A Network-State Management Service

Hari Raghavendar Rao Bandari
Sameer Kulkarni

[A] Peng Sun, Ratul Mahajan, and Jennifer Rexford, "A Network-State Management Service," in ACM SIGCOMM, 2014

Agenda

- Introduce the Network State Management.
- Go over the key findings of the paper in terms of:
  - Motivation and Problem Statement
  - Objectives and Concerns identified
  - Proposed solutions and Results
  - Limitations observed in this papers.

Introduction

“How to safely run Multiple Network management application on shared infrastructure”

Traffic Engineering

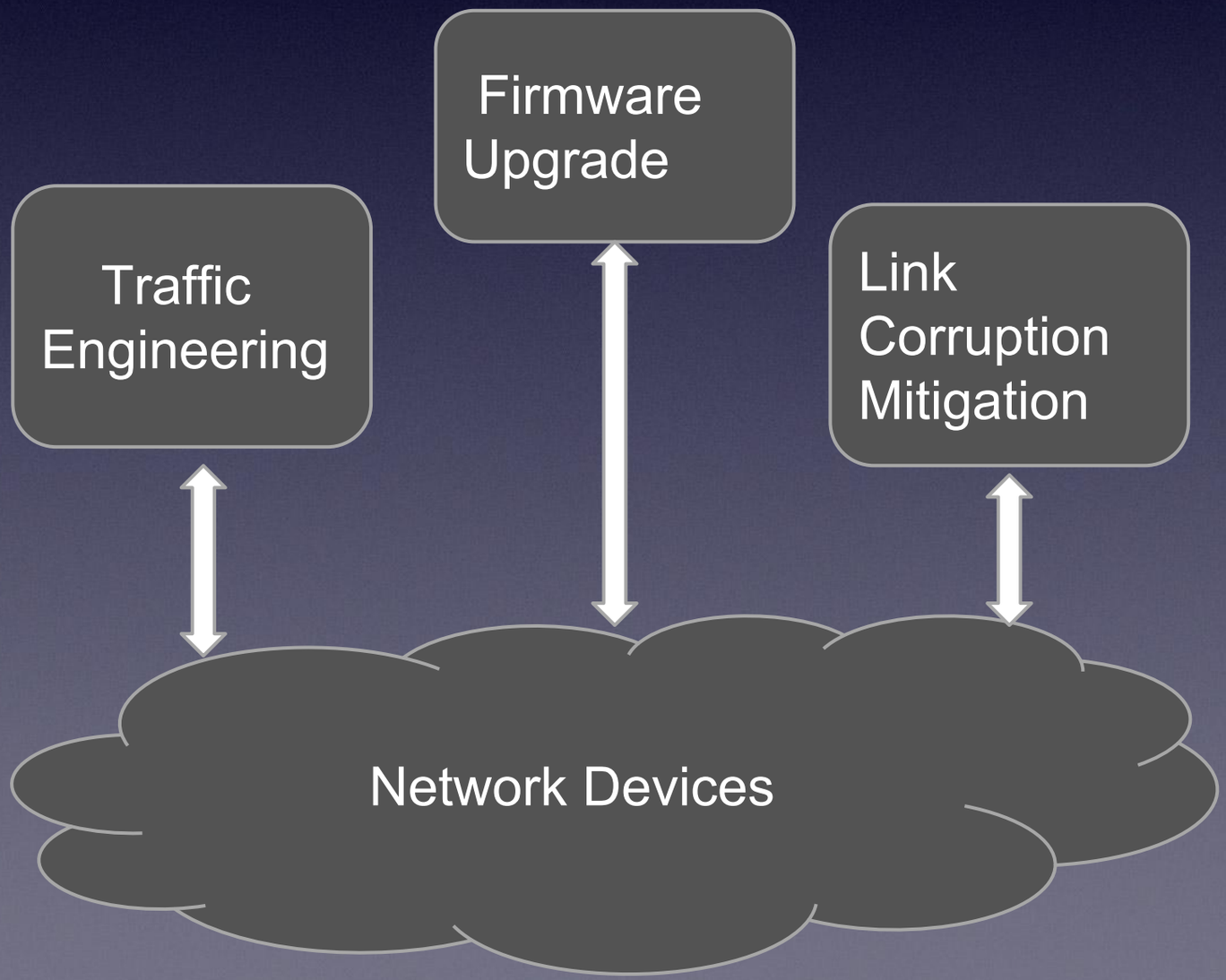
Load balancing

Link Corruption Mitigation

Device Firmware Update

Motivation and Problem statement:

“Running network management application independently ” but it has “two problems “



Problems :

Figure 1: Example of an application conflict

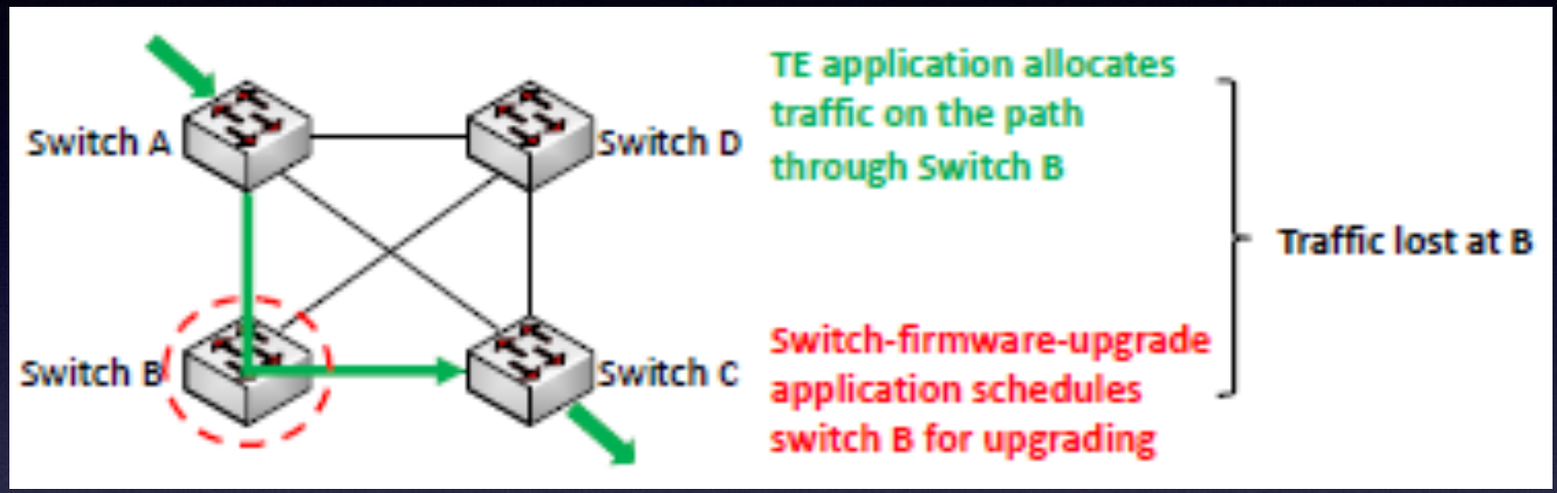
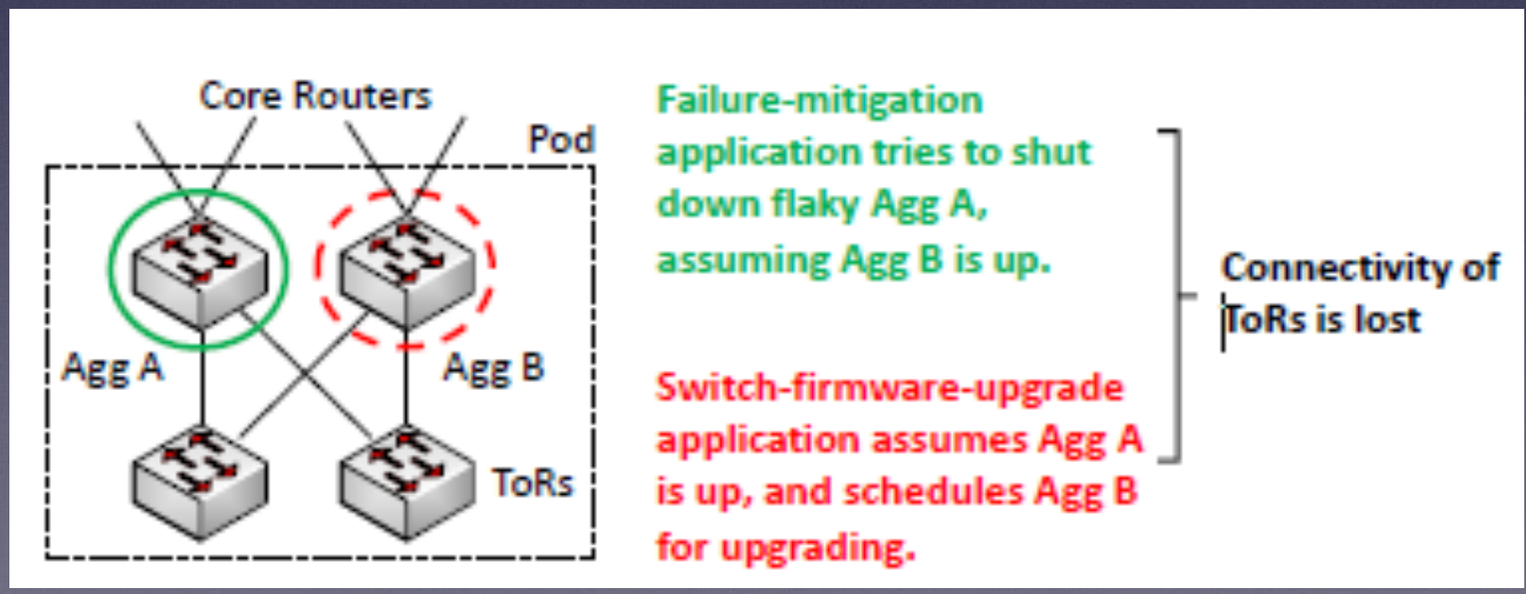


Figure 2: Example of a safety violation



Statesman Model

Three View Model

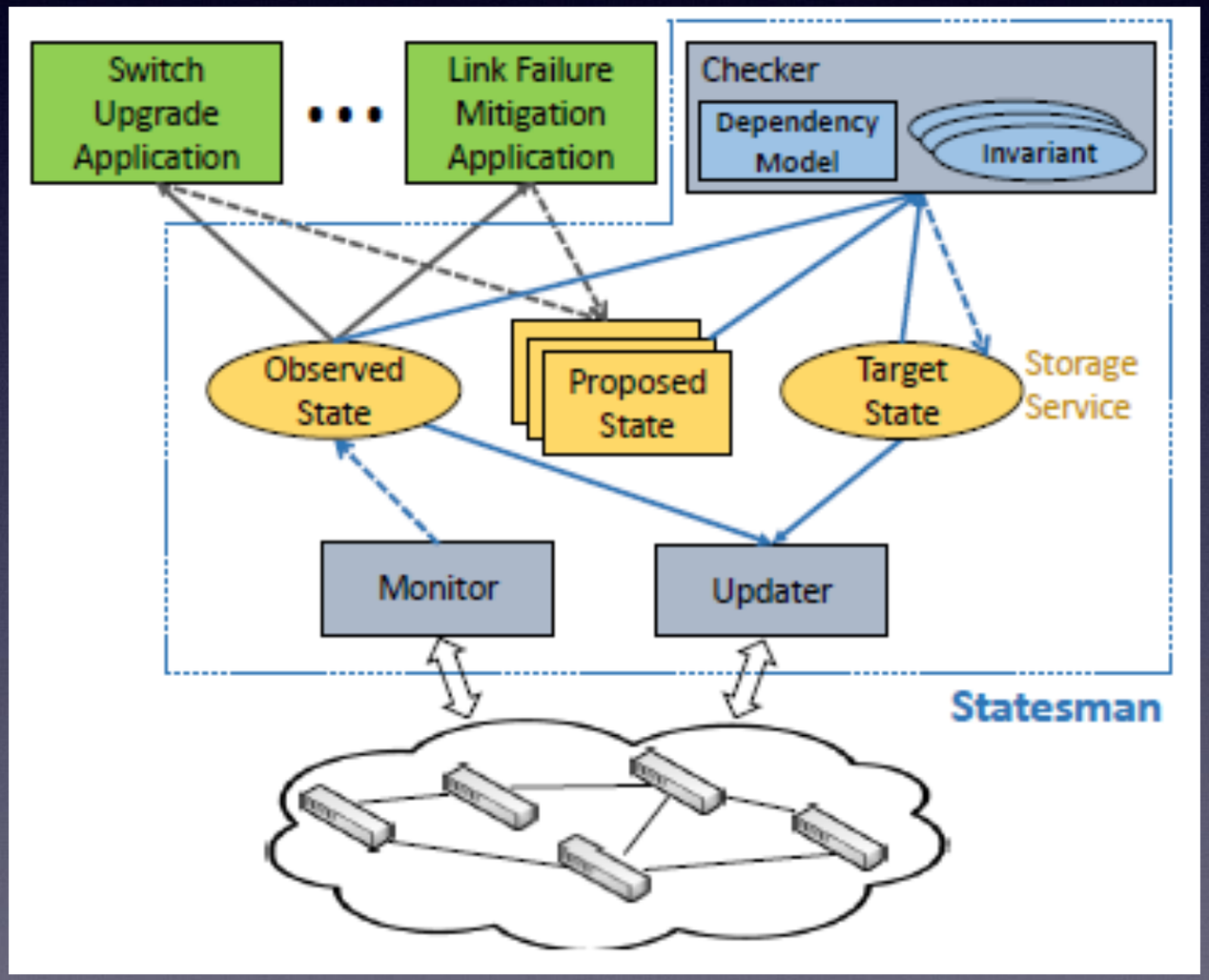


Table 1: Input and output of each component in Statesman

Component	Input	Output
Monitor	Switch/link data	OS
Checker	OS PSes TS	TS
Updater	OS TS	Switch update commands

Managing Network State

Figure 4: Network state dependency model

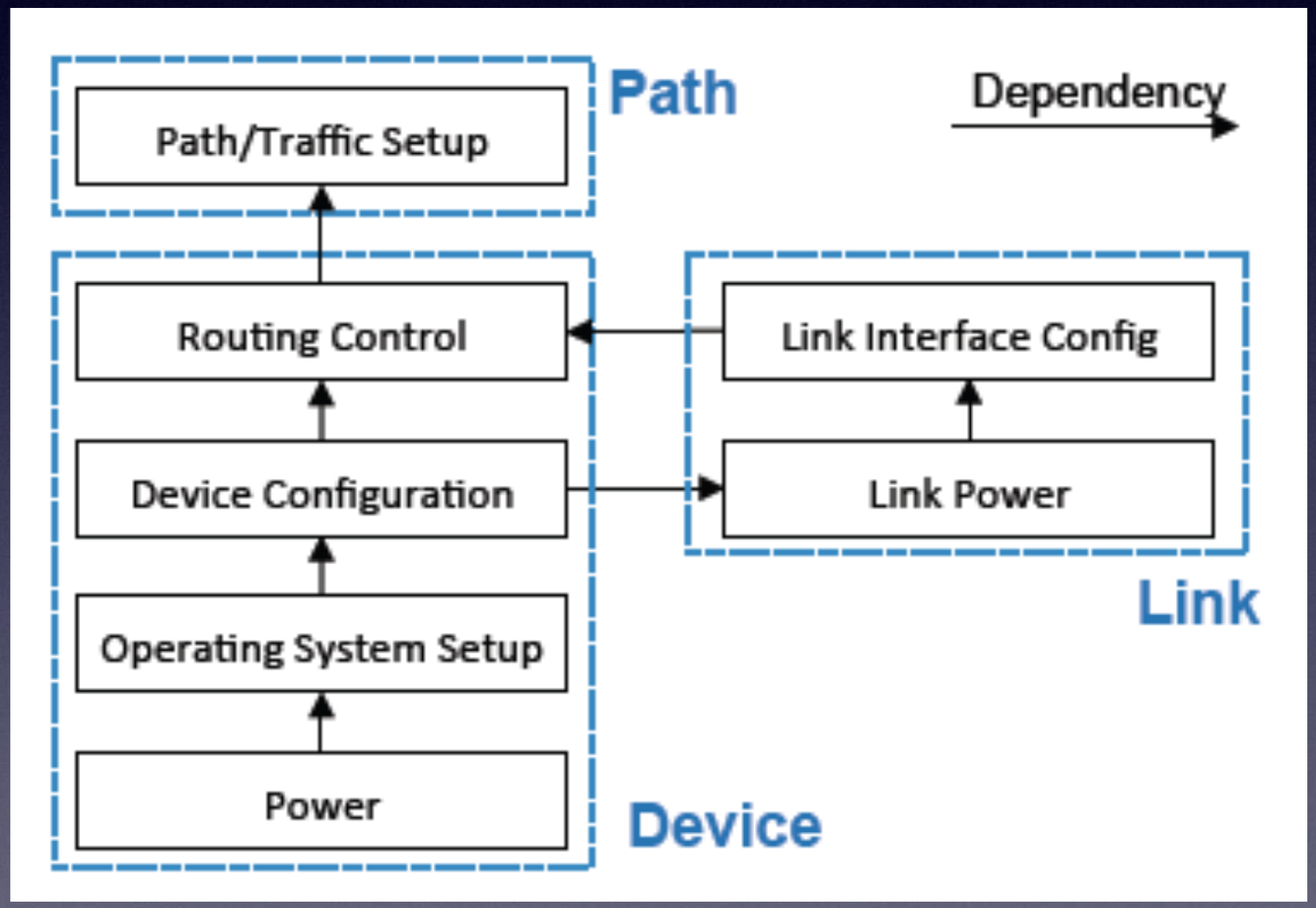
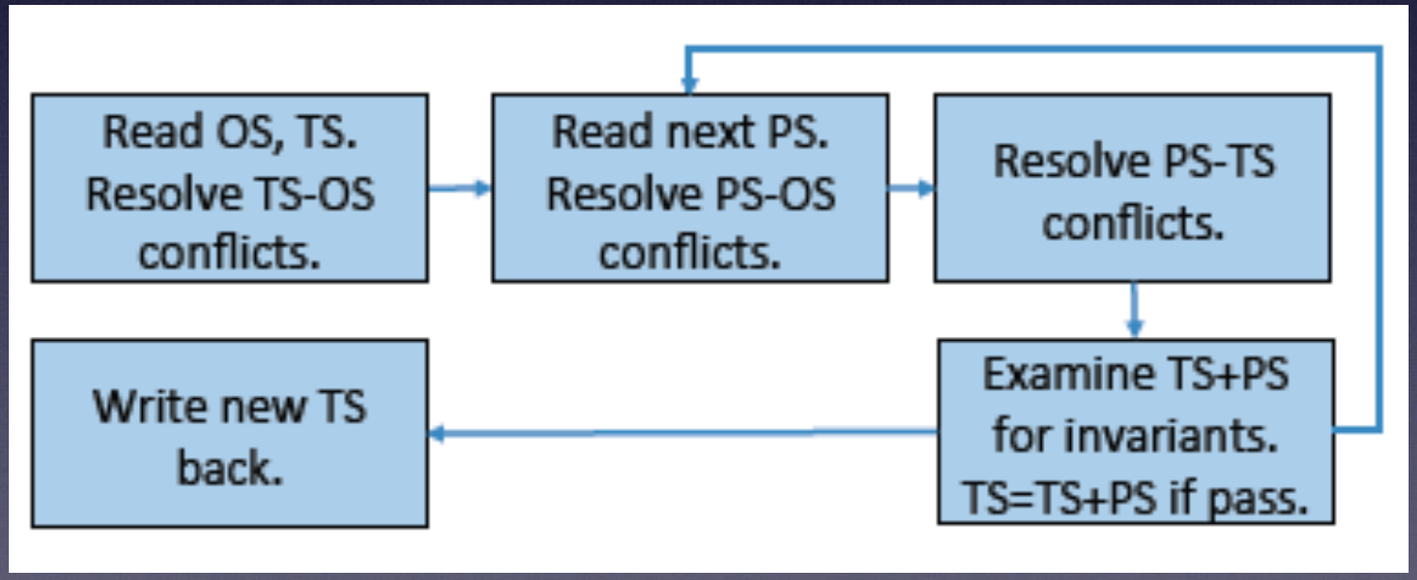


Table 2: Example network state variables

Entity	Level in dependency	Example state variables	Permission
Path	Path/traffic setup	Switches on path MPLS or VLAN config	ReadWrite ReadWrite
Link	Link interface config	IP assignment Control plane setup	ReadWrite ReadWrite
	Link power	Interface admin status Interface oper status	ReadWrite ReadOnly
	N/A (counters)	Traffic load Packet drop rate	ReadOnly ReadOnly
Device	Routing control	Flow-link routing rules Link weight allocation	ReadWrite ReadWrite
	Device configuration	Mgmt. interface setup OpenFlow agent status	ReadWrite ReadWrite
	Operating system setup	Firmware version Boot image	ReadWrite ReadWrite
	Power	Admin power status Power unit reachability	ReadWrite ReadOnly
	N/A (counters)	CPU utilization Memory utilization	ReadOnly ReadOnly

Checking Network State

Figure 5: Flow of the checker’s operation



How Merging Works

- Combine multiple proposed states into a safe target state

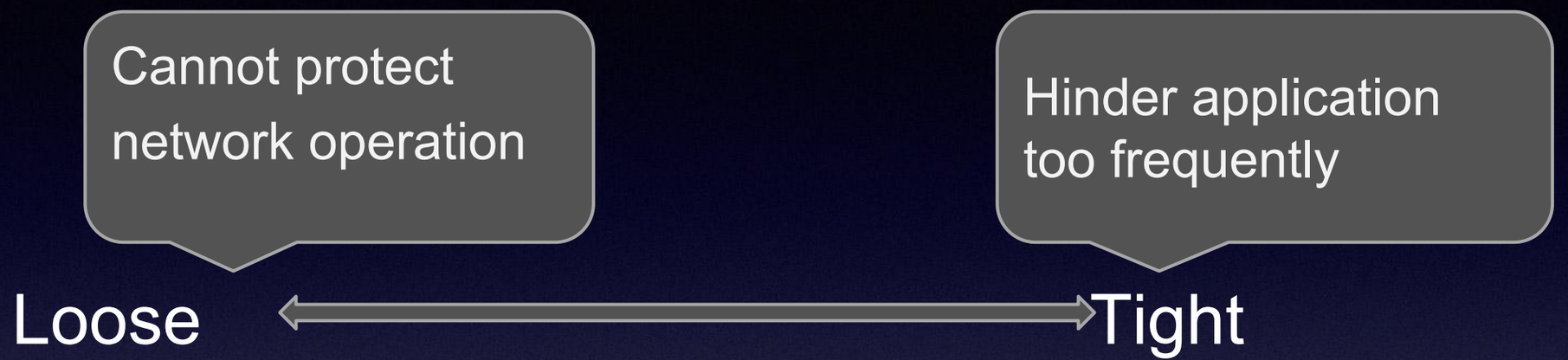
Conflict resolution

- Last-writer-wins
- Priority-based locking
- *Sufficient for current deployment*

Safety invariant checking

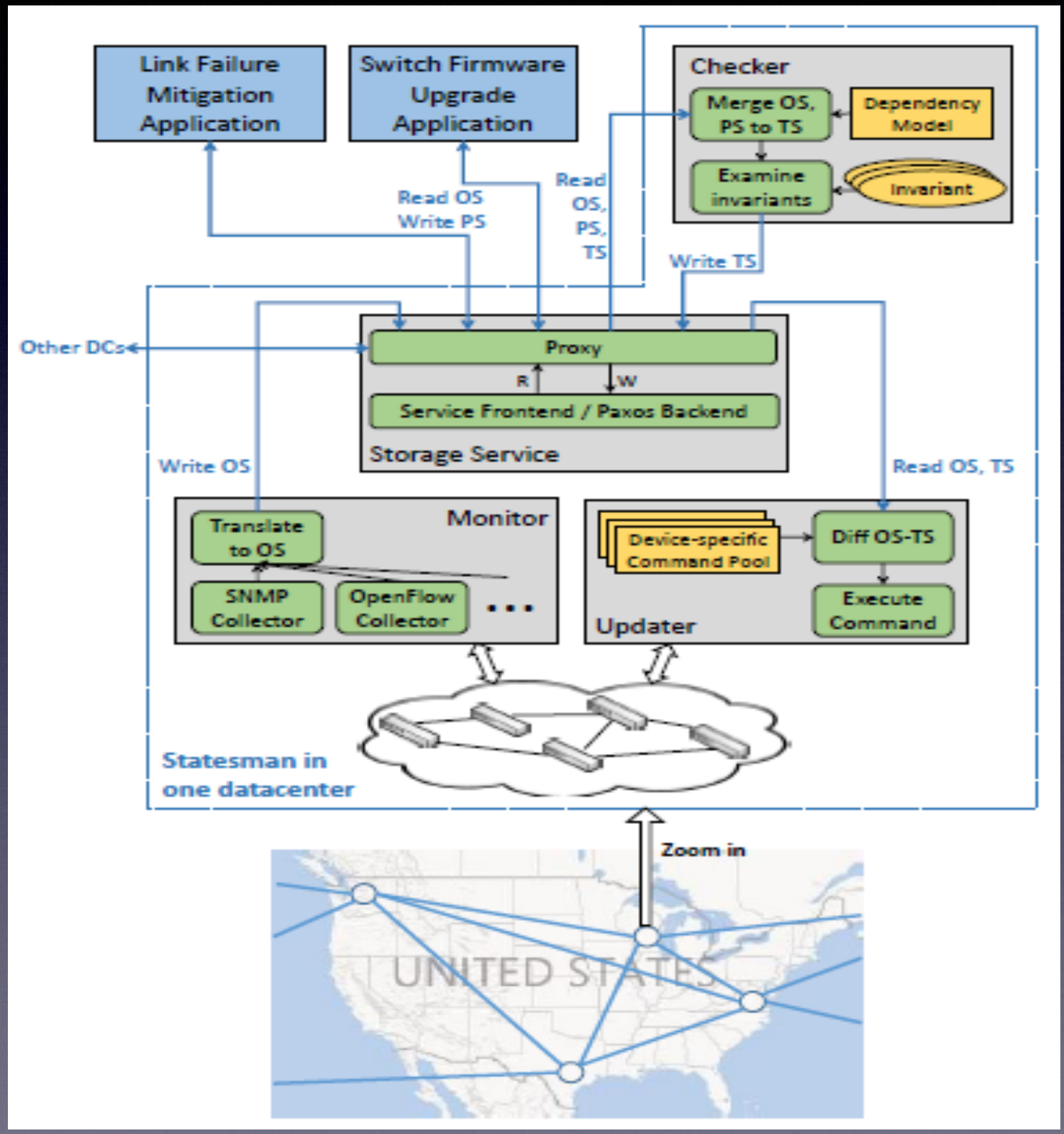
- Partial rejection & Skip update

Choose Safety Invariants



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

System Design and Implementation



Read-Write APIs

Table 3: Read-write APIs of Statesman

GET	NetworkState/Read?Datacenter={*dc*}&Pool={*p*} &Freshness={*c*}&Entity={*e*}&Attribute={*a*}
POST	NetworkState/Write?Pool={*p*} (Body is list of NetworkState objects in JSON)

(a) HTTP Request

Datacenter	*dc*	Datacenter name
Pool	*p*	OS, PS, or TS
Freshness	*c*	Up-to-date or bounded-stale
Entity	*e*	Entity name (i.e., switch, link, or path)
Attribute	*a*	State variable name

(b) Parameters

Application Experiences

7.2 Maintaining Network-wide Invariants

Figure 7: Network topology for the scenario in §7.2

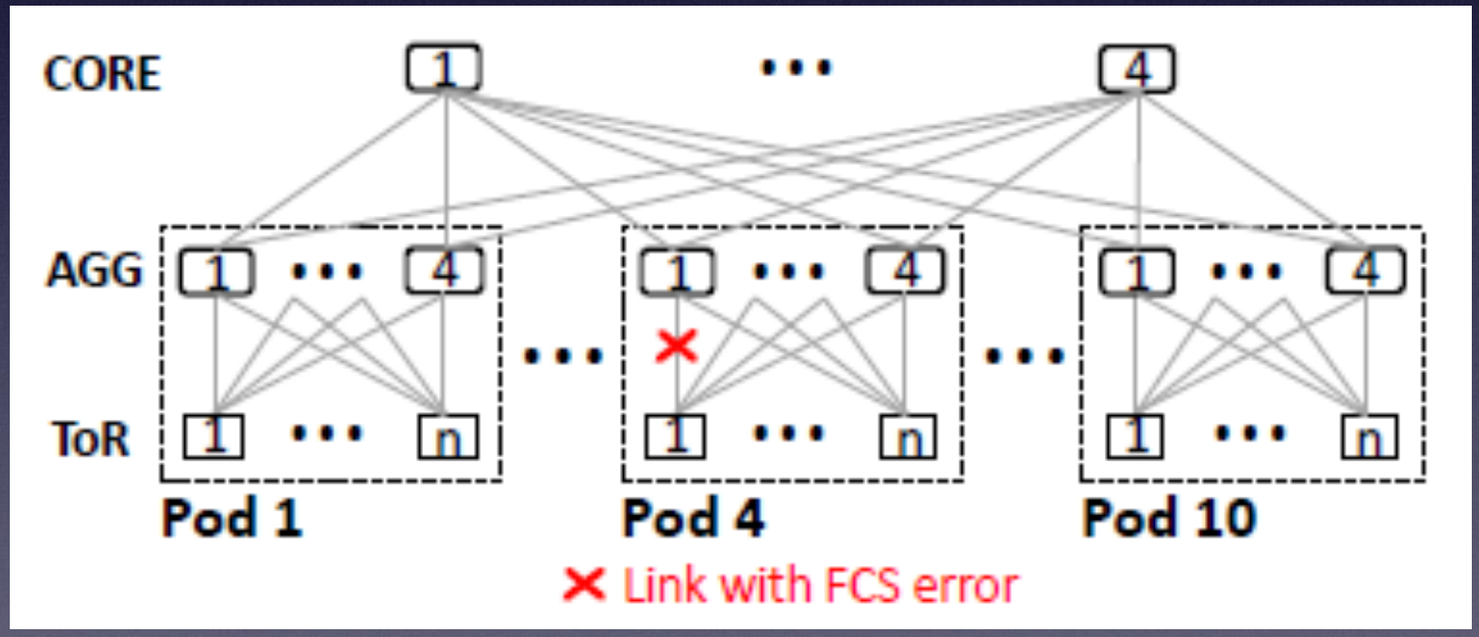
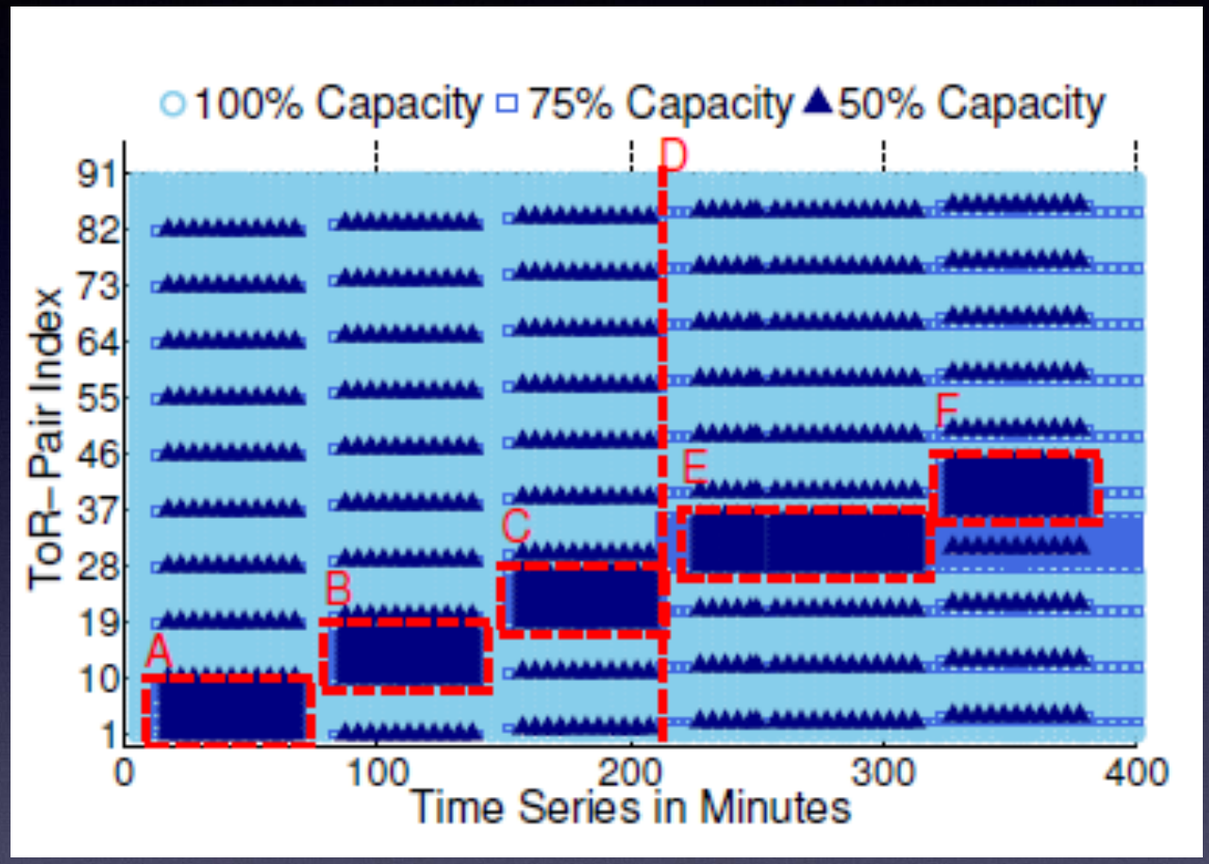
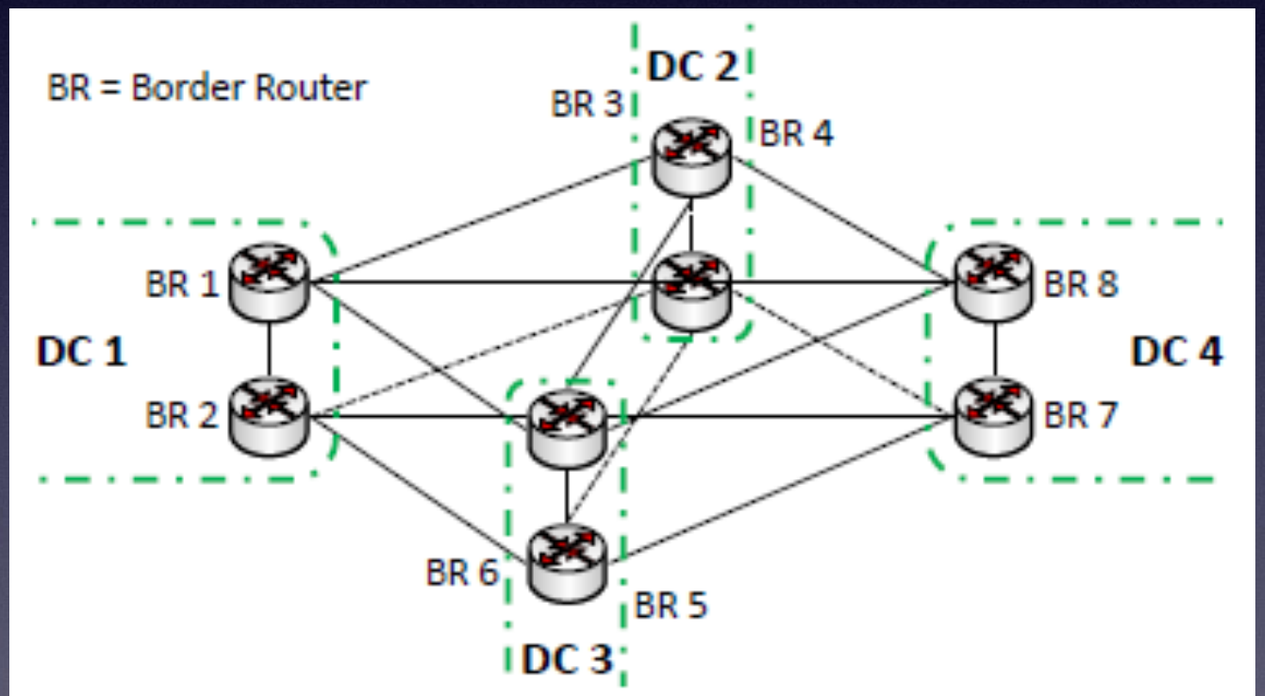


Figure 8: Illustration of how the switch-upgrade and failure mitigation applications interact through the checker. Y-axis: One ToR from each one of 10 pods to form directional ToR pairs, indexed by the originating ToR/Pod. A, B, C: The switch upgrade application upgrades Pod 1, 2, 3 normally while the checker maintains an invariant that each ToR pair has at least 50% of baseline capacity. D: The failure-mitigation application detects a failing link ToR1-Agg1 in Pod 4 and shuts the link down. E: Due to the down link, upgrade of Pod 4 is automatically slowed down by the checker to maintain the capacity invariant. F: Upgrade of Pod 5 resumes the normal pattern.



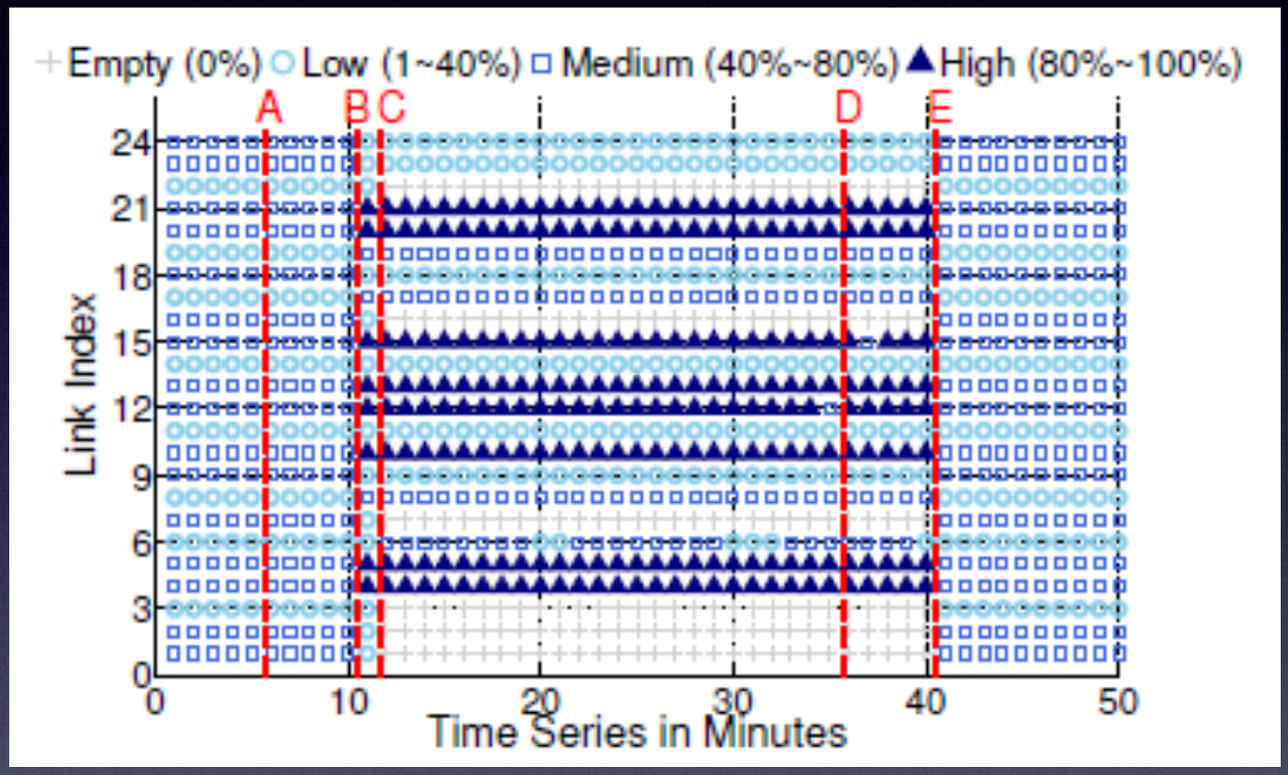
7.3 Resolving Application Conflicts

Figure 9: WAN topology for the scenario in §7.3



7.3 Resolving Application Conflicts

Figure 10: Illustration of how Statesman resolves conflicts between the inter-DC TE and switch-upgrade applications. A: The switch-upgrade application acquires the high-priority lock of BorderRouter1 (BR1). B: The TE application fails to acquire the low-priority lock and moves traffic away from BR1 to other links. C: The switch-upgrade application starts to upgrade BR1 since traffic is zero. D: Upgrade of BR1 is done. The switch-upgrade application releases the high-priority lock. E: The TE application re-acquires the low-priority lock of BR1 and moves traffic back.



7.4 Handling Operational Failures

Figure 11: Time series of firmware upgrade at scale

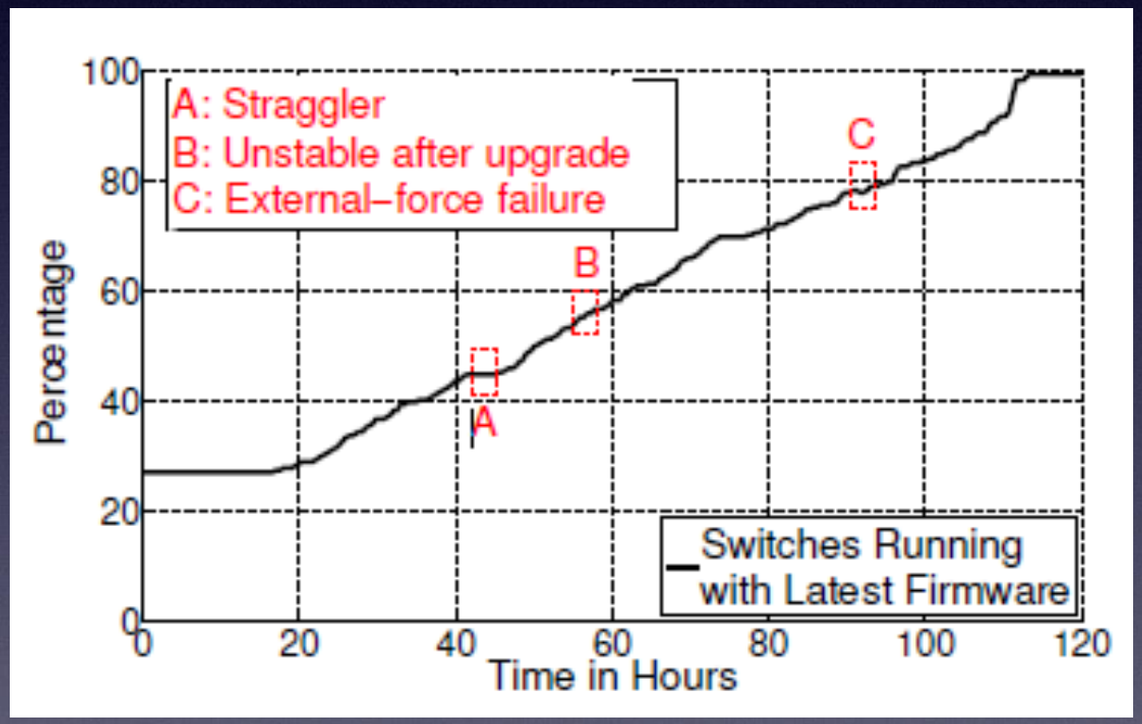
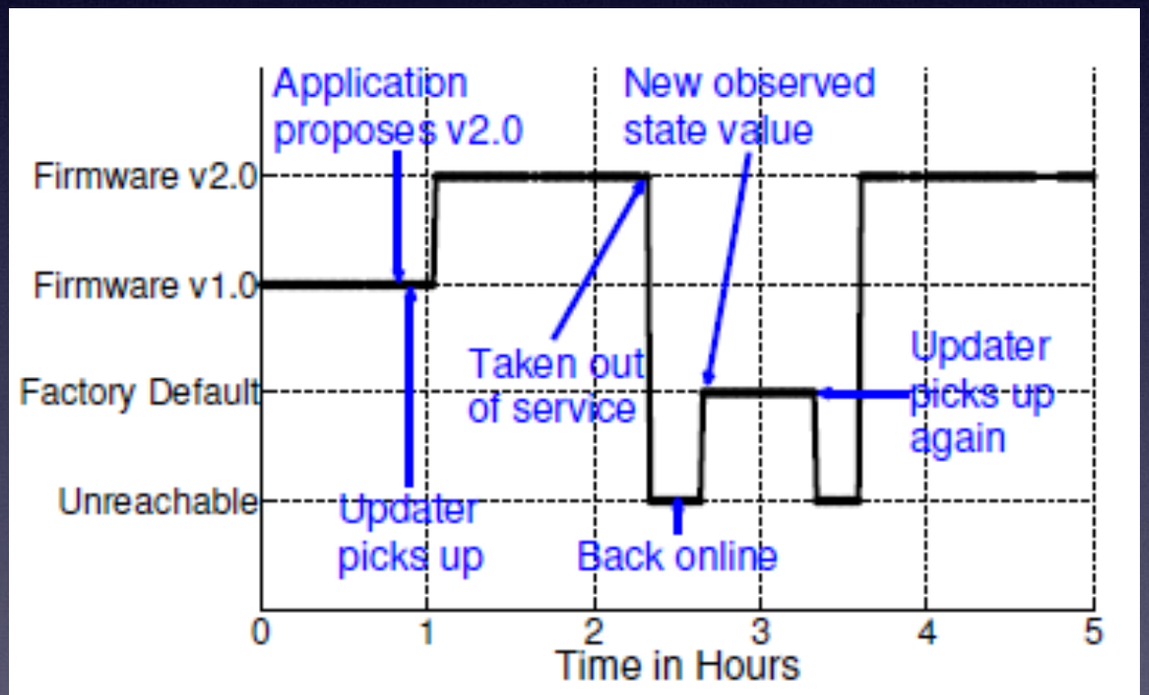
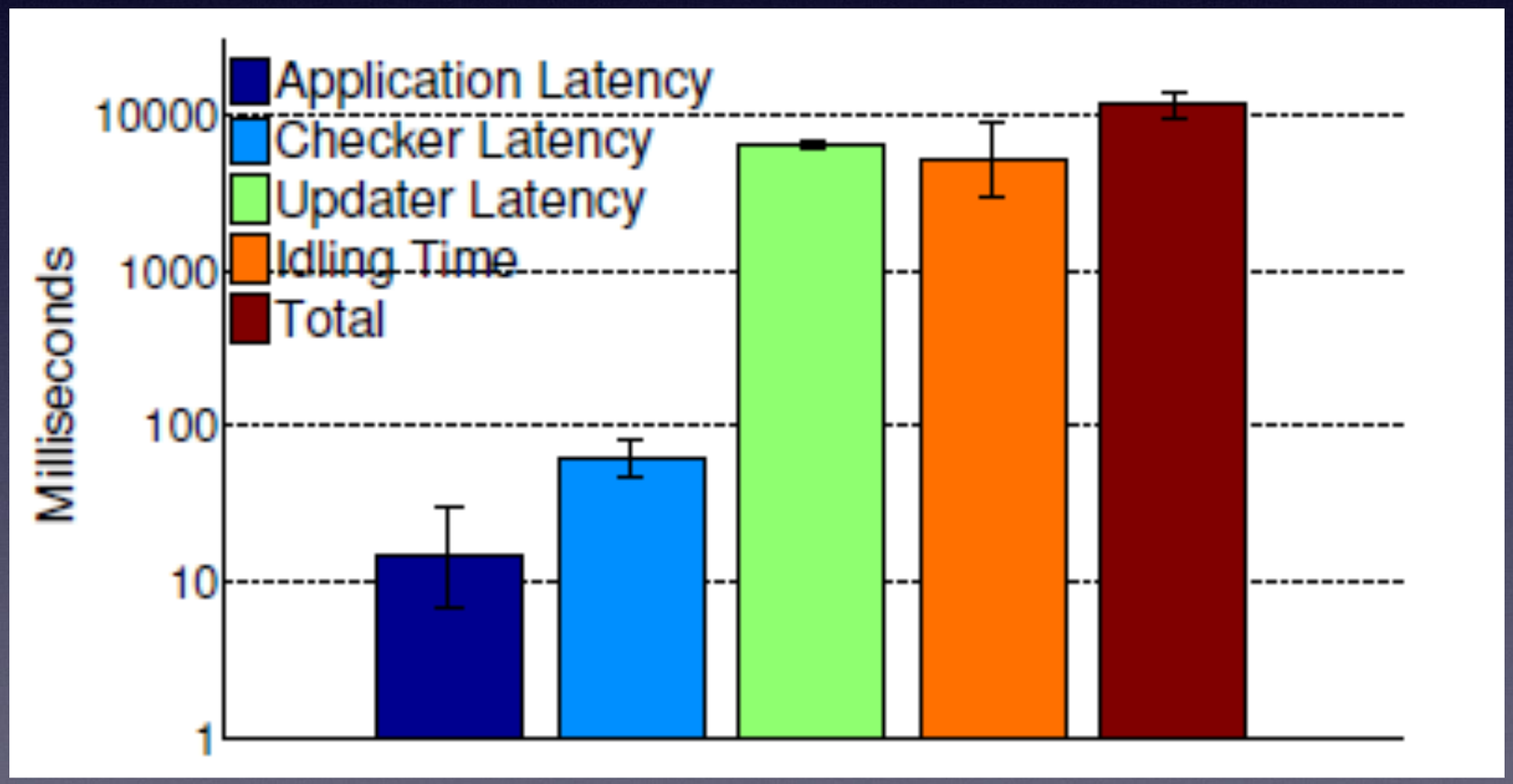


Figure 12: External-force failure in firmware upgrade



System Evaluation

Figure 13: End-to-end latency breakdown



System Evaluation

Figure 14: Network state scale & checker performance

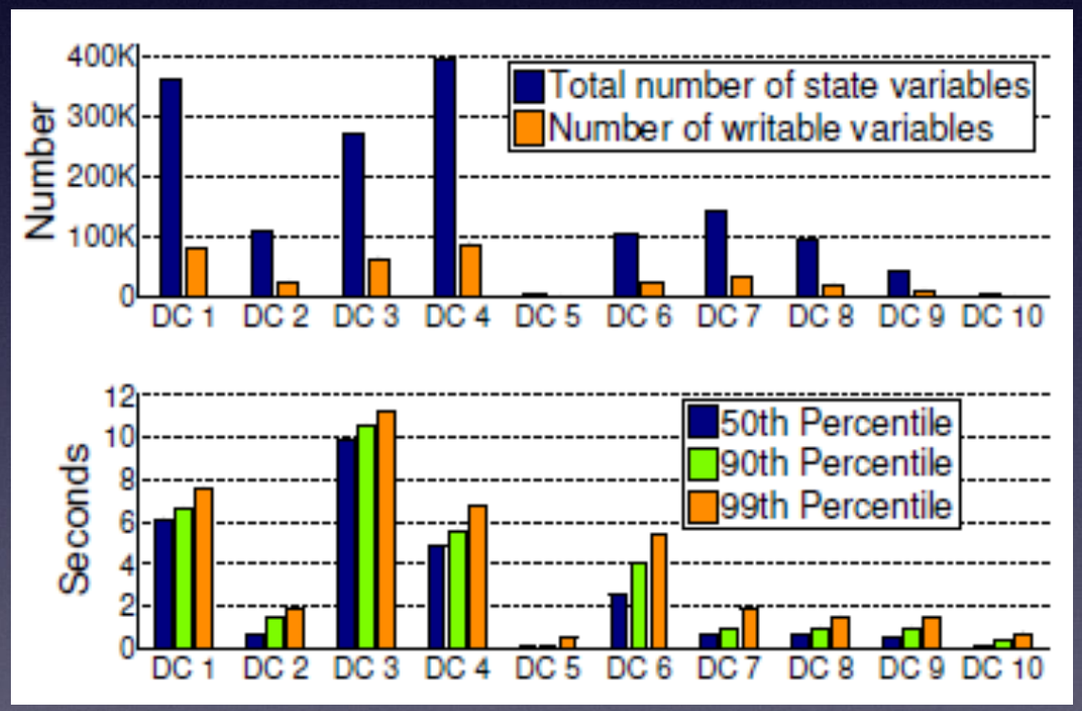
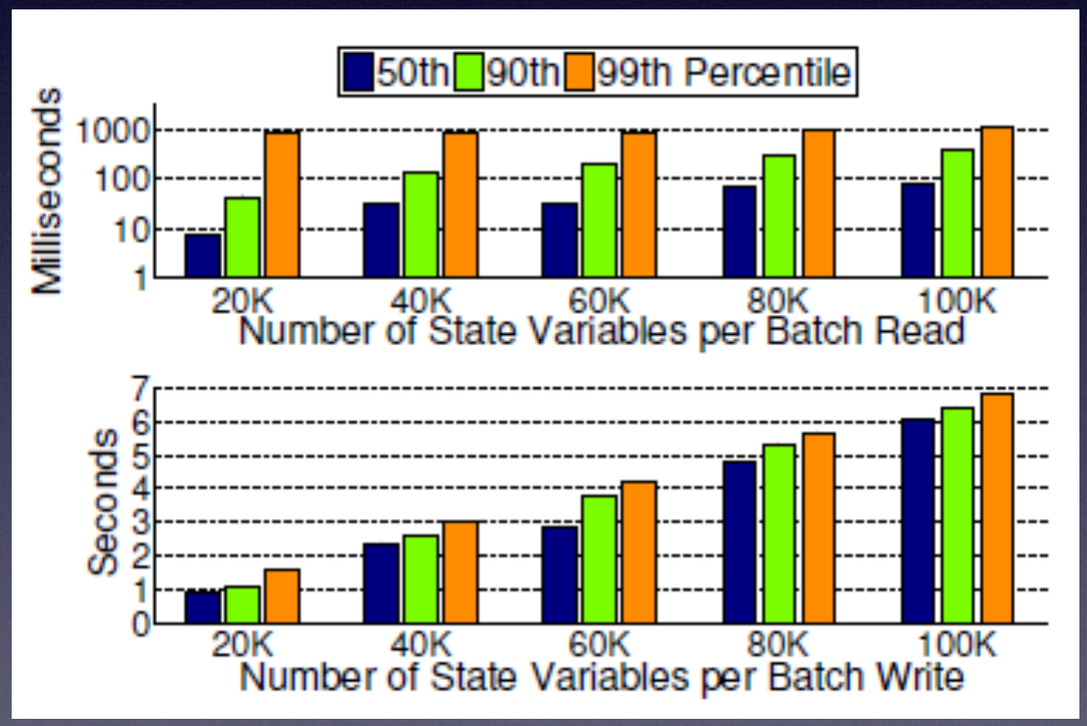


Figure 15: Read-write micro-benchmark performance



Limitations:

- Software Implementation and Deployment Overheads.
- Cost/effort for building applications that use the statesman
  - Statesmen Upgrade Overheads to suit new applications
  - Interface level details for applications to use statesman
- Invariant set is very limited and the specifics of adding/addressing them is not detailed.
- Asynchronous components are seen to yield lot of idling time.

- Thank You.
- Questions?