Study the past if you would define the future*: Secure Multi-Party SDN Updates

Liron Schiff  
Tel Aviv University, Israel

Stefan Schmid  
Aalborg University, Denmark  
TU Berlin, Germany

*Confucius
Routing Network

• A distributed process
Routing Network

• With distributed control
Routing Network

• Example: shortest paths
Routing Network

• Example 2: failure recovery
Routing Network

• But: expensive, not flexible...
Software Defined Network (SDN)

• Centralized control
Software Defined Network (SDN)

• Centralized control
Software Defined Network (SDN)

Centralized control

Control plane network

Fast data plane processing
Software Defined Network (SDN)

- Software frameworks
- Open standards
Software Defined Network (SDN)

- Events are sent to controller
  - Link failures
  - Unhandled packets
  - etc...

Software Defined Network (SDN)

- Controller sends commands
  - Forwarding rule updates
  - Statistics requests
  - etc...

commands
Software Defined Network (SDN)

- One point of failure:
Software Defined Network (SDN)

- Logically centralized, physically distributed
Distributed Control Security Issues

• The control network can be compromised
  – Advanced Persistent Threat (APT)
  – Insider

• Some controllers might be more vulnerable
  – Physical isolation
  – Logical isolation (policies)
  – Control applications
  – Admins

• We assume most controllers are secured
Distributed Control Security Issues

• A compromised control network:

• (we focus on one device)
Distributed Control Security Issues

- Malicious commands attack
Distributed Control Security Issues

• Solution: Sign commands with threshold cryptography

Fleet [S. Matsumoto, S. Hitz, A. Perrig 2014]
Distributed Control Security Issues

- Controller state corruption: deletion or injection of events

**Diagram:**
- Controller additions?
- Reboots?
- Bad cmds
- Remote controller
- Events
Theoretical Approach

• Use a distributed shared log algorithm (e.g., Paxos)

• Each entity (device / controller):
  – Suggests log entry (event / command)
  – Considers previous entries

• Controllers can support other’s suggestion

• Device performs commands supported by majority
Theoretical Approach

• Cons:
  – Most distributed shared log implementations are hard to verify [D. Ongaro J. Ousterhout 2014]
  – Limited support for failures / adversaries
  – Expensive design for devices
Our Approach

• Recognize the asymmetry
  – Device failure is inherently blocking
  – Device “knows” the correct input (event history)

• Light adaptation to devices
  – Store the hash of all sent events/commands (history)
  – Accept command iff:
    • Contains correct hash
    • Signed (including the hash) by majority of controllers
Our Approach

• Other considerations:

  – Prevent race conditions (events vs. commands)
    • Keep a buffer of recent hash values
    • Accept commands with hash within buffer

  – Support fast initialization of new (or delayed) controllers
    • Commands includes controller state hash
    • New controller contacts “old” controller to receive state
    • Then contacts device to verify state
Similar Distributed Control Issue

- Concurrent configuration updates
- Example: load-balancing

![Diagram showing load balancing between two systems with link loads 1 and 2.]
Similar Distributed Control Issue

- Concurrent configuration updates
- Example: load-balancing

Add 2 flows to left

Add 2 flows to left
Our Approach [CCR16]

• Consider the centrality of the device
  – Device failure is inherently blocking
  – Device “knows” the current configuration

Zero
• Light adaptation to devices
  – Implement conditional updates
  – Based on OpenFlow (v1.4)

• Transactions over switch configuration space!

Our Approach [CCR16]

• Conditional updates

If left=1 then
Add 2 flows to left

If left=1 then
Add 2 flows to left

Abort
Summary

• SDN control plane might be compromised.

• Past events must be considered and verified.

• Our device centric approach provides a lightweight solution.

• Same approach can solve concurrency issues.