Versioned software defined networks

Behram Mistree and Phil Levis
Stanford University
Debugging networks is difficult

- **Invisibility**: Automated protocols, programs, and services change how packets are forwarded
- **Non-proportionality**: Local changes can have global effects
- **Liveness**: System actively forwards packets while it is being debugged
Consequences

• VoIP outage
  – Hours-long outage
  – Dozens of emails

• Spanning tree protocol mismatch
  – 15-45 minute outages for isolated VLANs
Versioned networking

Versioned networks tries to make it easier to debug networks by helping a debugger to:

1. Quickly identify the cause of a problem and
2. Why that cause produces the problem

By allowing the debugger to ask four questions:

1. *What* did/does the network look like?
2. *When* did an operation take place?
3. *Why* did a program behave the way that it did?
4. *Who* executed that program?
Versioned networks --- debugging with what, when, why, who

• Using what and when, construct a version history
• Automatically walk version history until find issue
• Using why and who, examine/undo operation associated version change that caused issue
Versioned network debugging: Multiple switches

Version history A

Version history B

Version history C
Goal: What, when, why, who

• What did/does the network look like?
• When did an operation take place (relative to other operations in the system)?
• Why did a program behave the way that it did?
• Who executed that program?
Outline

• Versioned networking goals
• How do you build a versioned networking system?
• Evaluation
• Discussion
Four challenges

• Switches can *reorder* flow mods
• Switches can *reject* flow mods
• Switches *never send a positive ack* for flow mods
• Distributed system

Two techniques

• Automatic barrier insertion
• Transactions
1. Add Entry A
2. Rm Entry A
Technique 1: Barrier insertion

• Barriers
  – Switch returns barrier response after all previous commands processed

• Automatically insert barriers after operations
Technique 1: Barrier insertion

1. Send set of operations that commute
2. Send barrier
3. Receive barrier response
4. Send another set of operations that commute
5. Send barrier
6. Receive barrier response

What
Because operations between barriers commute, reordering safe

When
Because wait on barrier response before issuing new commands, know *when* operations have completed
Technique 2: Transactions

• *Why* did running the VoIP SDN program insert bad flow mods?

• *Insight:* Leverage determinism
  – Absent races, re-executing a program with identical arguments and internal state produces the same output
Technique 2: Transactions

• When each program begins, log its arguments
• Take locks on internal state program touches
• Log changes to all internal state
1. Log H1, H2
2. Take locks on internal objects program touches
3. After execution logic has completed:
   1. Issue flowmods to S1 and S2
   2. Issue barriers to S1 and S2
4. On barrier success reply
   1. Log changes to internal objects
   2. Release locks
Guarantees

• Preserves causality in logs
• Allows offline replay
• Supports what, when, why, who queries
Outline

- Versioned networking goals
- How do you build a versioned networking system?
- Evaluation
- Discussion
Criteria

• Ease of use
• Performance
• Correctness
Ease of use

100 version histories
100k operations across them
<20 seconds to replay operations
Performance

![Graph showing performance metrics]
Correctness

• Forcing reordering between barriers, over 100k operations, switch state agrees with version history

• Over 100k operations with 1% and 5% errors, switch state agrees with version history
Outline

• Versioned networking goals
• How do you build a versioned networking system?
• Evaluation
• Discussion
SDN

- **Visibility**: Have access to switching and forwarding state in a way that never did before
- **Manipulability**: Multiple layers and vendors provide a uniform API to program the network
Versioned software defined networking

- Retrospective visibility and manipulability
  - What
  - When
  - Why
  - Who
Soap box

• Many existing tools and frameworks do not provide visibility for you. You can’t even answer questions such as, approximately, what did my network look like?

• Eventual consistency is bullshit. Networks that sometimes go down for 30-45 minutes is bullshit.

• It’s time to build, monitor, and debug networks in intelligent ways
Technique 1: Barrier insertion

• I think that ONOS may be starting to add these as well, but I know of no other system that either exposes them or automatically adds them.

• In these systems, it requires:
  – Bug free code
  – Heroic effort
  – Polling
Barrier insertion batching

• If a program makes changes over a series of switches, should we do each serially? Or should we batch them?
Versioned networking

- **What** did/does the network look like?
- **When** did an operation take place (relative to other operations in the system)?
- **Why** did a program behave the way that it did?
- **Who** executed that program?

- Offline, can modify and replay
- Rollback and roll forward
- Over series of controller, not just one