Abstractions for Network Update

Nate Foster
Mark Reitblatt

Jen Rexford
Cole Schlesinger
Dave Walker
Updates Happen

Network Updates
- Maintenance
- Failures
- ACL Updates
Network Updates Are Hard
Network Updates Are Hard
Prior Work

Consensus Routing

Reliable BGP

Graceful Migration

Seamless Migration
Prior Work

- Consensus Routing
- Reliable BGP
- Graceful Migration
- Seamless Migration
Network Update Abstractions

Goal
- Tools for whole network update

Our Approach
- Develop update abstractions
- Endow them with strong semantics
- Engineer efficient implementations
Example: Distributed Access Control

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Example: Distributed Access Control

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Traffic: I → F1, F2 → F3
- Web: Allow
- *: Allow
- F1, F2, F3 connected with arrows indicating traffic flow.
Example: Distributed Access Control

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Naive Update

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Order

F2
F3
I
Naive Update

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Order

Traffic
Naive Update

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Naive Update

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Order
Naive Update

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Order

Traffic
Use an Abstraction!

Security Policy

UPDATE

![Diagram of network with security policy](image-url)
Atomic Update?

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Traffic flow diagram:
- From F1, F2 to F1
- From F3 to F2
- From F2 to F3
- From F1 to F3

Traffic direction:
- From left to right
Atomic Update?

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Traffic
Atomic Update?

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Traffic:
- F1, F2: Web: ✓, *: ✗
- F3: Web: ✓, *: ✓

Diagram:
- F1 connects to F2 and F3.
- F2 connects to F1 and F3.
- F3 connects to F1 and F2.
Atomic Update?

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Traffic:

F1: Web: ✓, *: ×
F2: Web: ✓, *: ×
F3: Web: ✓, *: ✓
F1 → F2
F1 → F3
F2 → F3
Atomic Update?

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Per-Packet Consistent Updates

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

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>
Universal Property Preservation

**Theorem:** Per-packet consistent updates preserve all trace properties.

**Trace Property**
Any property of a *single* packet’s path through the network.

**Examples of Trace Properties:**
Loop freedom, access control, waypointing ...

**Trace Property Verification Tools:**
Anteater, Header Space Analysis, ConfigChecker ...
Formal Verification

**Corollary:** To check an invariant, verify the old and new configurations.

- **Security Policy** → **Analyzer** → ✓
- **Security Policy** → **Analyzer** → ✓

**Verification Tools**
- Anteater [SIGCOMM '11]
- Header Space Analysis [NSDI '12]
- ConfigChecker [ICNP '09]
MECHANISMS
2-Phase Update

Overview
- Runtime instruments configurations
- Edge rules stamp packets with version
- Forwarding rules match on version

Algorithm (2-Phase Update)
1. Install new rules on internal switches, leave old configuration in place
2. Install edge rules that stamp with the new version number
2-Phase Update in Action
2-Phase Update in Action
2-Phase Update in Action

Traffic

15
2-Phase Update in Action
2-Phase Update in Action

Traffic
Optimized Mechanisms

Optimizations
- Extension: strictly adds paths
- Retraction: strictly removes paths
- Subset: affects small # of paths
- Topological: affects small # of switches

Runtime
- Automatically optimizes
- Power of using abstraction
Subset Optimization
Subset Optimization
Correctness

**Question:** How do we convince ourselves these mechanisms are correct?

**Solution:** We built an operational semantics, formalized our mechanisms and proved them correct.

**Example:** 2-Phase Update

1. Install new rules on internal switches, leave old configuration in place  \} \text{ Unobservable}

2. Install edge rules that stamp with the new version number \} \text{ One-touch}
Correctness

Question: How do we convince ourselves these mechanisms are correct?

Solution: We built an operational semantics, formalized our mechanisms and proved them correct.

Example: 2-Phase Update

1. Install new rules on internal switches, leave old configuration in place

2. Install edge rules that stamp with the new version number

Unobservable

One-touch

Theorem: Unobservable + one-touch = per-packet.
IMPLEMENTATION
&
EVALUATION
Implementation

Runtime
- NOX Library
- OpenFlow 1.0
- 2.5k lines of Python
- update(config, topology)
- Uses VLAN tags for versions
- Automatically applies optimizations

Verification Tool
- Checks OpenFlow configurations
- CTL specification language
- Uses NuSMV model checker
Evaluation

**Question:** How much extra rule space is required?

**Setup**
- Mininet VM

**Applications**
- Routing and Multicast

**Scenarios**
- Adding/removing hosts
- Adding/removing links
- Both at the same time

**Topologies**
- Fattree
- Small-world
- Waxman
Results: Routing Application

- Full
- Subset

Fattree  
Small-world  
Waxman
Results: Routing Application

Worst-Case Rule Overhead

Fattree
Small-world
Waxman
Results: Routing Application

Worst-Case Rule Overhead

- Full
- Subset

<table>
<thead>
<tr>
<th></th>
<th>Host</th>
<th>Link</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fattree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small-world</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Waxman</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22
WRAP UP
Conclusion

Update abstractions
• Per-packet
• Per-flow

Mechanisms
• 2-Phase Update
• Optimizations

Implementation
• Runtime
• Verifier

Formal model
• Network operational semantics
• Universal property preservation
Thank You!

Collaborators
Shrutarshi Basu (Cornell)
Arjun Guha (Cornell)
Stephen Gutz (Cornell)
Rob Harrison (West Point)
Nanxi Kang (Princeton)
Naga Praveen Katta (Princeton)
Chris Monsanto (Princeton)
Josh Reich (Princeton)
**Cole Schlesinger** (Princeton)
Robert Soulé (Cornell)
Alec Story (Cornell)
**Nate Foster** (Cornell)
Mike Freedman (Princeton)
**Jen Rexford** (Princeton)
Emin Gün Sirer (Cornell)
**Dave Walker** (Princeton)
BACKUP SLIDES
Beyond Per-Packet

Per-flow consistent update
Each set of related packets processed with old or new configuration, but not a mixture of the two.

Use Cases
- Load balancer
- Flow affinity
- In-order delivery

Mechanism
- 2-Phase Update + “flow tracking”