Abstractions for Network Update

Nate Foster
Mark Reitblatt

Cole Schlesinger
Jennifer Rexford
David Walker

frenetic >>

amazon webservices™

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

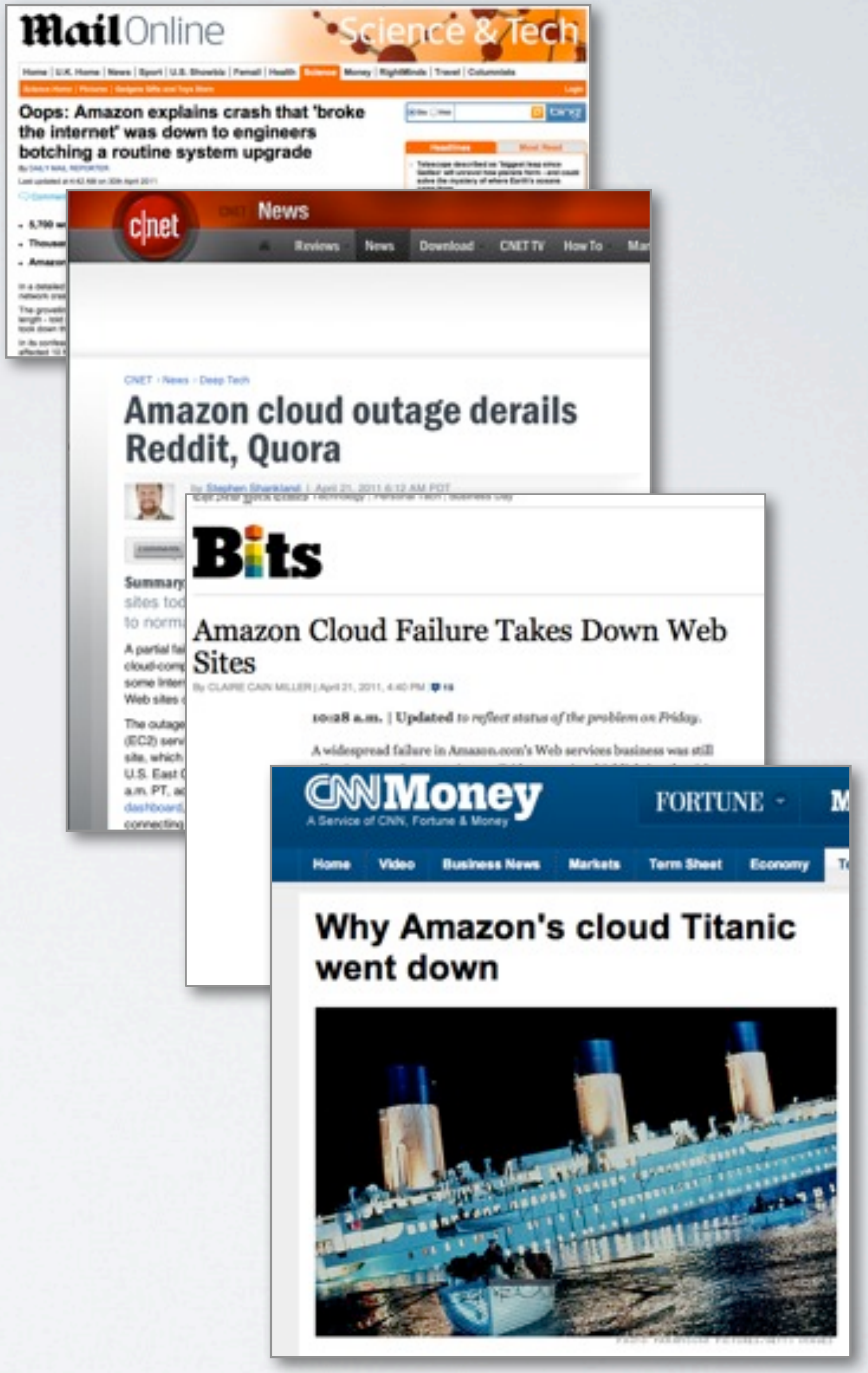
During the change, one of the steps is to shift traffic off of one of the redundant routers...

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

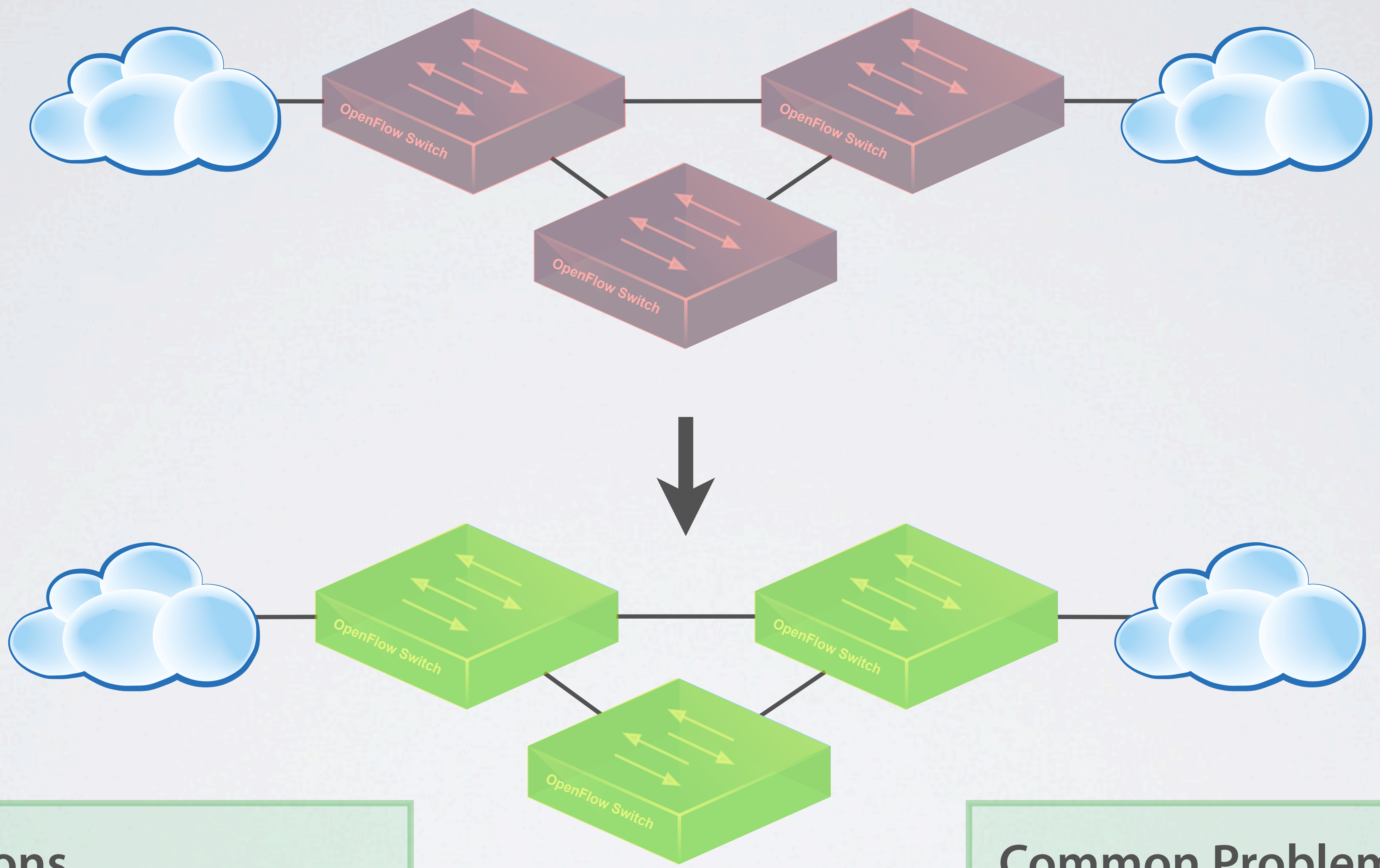
This led to a “re-mirroring storm”...

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**.



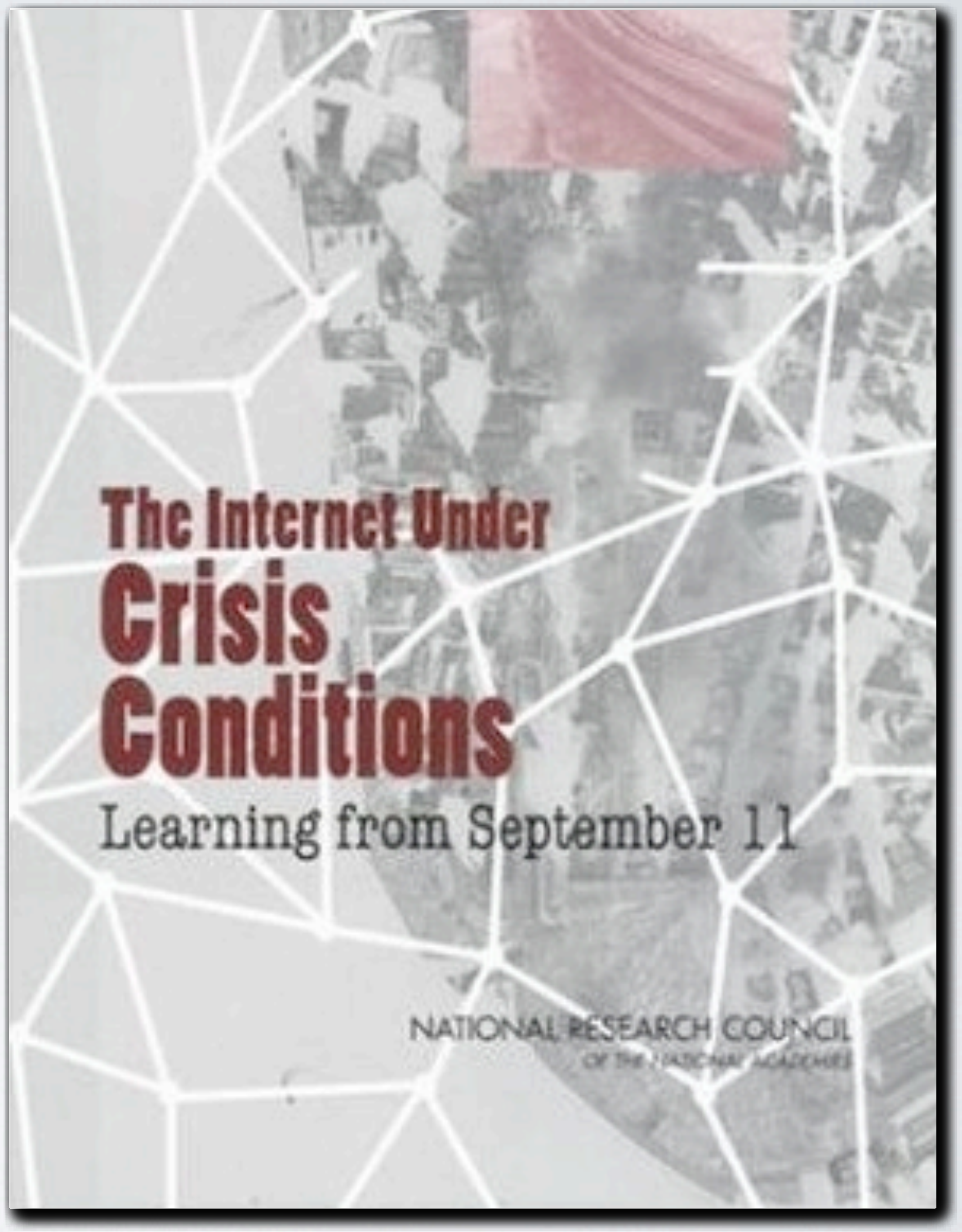
Network Updates



Reasons
- Routine maintenance
- Unexpected failure
- Traffic engineering
- Fine-grained security

Common Problems
- Lost packets
- Broken connections
- Forwarding loops
- Security vulnerabilities

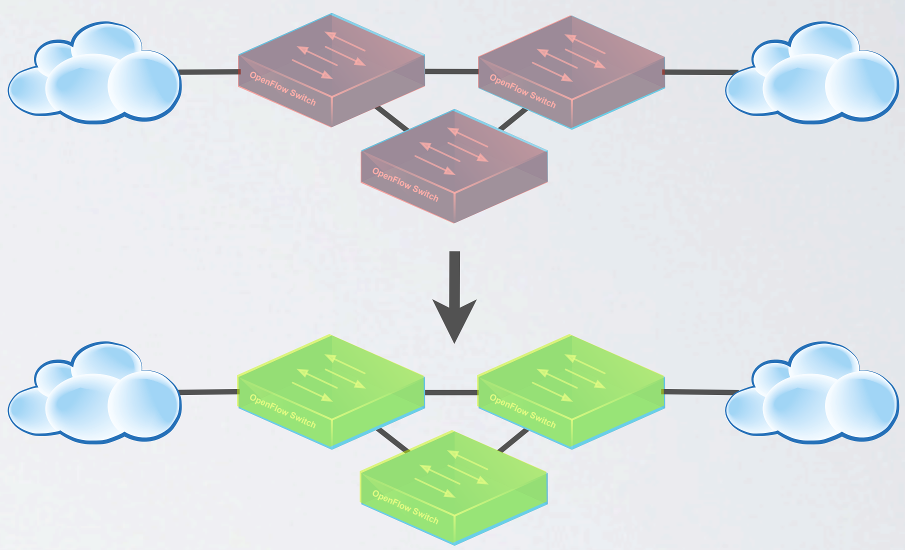
Prior Work


The Internet Under
Crisis
Conditions
Learning from September 11
NATIONAL RESEARCH COUNCIL

Kinetic: Abstractions for Network Update

Main Challenge

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



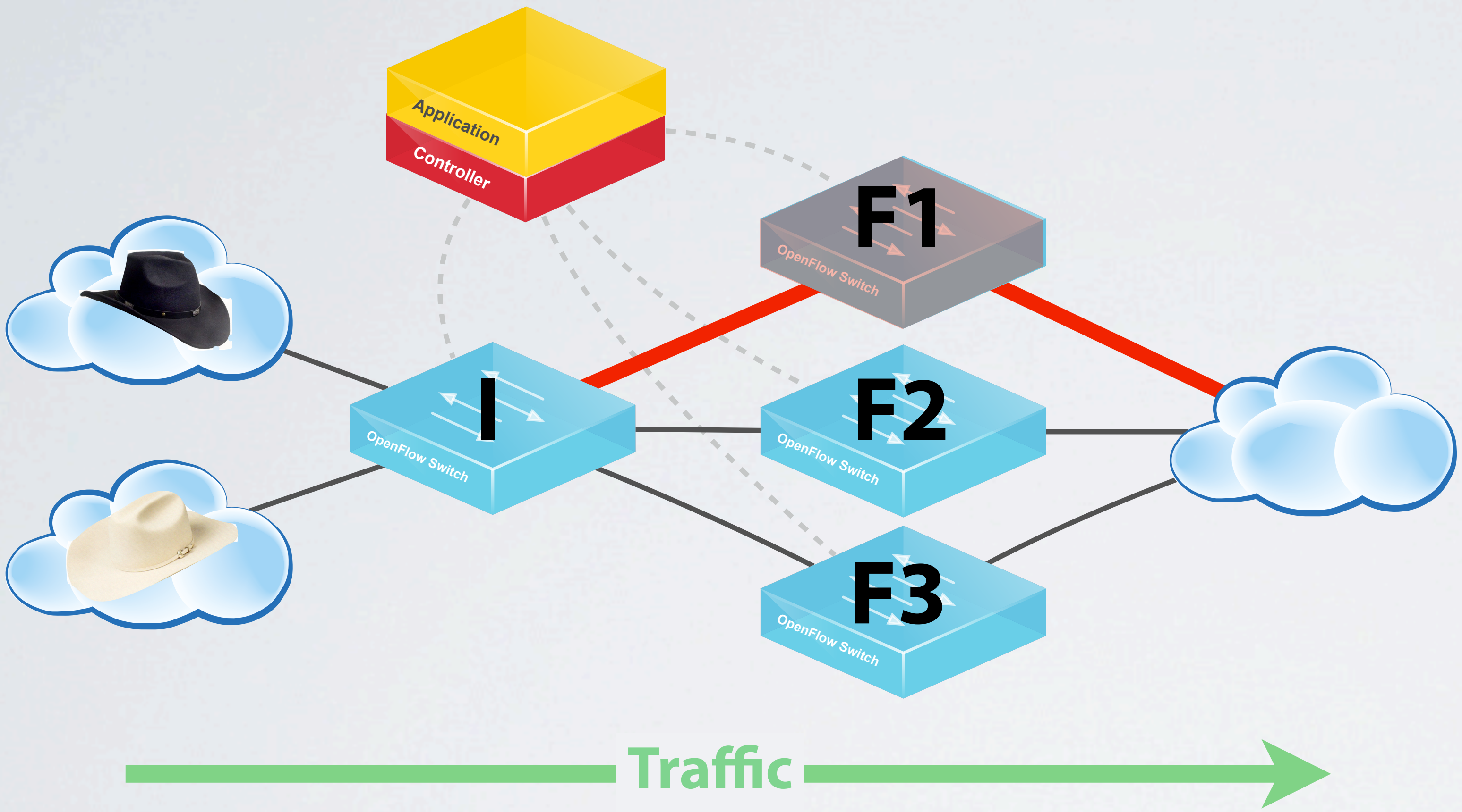
Kinetic Approach

- Provide programmers with abstractions for updating the whole 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

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}

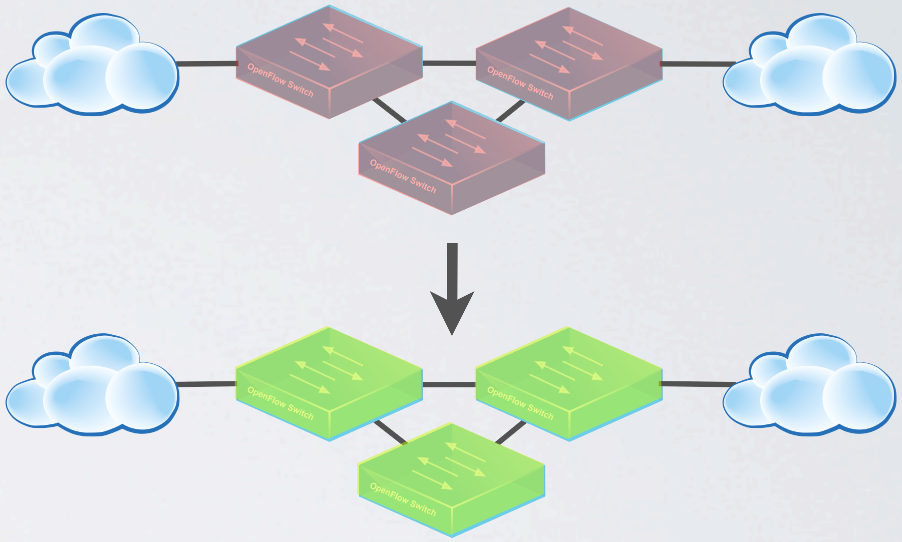
? →

Configuration B
Process black-hat traffic on {F1,F2} Process white-hat traffic on F3

Update Semantics

Atomic Updates

- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets



Per-Packet Consistent Updates

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

Per-Flow Consistent Updates

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

Per-Packet Consistency, Formally

Global Configuration — **Update** — **Queue**

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

Queues

- Keep track of packets waiting to be processed at each location
- Record the full processing history of each packet as a *trace*

Definition (Per-Packet Consistency)

An update us is *per-packet consistent* if for every Q' and t such that $\langle C_1, Q \rangle \xrightarrow{us}{}^{\star} \langle C_2, Q' \rangle$ and $t \in Q'$, there exist Q_i and Q'' such that $\langle C_1, Q_i \rangle \longrightarrow^{\star} \langle C_1, Q'' \rangle$ or $\langle C_2, Q_i \rangle \longrightarrow^{\star} \langle C_2, Q'' \rangle$ and $t \in Q''$.

Properties

Trace: sequence of port-packet pairs

Property: prefix-closed set of traces

Satisfaction:

- $Q \models P$: every trace in Q an element of P
- $C \models P$: every queue generated by C satisfies P

Definition (Universal Property Preservation)

An update us is *universal property preserving* if for all properties P such that $C_1 \models P$ and $C_2 \models P$ and all executions $\langle C_1, Q \rangle \xrightarrow{us}{}^{\star} \langle C_2, Q' \rangle$ we have that $Q' \models P$.

Theorem

An update us is per-packet consistent if and only if it is universal property preserving.

Consistent Update Classes

One-Touch Update

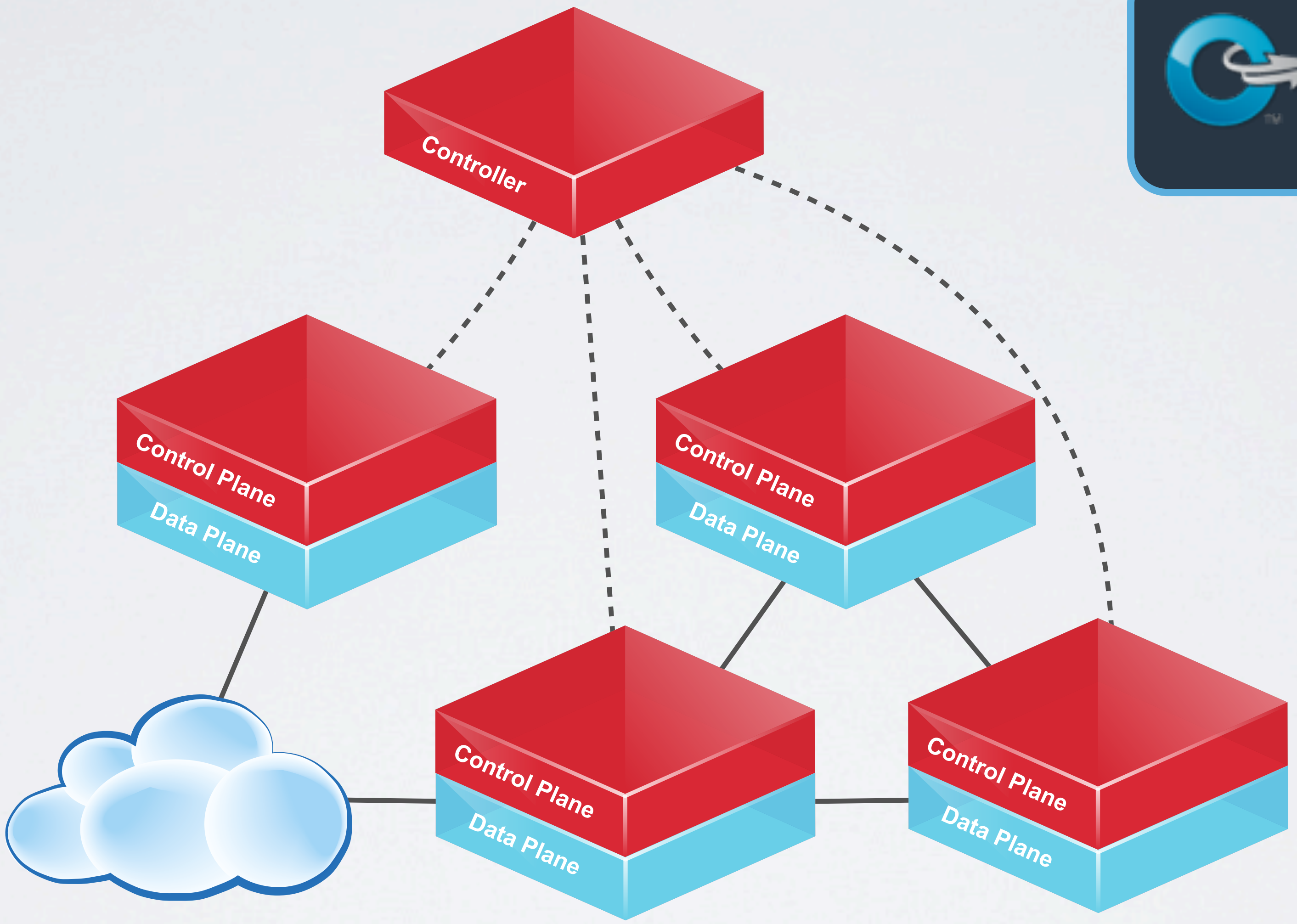
- Packets traverse updated region of the network at most once
- Example: Loop-free single switch update
- Example: Ingress port update

Unobservable Update

- Set of traces generated by new configuration unchanged
- Example: Internal switch update

Theorem (Composition Principle)
The composition of an unobservable update and a per-packet consistent update is a per-packet consistent update.

Implementation Architecture



OpenFlow

Implementation

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

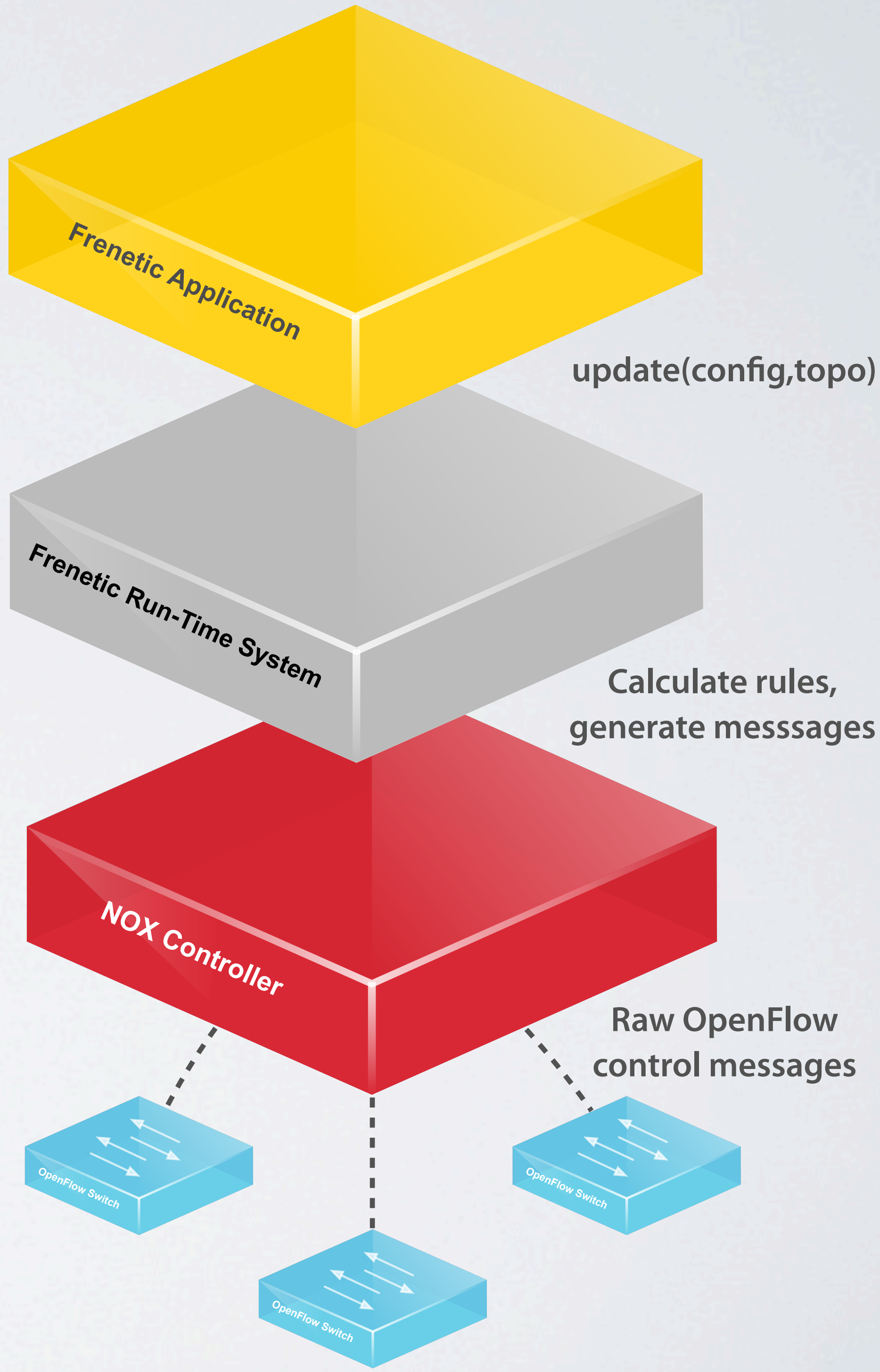
- Update strictly removes paths

Sub-space update

- Update modifies a small number of paths

Sub-topology update

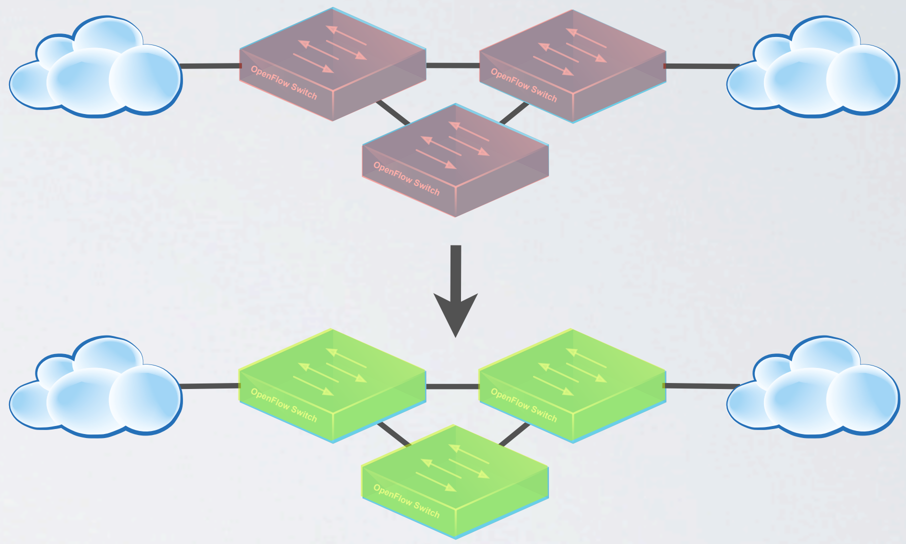
- Update affects a small number of switches



The Abstractions at Work

```
# 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)}
```

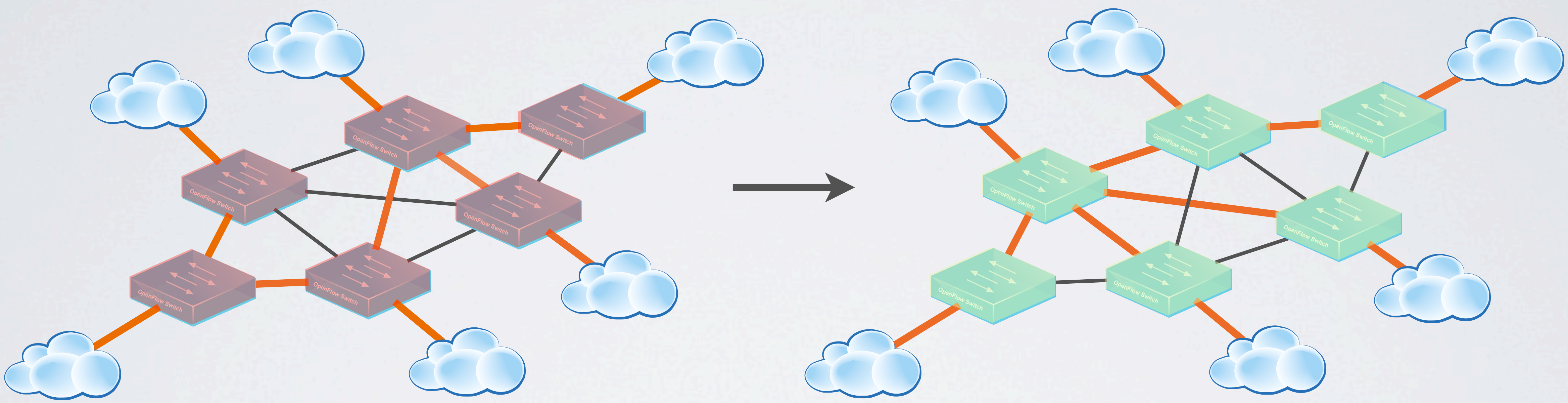


```
# Main Function
topo = NXTopo(...)
per_packet_update(configA, topo)
...wait for traffic load to shift...
per_packet_update(configB, topo)
```

Other Examples

- Point-to-point routing
- Multicast routing
- Application load balancer

Demo



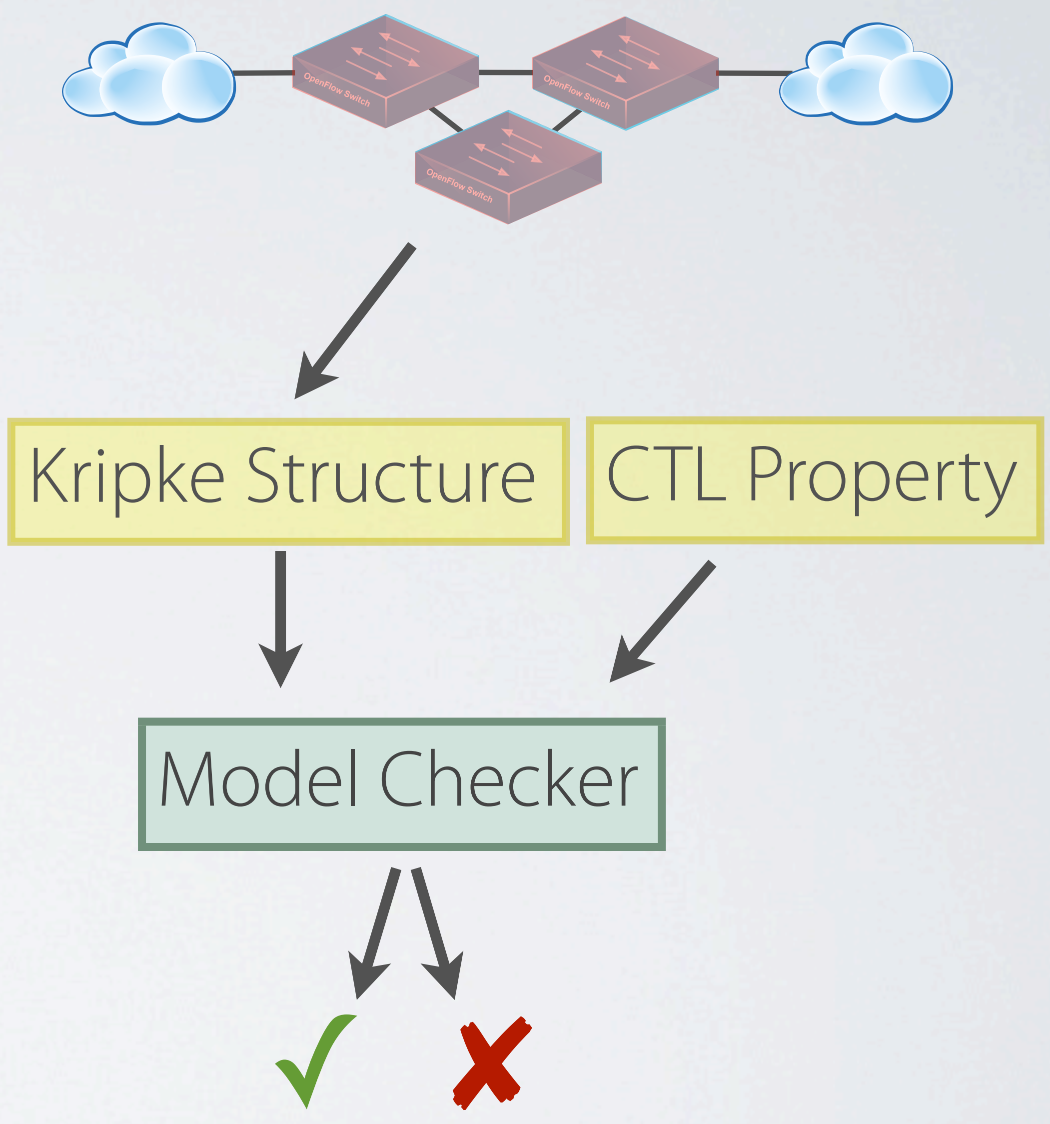
Formal Verification

Universal Preservation 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



Per-Flow Consistency

Use Cases

- In-order delivery
- Load balancer

Per-Packet Consistent Update

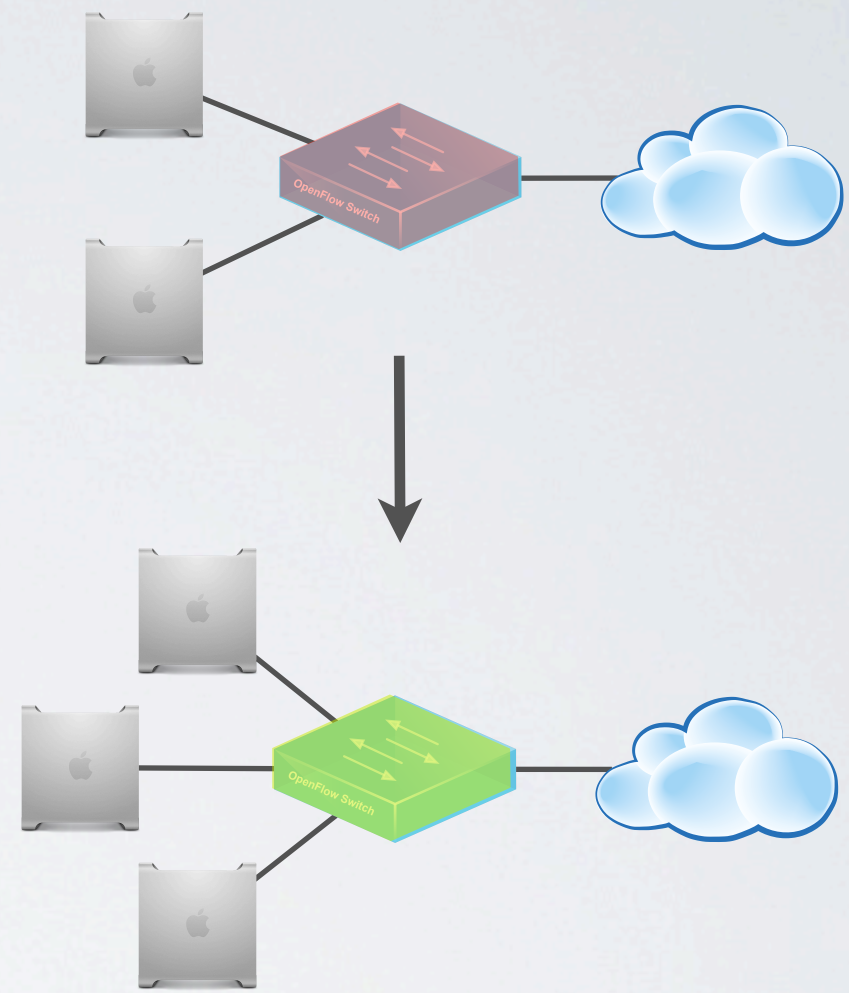
Every set of related packets processed with old or new configuration, but not a mixture of the two.

Main challenge

- Need to identify active flows

Implementation mechanisms

- Switch rules with timeouts
- Wildcard cloning
- End-host feedback



Ongoing Work

Other abstractions

- Loop-free updates
- Destination-preserving updates

Update Synthesis

- Programmer specifies an invariant
- Compiler generates an update that preserves it

Better responsiveness

- Act quickly when failures occur
- Monotonic updates?

Enhanced fault tolerance

- Use compiler to "harden" configurations
- Pre-load a backup

Leverage end hosts

- Help identify active flows
- Consistent source routing?

Thank You!

Collaborators

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

Funding

http://frenetic-lang.org