A Network-State Management Service

Peng Sun
Ratul Mahajan, Jennifer Rexford,
Lihua Yuan, Ming Zhang, Ahsan Arefin

Princeton & Microsoft
Complex Infrastructure

Microsoft Azure

<table>
<thead>
<tr>
<th>Number of</th>
<th>2010</th>
<th>2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Center</td>
<td>A few</td>
<td>10s</td>
</tr>
<tr>
<td>Network Device</td>
<td>1,000s</td>
<td>10s of 1,000s</td>
</tr>
<tr>
<td>Network Capacity</td>
<td>10s of Tbps</td>
<td>Pbps</td>
</tr>
</tbody>
</table>

Variety of vendors/models/models/time
Management Applications

- Traffic Engineering
- Load Balancing
- Link Corruption Mitigation
- Device Firmware Upgrade
Our Question

How to safely run *multiple* management applications on *shared* infrastructure
Naïve Solution

- Run independently

- Traffic Engineering
- Link Corruption Mitigation
- Firmware Upgrade

Network Devices
Naïve Solution

- It does not work due to 2 problems

Traffic Engineering

Link Corruption Mitigation

Firmware Upgrade

Network Devices
Problem #2: Safety Violation

Link-corruption-mitigation shuts down faulty Agg A

Firmware-upgrade schedules Agg B to upgrade
Potential Solution #1

- One monolithic application
- Central control of all actions
Too Complex to Build

- Difficult to develop
  - Combine all applications that are already individually complicated
- High maintenance cost
  - for such huge software in practice
Potential Solution #2

• Explicit coordination among applications
• Consensus over network changes
Still Too Complex

- Hard to understand each other
- Diverse network interactions

<table>
<thead>
<tr>
<th>Application</th>
<th>Routing</th>
<th>Device Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Engineering</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Firmware upgrade</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Main Enemy: Complexity

- Application development
- Application coordination

Simple

Independ- dent

Explicitly coordinate

Monolithic

Complex
What We Advocate

• Loose coupling of applications
• Design principle:
  • Simplicity with safety guarantees
• Forgo joint optimization
  • Worthwhile tradeoff for simplicity
  • Applications could do it out-of-band
Overview of Statesman

• Network operating system for safe multi-application operation

• Uses network state abstraction
  • Three views of network state
  • Dependency model of states
The “State” in Statesman

- Complexity of dealing with devices
- Heterogeneity
- Device-specific commands
## State Variable Examples

<table>
<thead>
<tr>
<th>State Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Power Status</td>
<td>Up, down</td>
</tr>
<tr>
<td>Device Firmware</td>
<td>Version number</td>
</tr>
<tr>
<td>Device SDN Agent Boot</td>
<td>Up, down</td>
</tr>
<tr>
<td>Device Routing State</td>
<td>Routing rules</td>
</tr>
<tr>
<td>Link Admin Status</td>
<td>Up, down</td>
</tr>
<tr>
<td>Link Control Plane</td>
<td>BGP, OpenFlow, …</td>
</tr>
</tbody>
</table>
Simplify Device Interaction

Past

Application

Device Statistics

SNMP, OF, vendor API, ...

Device-specific cmds

Network Devices

Now

Application

Read

Network State

Write

Network Devices
### Views of Network State

<table>
<thead>
<tr>
<th>Observed State</th>
<th>Actual state of the whole network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target State</td>
<td>Desired state to be updated on the whole network</td>
</tr>
</tbody>
</table>

- **Observed State**: The actual state of the whole network.
- **Target State**: The desired state to be updated on the whole network.
Two Views Are Not Enough

Observed State

Target State

Network Devices

Application
Two Views Are Not Enough

One More View

Proposed State: A group of entity-variable-values desired by an application

Observed State

Proposed State

Target State

Application

Network Devices
How Merging Works

• Combine multiple proposed states into a safe target state

• Conflict resolution
  • Last-writer-wins
  • Priority-based locking
  • Sufficient for current deployment

• Safety invariant checking
  • Partial rejection & Skip update
Choose Safety Invariants

- **Loose**
  - Cannot protect network operation

- **Tight**
  - Hinder application too frequently

- Our current choice
  - Connectivity: Every pair of ToRs in one DC is connected
  - Capacity: 99% of ToR pairs have at least 50% capacity
Recap of Three-View Model

- Simplify network management

**Observed State**
What we see from the network

**Proposed State**
What we want the network to be

**Target State**
What can be actually done on the network

**Application**
Statesman
Yet Another Problem

• What’s in Proposed State
  • Small number of state variables that application cares

• Implicit conflicts arises
  • Caused by state dependency
Implicit Conflict

TE writes new value of routing state of B for tunneling traffic

Firmware-upgrade writes new value of firmware state of B
Dependency Relations

- PathState
- RoutingState
- ConfigurationState
- FirmwareVersion
- PowerState

ConfigurationState

AdminState

Device

Link
Build in Dependency Model

- Statesman calculates it internally
- Only exposes the result for each state variable
  - Whether the variable is controllable
Statesman System

Storage Service

Observed State

Proposed State

Target State

Monitor

Updater

Checker
Deployment Overview

- Operational in Microsoft Azure for 10 months
- Cover 10 DCs of 20K devices
Production Applications

• 3 diverse applications built
  • Device firmware upgrade
  • Link corruption mitigation
  • Traffic engineering

• Finish within months

• Only thousands of lines of code
Case #1: Resolve Conflict

Inter-DC TE & Firmware-upgrade

DC = Data Center
BR = Border Router
Firmware-upgrade acquires lock of BR1
TE fails to acquire lock, and moves traffic away.
TE fails to acquire lock, and moves traffic away
BR1 firmware upgrade starts
BR1 firmware upgrade starts

BR1 firmware upgrade ends. Lock released.
BR1 firmware upgrade starts

TE re-acquires lock, and moves traffic back
BR1 firmware upgrade starts

TE re-acquires lock, and moves traffic back
Case #1 Summary

• Each application:
  • Simple logic
  • Unaware of the other

• Statesman enables:
  • Conflict resolution
  • Necessary coordination
Case #2: Maintain Capacity Invariant

Firmware-upgrade & Link-corruption-mitigation

Link corrupting packets
Upgrade proceeds in normal speed in Pod 3 and 5.
Upgrade proceeds in normal speed in Pod 3 and 5.
Upgrade proceeds in normal speed in Pod 3 and 5
Upgrade proceeds in normal speed in Pod 3 and 5.

Upgrade in Pod 4 is slowed down by checker due to lost capacity.
Upgrade proceeds in normal speed in Pod 3 and 5

Upgrade in Pod 4 is slowed down by checker due to lost capacity
Case #2 Summary

- Statesman:
  - Automatically adjusts application progresses
  - Keeps the network within safety requirements
Conclusion

• Need network operating system for multiple management applications

• Statesman
  • Loose coupling of applications
  • Network state abstraction

• Deployed and operational in Azure
Thanks!

Questions?

Check paper for related works