SDN Control Abstractions

Brighten Godfrey
CS 538 March 8, 2017
Outline

Frenetic

Network updates

Beyond the research
Network Updates

Slides Courtesy
Nate Foster!
Abstractions for Network Update

Nate Foster
Mark Reitblatt
Jen Rexford
Cole Schlesinger
Dave Walker
Updates Happen

Network Updates
• Maintenance
• Failures
• ACL Updates

Desired Invariants
• No black-holes
• No loops
• No security violations
Network Updates Are Hard
Prior Work

Consensus
Routing

Graceful
Migration

Seamless
Migration

Reliable BGP

Seamless Network-Wide IGP Migrations

Laurent Vanbever, Cristel Pelsser, Olivier Bonaventure, Stefano Vissicchio

Network-wide IGP migrations are among the most complex IGP transitions. Migrating an IGP is a complex procedure that requires prior specific knowledge of the IGP. Migrating an IGP can be problematic. Indeed, an IGP needs to learn or announce the prefixes by introducing or removing announcements. Route summarization also helps improving the stability by limiting the visibility of local events. Actually, some IGP migrations combine several migration scenarios, including the addition or the replacement of an IGP. It is then important to understand what is happening during a migration since all the routers have to be reconfigured in a proper manner. Simple solutions like restarting the network with another IGP may lack appropriate tools and techniques to seamlessly migrate a network-wide IGP. We illustrate the benefits of our methodology by considering several migration scenarios, including the addition or the replacement of a routing protocol or the modification of an IGP in an AS.

Categories and Subject Descriptors:
C.2.3 [Computer-Communications Networks]: Reliability

General Terms:
Design, Experimentation, Reliability

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.
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

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:
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

Order
- F1
- F2
- F3
- I

Traffic
Use an Abstraction!

Security Policy

UPDATE
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>

![Diagram showing traffic flow and security policy with a table listing security rules]

- **Traffic**
  - Web: ✓, *: ✗
  - Non-web: ✗, *: ✓
  - Any: ✓, *: ✓
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></td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td></td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td></td>
<td>Any</td>
<td>Allow</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.

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

Traffic

F1

F2

F3

F1, Web: ✓
F1, *: X

F2, Web: ✓
F2, *: X

F3, *: ✓
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 Diagram]

The diagram illustrates a network with switches labeled F1, F2, and F3, connected by arrows indicating traffic flow. The network includes web traffic, represented by symbols and checks, indicating optimization or selection criteria. The traffic paths are indicated by arrows, showing how data flows through the network system.
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

**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
**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

- Fattree
- Small-world
- Waxman

Worst-Case Rule Overhead

- Host
- Link
- Both

Full
Subset

Fattree
Small-world
Waxman
Beyond today’s research

Industry policy languages, e.g. OpenStack Congress
[https://wiki.openstack.org/wiki/Congress]

• “App A is only allowed to communicate with app B.”
• “Virtual machine owned by tenant A should always have a public network connection if tenant A is part of the group B.”
• “Virtual machine A should never be provisioned in a different geographic region than storage B.”

High-level abstractions one of the big open questions for SDN

• What can people use? Who is doing the programming?
• What’s the killer app?
• Does different hardware change the abstraction?