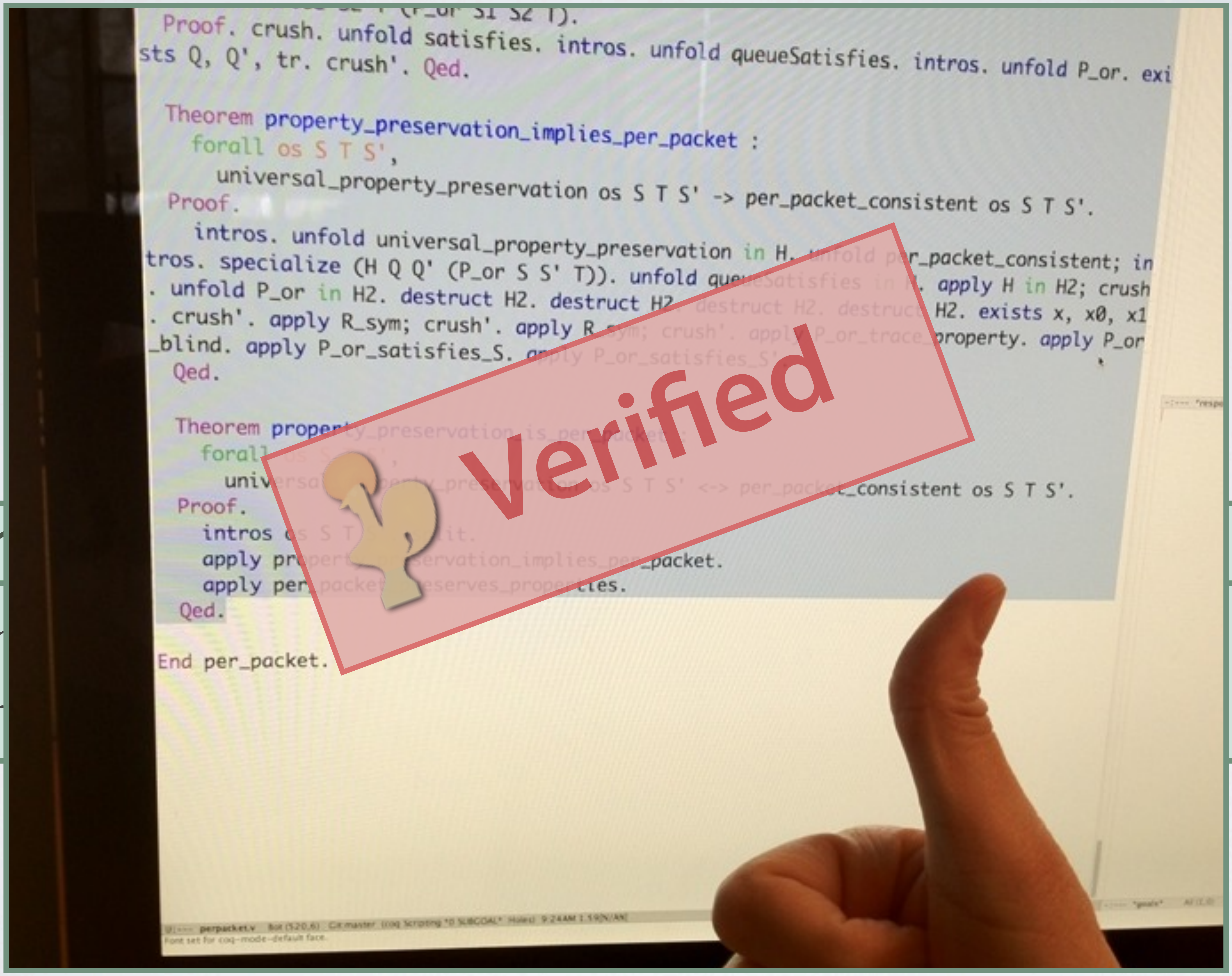
Global Config **Update** **Packet Queue**

$$\langle C, Q \rangle \xrightarrow{u} \langle C', Q' \rangle$$

Theorem

An update u C_1 to C_2 is per-packet consistent if and only if it preserves all properties satisfied by C_1 and C_2.

Formal Verification



```
(P_or S1 S2 T).
  Proof. crush. unfold satisfies. intros. unfold queueSatisfies. intros. unfold P_or. exi
sts Q, Q', tr. crush'. Qed.

  Theorem property_preservation_implies_per_packet :
    forall os S T S',
      universal_property_preservation os S T S' -> per_packet_consistent os S T S'.
  Proof.
    intros. unfold universal_property_preservation in H. unfold per_packet_consistent; in
tros. specialize (H Q Q' (P_or S S' T)). unfold queueSatisfies in H. apply H in H2; crush
. unfold P_or in H2. destruct H2. destruct H2. destruct H2. destruct H2. exists x, x0, x1
. crush'. apply R_sym; crush'. apply R_sym; crush'. apply P_or_trace_property. apply P_or
_blind. apply P_or_satisfies_S. apply P_or_satisfies_S'.
  Qed.

  Theorem property_preservation_is_per_packet :
    forall os S T S',
      universal_property_preservation os S T S' <-> per_packet_consistent os S T S'.
  Proof.
    intros os S T S'; split.
    apply property_preservation_implies_per_packet.
    apply per_packet_preserves_properties.
  Qed.

End per_packet.
```

Th

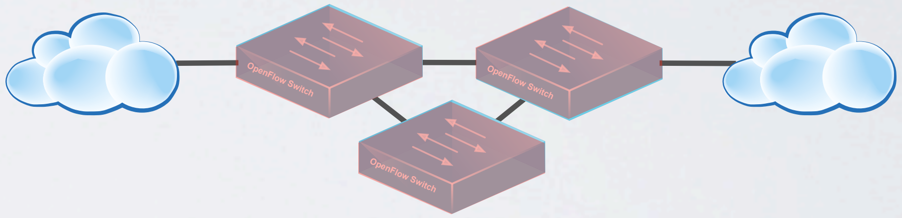
Ar t

pr

Formal Verification

Corollary

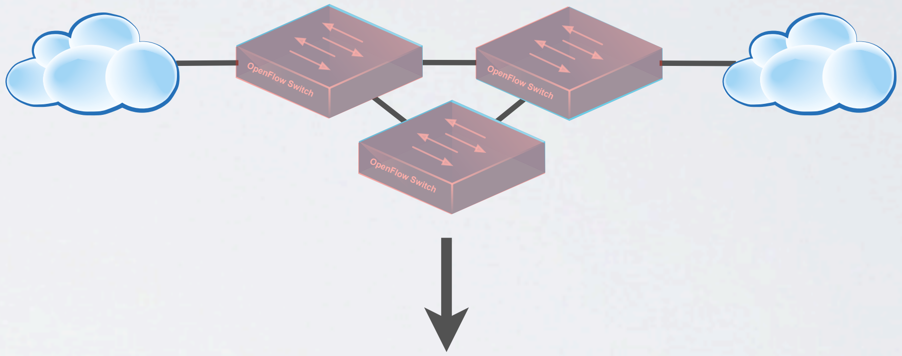
To verify that a property is invariant, simply check that the old and new configurations satisfy it



Formal Verification

Corollary

To verify that a property is invariant, simply check that the old and new configurations satisfy it

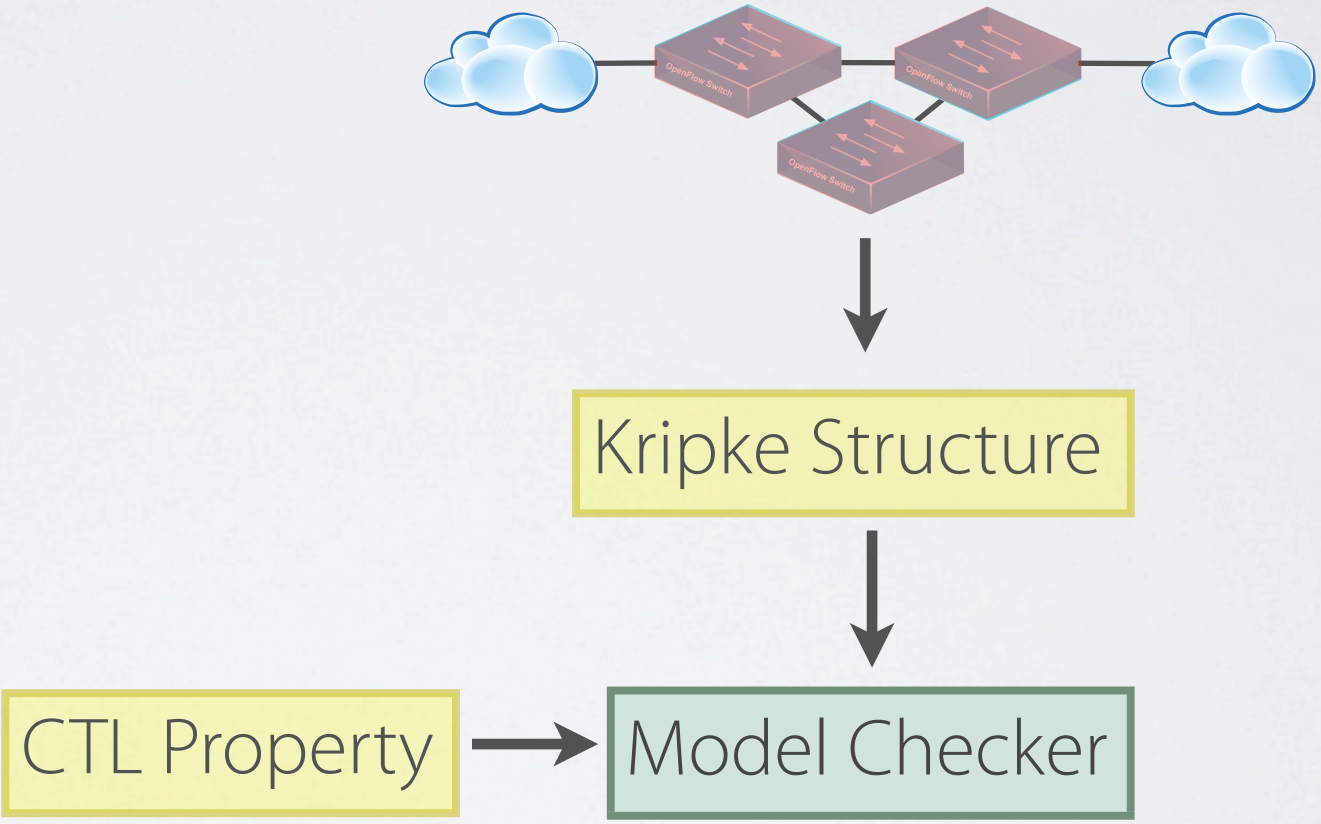


Kripke Structure

Formal Verification

Corollary

To verify that a property is invariant, simply check that the old and new configurations satisfy it



Kripke Structure

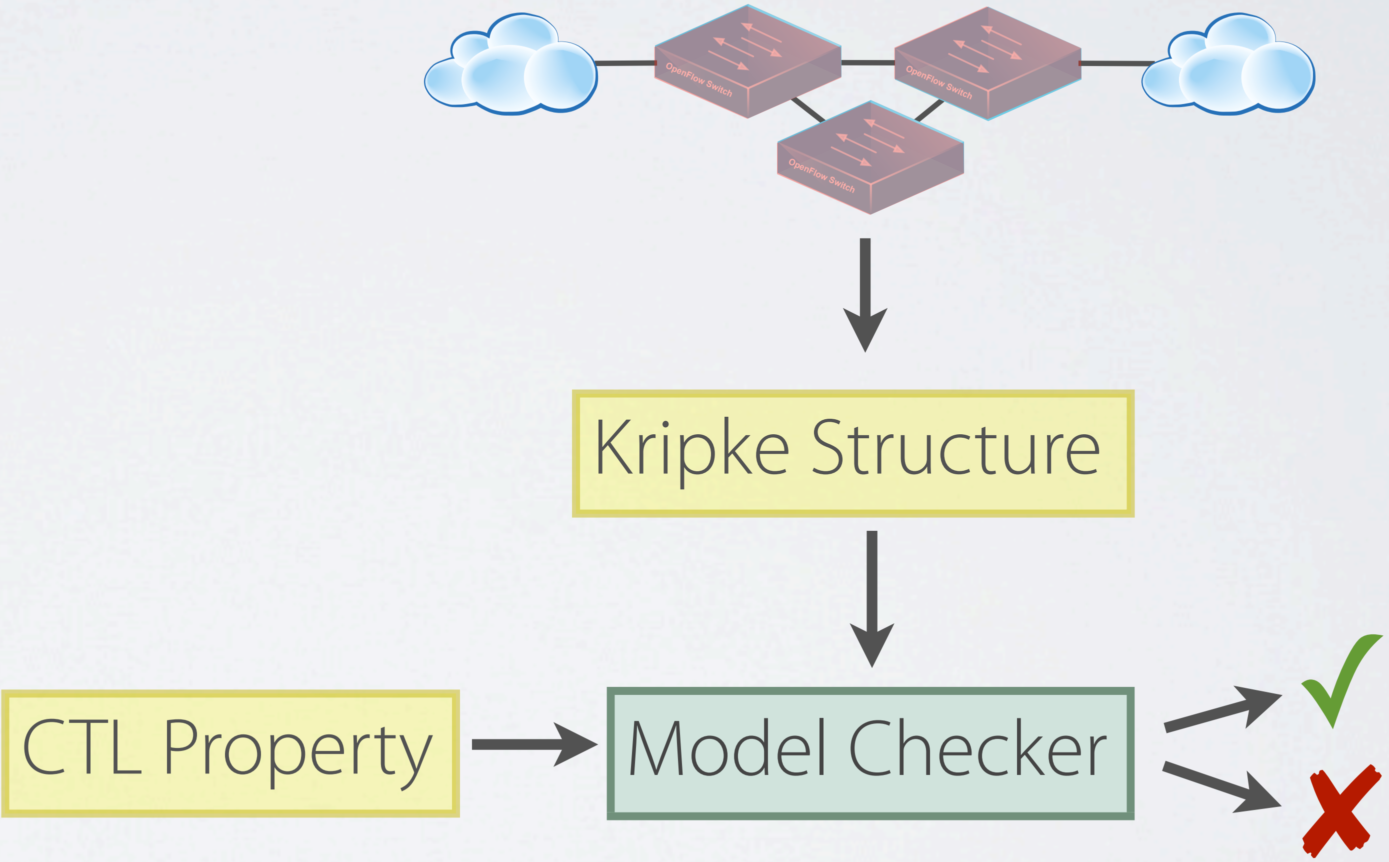
CTL Property

Model Checker

Formal Verification

Corollary

To verify that a property is invariant, simply check that the old and new configurations satisfy it



Kripke Structure

CTL Property → Model Checker → ✓ / ✗

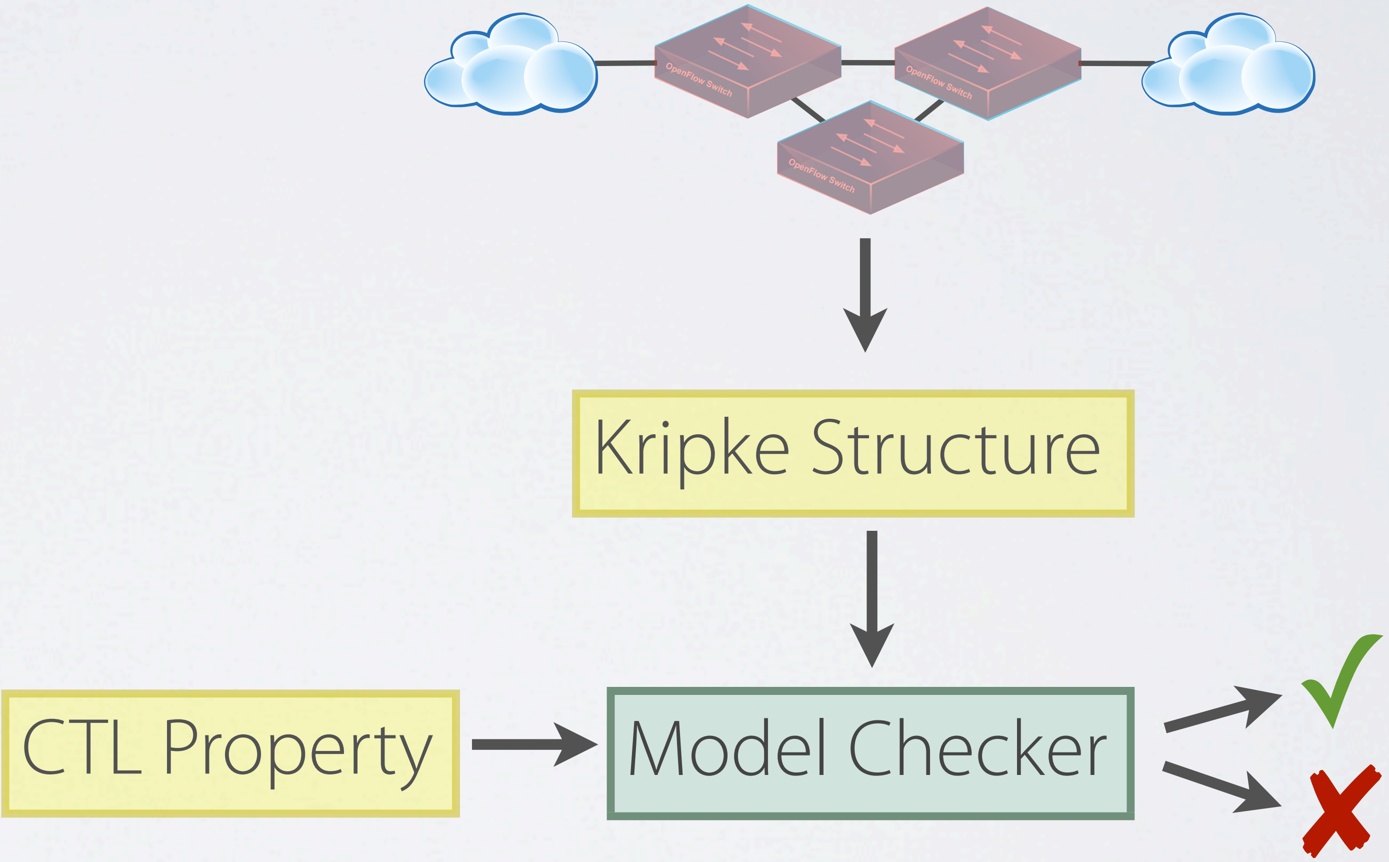
Formal Verification

Corollary

To verify that a property is invariant, simply check that the old and new configurations satisfy it

Properties

- Connectivity
- Loop freedom
- Blackhole freedom
- Access control
- Waypointing
- Totality



Kripke Structure

CTL Property

Model Checker

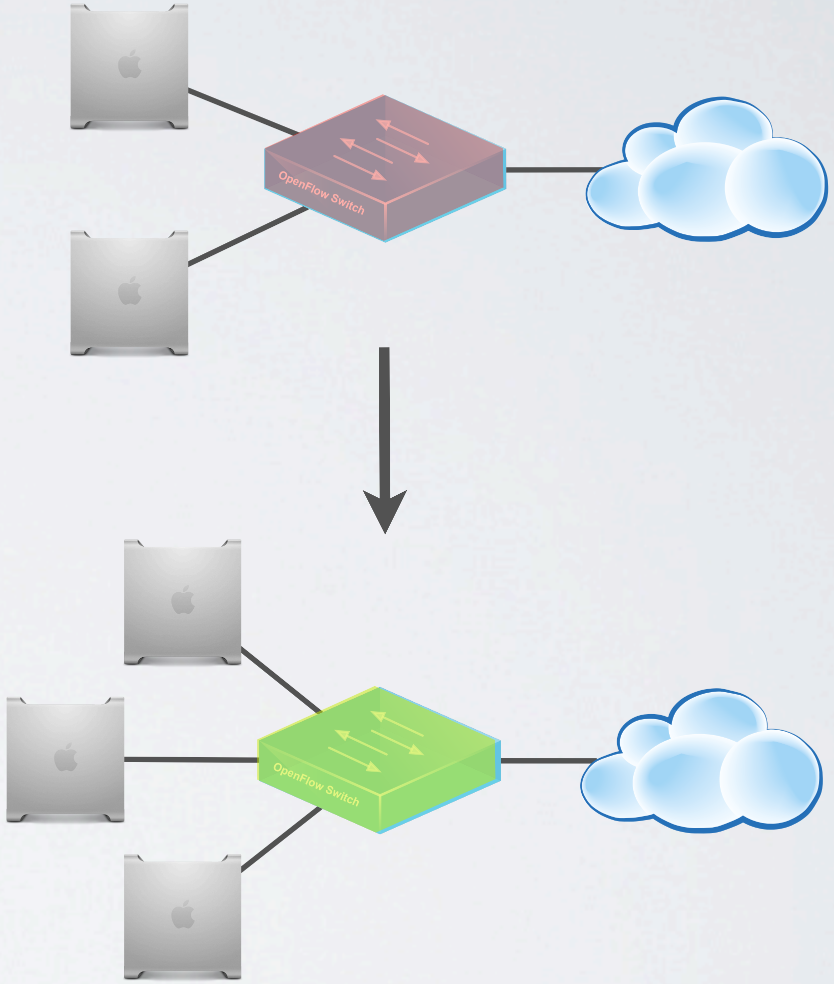
✓

✗

Per-Flow Consistency

Use Cases

- Load balancer
- Flow affinity
- In-order delivery



Per-flow consistent updates

Every *set* of related packets processed with old or new configuration, but not a mixture of the two.

Implementation mechanisms

- Need to identify active flows
- Rules with soft timeouts
- DevoFlow wildcard cloning
- End-host feedback

Ongoing Work

Other abstractions

- Loop-freedom
- Affinity preserving

Update Synthesis

- Programmer specifies an invariant
- Compiler constructs an update protocol

Enhanced fault tolerance

- Rapid response when failures occur
- Compiler "hardens" configurations
- Pre-loads backup policy

Leverage end hosts

- Help identify active flows

Thank You!

Collaborators

- Shrutarshi Basu (Cornell)
- Mike Freedman (Princeton)
- Rob Harrison (West Point)
- Chris Monsanto (Princeton)
- **Mark Reitblatt** (Cornell)
- Gün Sirer (Cornell)
- **Cole Schlesinger** (Princeton)
- Alec Story (Cornell)
- **Jen Rexford** (Princeton)
- **Dave Walker** (Princeton)

Funding

frenetic >>

http://frenetic-lang.org