Towards Distributed and Reliable Software Defined Networking

Marco Canini (TU Berlin & T-Labs & UCL)
Petr Kuznetsov (TU Berlin & TU Berlin & Paris Tech)
Dan Levin (TU Berlin)
Stefan Schmid (TU Berlin & T-Labs)
Towards Distributed and Reliable Software Defined Networking

The Case for Software Transactional Networking?

Marco Canini (TU Berlin & T-Labs & UCL)
Petr Kuznetsov (TU Berlin & TU Berlin & Paris Tech)
Dan Levin (TU Berlin)
Stefan Schmid (TU Berlin & T-Labs)
The 1-Slide SDN Lecture

SDN
- Control of (forwarding) rules in network from simple, logically centralized vantage point
- Flow concept: install rules to define flow
- Match-Action concept: apply actions to packets
- Specifies global network policies, e.g., load-balancing, adaptive monitoring / heavy hitter detection, …
Vision: Middleware for Concurrent and Robust Policy Installation

Middleware

install concurrent policies

ACLs!

Tunnels!

Middleware

Install
ACK/NAK

Install
ACK/NAK

Stefan Schmid (T-Labs)
Vision: Middleware for Concurrent and Robust Policy Installation

Robust

ACLs!

Tunnels!
failures (fail-stop)

Install
ACK/NAK

Install
ACK/NAK

Middleware

compose and install concurrent policies

Stefan Schmid (T-Labs)
Policies and Composition

- **Policy** = defined over (header) domain ("flow space")
- **Policy priority**
- **Implies rules** on switch ports
- **Conflict** = overlapping domains, same priority, different treatment

- **Policy composition** = combined policy, avoids conflicts
  - E.g., composition by priorities or most specific, or do **both** parts
  - Implement exactly one policy if two conflict
  - Only known central solution: need to compose, e.g., Frenetic/Pyrethic:

```
src=* dst=11* to port A
prio=1

src=10* dst=* to port B
prio=1
```
Policy Installation

- **SDN Match-Action**
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)

- **Consistent Update: 2-phase**
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag

Known central solution (our model):

- SDN Match-Action
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)

- Consistent Update: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag

**Abstractions for Network Update**

<table>
<thead>
<tr>
<th>Mark ReiBlatt</th>
<th>Nate Foster</th>
<th>Jennifer Rexford</th>
<th>Cole Schlesinger</th>
<th>David Walker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell</td>
<td>Cornell</td>
<td>Princeton</td>
<td>Princeton</td>
<td>Princeton</td>
</tr>
</tbody>
</table>

**ABSTRACT**

Configuration changes are a common source of instability in networks leading to changes, performance disruptions, and security vulnerabilities. From when the initial and final configurations are correct, the update process itself often goes through intermediate configurations that exhibit incorrect behaviors. This paper introduces the notion of consistent network updates—updates that are guaranteed to preserve well-defined behaviors when transitioning between configurations. We identify two distinct consistency levels: perceptual and perflow, and we present general mechanisms for implementing them in software-defined networks using switch APIs like OpenFlow. We develop a formal model of OpenFlow networks and prove that consistent updates preserve a large class of network properties. These proofs are based on several optimizations that makes the overhead small to perform consistent updates. We present a certificate that the network is consistent.
Policy Installation

Initially

- **SDN Match-Action**
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)

- **Consistent Update**: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag
Policy Installation

Phase 1

- SDN Match-Action
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)
- Consistent Update: 2-phase
  - At internal ports: add new rules for new policy with new tag
  - Then at ingress ports: start tagging packets with new tag
Policy Installation

Phase 2

- **SDN Match-Action**
  - Match header (define flow)
  - Execute action (e.g., add tag or forward to port)

- **Consistent Update:** 2-phase
  - **At internal ports:** add new rules for new policy with new tag
  - **Then at ingress ports:** start tagging packets with new tag
But what about distributed and multi-author policies?

One guy in charge of setting up tunnels, one guy in charge of ACLs, …

Stefan Schmid (T-Labs)
Idea: Distributed Version

Synchronize:
- Do not override conflicting policies
- Especially ingress port(s)

Share Tags:
- Agree on tags
Problem Statement

Goals

- **All-or-nothing**: policy fully installed or not at all
- **Conflict-free**: never two conflicting policies
- **Progress**: non-conflicting policy eventually installed; and: at least one conflicting policy
- **Per-packet consistency**: per packet only one policy applied (during journey through network)
- Always rules ready when packets arrive (not under control!)
Goal: Serializable!

Example

Three switches, three policies, policy 1 and 2 with independent flow space, policy 3 conflicting:

Left: Concurrent history: 3rd policy aborted due to conflict.
Right: In the sequential history, no two requests applied concurrently. No packet is in flight while an update is being installed.

No packet can distinguish the two histories! So as though the application of policy updates is atomic and packets cross the network instantaneously.
**Bad News: Impossible Without Atomic Read-Modify-Write Ports**

**Thm:** Without atomic rmw-ports, per-packet consistent network update is impossible if a controller may crash-fail.

**Proof:**
- Single port already!
- $\pi_1$ and $\pi_2$ are conflicting
- Descendant of state $\sigma$ is extension of execution of $\sigma$.
- State $\sigma$ is i-valent if all descendants of $\sigma$ are processed according to $\pi_i$. Otherwise it is undecided.
- Initial state is undecided, and in undecided state nobody can commit its request and at least one process cannot abort its request.
- There must exist a critical undecided state after which it’s univalent if a process not longer proceeds.
- Difference cannot be observed: overriding violates consistency (sequential composition).

QED
Valency Proof

bivalent

univalent

π₁

π₀

π₂

1

0

0

1

1

1
Thm: With atomic RMW, the TAG algorithm is correct and wait-free (up to n-1 failures).

Principles:
1. Unique tag per policy
2. Install at internal ports first
   (compose if necessary*)
3. Once installed at internal ports…
4. … add tag to all packets at ingress port(s)!

* requires atomic read-modify-write

Observations:
- Rule always ready internally (2)
- Per-packet consistency solved (4): packet never changes tag!
- Wait-free policy installation!

Stefan Schmid (T-Labs)
Conclusion

- Concurrent SDN policy updates: A case for “Software Transactional Networking”?
  
- Concurrent control not possible under atomic r/w, but possible under atomic r+w
  
- Future work: reduce tag size