Software Transactional Networking

A Robust and Distributed SDN Control Plane

Marco Canini

with Stefan Schmid, Petr Kuznetsov, Dan Levin
Network Policy Specification

Centralized Network Policy

Controller Platform

Routing  Load Balancing  Monitoring  Access Control  Waypoint
Network Policy Specification

1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. ...
7. ...

forward from X to Y
Centralized Network Control?

Fully centralized \(\rightarrow\) Inadequate availability, scalability and responsiveness
Distributed Network Control

Centralized Network Policy

Controller Platform

Composition?

Routing
Load Balancing
Monitoring
Access Control
Waypoint

Consistency and concurrency
Faults and asynchronous communication
What to do with packet dst = H and dport = 80?
Can Routing and Monitoring compose?
Ordering Policies

What to do with packet dst = H and src = S?
Does Waypoint have precedence over Routing?
In the general case, policies might conflict.

**Examples:**
- Overlapping domains and same precedence
- Scarce flowtable resources

Must avoid conflicting policies
Pick one and reject the other?
Now, consider policy composition in the distributed control plane...
Concurrency Issues

- **Routing**
  - $\text{dst} = H \rightarrow \text{fwd}(H)$

- **Monitoring**
  - $\text{dport} = 80 \rightarrow \text{count}$

- Empty
  - $\text{dst} = H \rightarrow \text{fwd}(H)$
  - $\text{dport} = 80 \rightarrow \text{count}$

- Empty
Concurrency Issues

Routing

Monitoring

Routing

Monitoring

Empty

Empty
Concurrency Issues
Concurrency Issues

Impossible to guarantee a deterministic outcome **without synchronization**
Two Underlying Problems

1. Consistent and Concurrent Policy Composition

2. Consistent and Robust Policy Composition
   – In face of controller failures

Can we realize a general distributed policy composition interface that is agnostic to control logic, state distribution and reader-writer model?
Software Transactional Networking

1. All-or-nothing semantics
2. Optimistic concurrency for policy composition
3. Non conflicting policies eventually installed
4. Per-packet consistent updates

We don’t control traffic!
Conceptualizing STN

Routing

Monitoring

Waypoint

STN

Atomic Read-Modify-Write
STN Algorithm

apply(π)

  tag = unique tag for π

  foreach internal port x do
    if policy at x can compose with π then
      foreach subset s of tagged policies at x do
        tag’ = composition of tags of s and tag
        add composed policy with π, tag’ to x
      end
    else remove installed rules; return nack(reason)
  end

  foreach ingress port y do
    tag’ = composition of tags of y and tag
    add rules to tag packets with tag’ composed with policy at y and π
  end

end
STN Example

Routing

Monitoring

Ingress ports

Internal ports
STN Example

\[ \pi_1 \]
- \( \text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \)
- \( \text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \ \text{fwd}(\square) \)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \ \text{fwd}(\square) \)

\[ \pi_2 \]
- \( \text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \)
- \( \text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \)
- \( \text{dport} = 80 \rightarrow \text{tag} = t_2; \ \text{count} \)
- \( \text{dport} = 80 \rightarrow \text{tag} = t_2; \ \text{count} \)
Routing Monitoring

\[ \pi_1 \quad \quad \pi_2 \]

- \( \text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \)
- \( \text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\square) \)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\square) \)

- \( \text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \)
- \( \text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \)
- \( \text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count} \)
- \( \text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count} \)
STN Example

\[ \pi_1 \\
\text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \\
\text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \\
\text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(H) \\
\text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(H) \\
\]

\[ \pi_2 \\
\text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \\
\text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \\
\text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count} \\
\text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count} \\
\]

\[ \text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H) \\
\text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count} \\
\text{tag} = t_{1,2} \land \text{dst} = H \land \text{dport} = 80 \rightarrow \text{count}; \text{fwd}(H) \\
\text{tag} = t_{1,2} \land \text{dst} = H \rightarrow \text{fwd}(H) \\
\text{tag} = t_{1,2} \land \text{dport} = 80 \rightarrow \text{count} \\
\]

\( \pi_1 \) is incomplete but packets at \( \text{H} \) get tagged with \( t_1 \)
STN Example

\[ \pi_1 \]
- \( \text{tag} = t_1 \land \text{dst} = H \Rightarrow \text{fwd}(H) \)
- \( \text{tag} = t_1 \land \text{dst} = H \Rightarrow \text{fwd}(H) \)
- \( \text{dst} = H \Rightarrow \text{tag} = t_1; \text{fwd}(\_\_) \)
- \( \text{dst} = H \Rightarrow \text{tag} = t_1; \text{fwd}(\_\_) \)

\[ \pi_2 \]
- \( \text{tag} = t_2 \land \text{dport} = 80 \Rightarrow \text{count} \)
- \( \text{tag} = t_2 \land \text{dport} = 80 \Rightarrow \text{count} \)
- \( \text{dport} = 80 \Rightarrow \text{tag} = t_2; \text{count} \)
- \( \text{dport} = 80 \Rightarrow \text{tag} = t_2; \text{count} \)

\( \pi_2 \) is incomplete but packets at \( \_\_ \) get tagged with \( t_2 \)
STN Example

$\pi_1$
- $\text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H)$
- $\text{tag} = t_1 \land \text{dst} = H \rightarrow \text{fwd}(H)$
- $\text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\Box)$
- $\text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\Box)$

$\pi_2$
- $\text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count}$
- $\text{tag} = t_2 \land \text{dport} = 80 \rightarrow \text{count}$
- $\text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count}$
- $\text{dport} = 80 \rightarrow \text{tag} = t_2; \text{count}$

$dport = 80 \rightarrow \text{tag} = t_2; \text{count}$
$\text{dst} = H \land \text{dport} = 80 \rightarrow \text{tag} = t_{1,2}; \text{count}; \text{fwd}(H)$
$\text{dst} = H \rightarrow \text{tag} = t_{1,2}; \text{fwd}(H)$
$dport = 80 \rightarrow \text{tag} = t_{1,2}; \text{count}$

$\pi_1$ is complete; packets at $\Box$ get tagged with $t_{1,2}$
STN Example

\[ \pi_1 \]
- tag = \( t_1 \land \text{dst} = H \) \( \rightarrow \) fwd(H)
- tag = \( t_1 \land \text{dst} = H \) \( \rightarrow \) fwd(H)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\text{black}) \)
- \( \text{dst} = H \rightarrow \text{tag} = t_1; \text{fwd}(\text{black}) \)

\[ \pi_2 \]
- tag = \( t_2 \land \text{dport} = 80 \) \( \rightarrow \) count
- tag = \( t_2 \land \text{dport} = 80 \) \( \rightarrow \) count
- dport = 80 \( \rightarrow \) tag = \( t_2 \); count
- dport = 80 \( \rightarrow \) tag = \( t_2 \); count

\[ \pi_1 \] and \( \pi_2 \) are complete; packets at \( \square \) and \( \blacksquare \) get tagged with \( t_{1,2} \)
Limitations of STN

• # tags grows exponentially with policies

• No fault tolerance
1. All-or-nothing semantics
2. Tolerate up to $f$ controller crash failures
3. Non conflicting policies eventually installed and at least one policy commits (among conflicting ones)
4. Ensure updates affect traffic as a sequential composition of their policies

Require robustness but not efficiency

Per-packet consistency
Conceptualizing CPC

Reliable but asynchronous channel

Every controller receives and installs every policy
CPC Example

Original history

Sequential equivalent
CPC implementation: model v1

- Controllers access switch ports with read and write operations
- Controllers can communicate via asynchronous message-passing
- Controllers may fail by crashing
- No synchrony assumptions
- Restrict policies to forwarding
  - Compose if domains are disjoint or related by precedence
  - Reject otherwise
Asynchronous read-write CPC

Theorem: 1-resilient read-write CPC is impossible

Proof sketch:

• Two ingress ports 1 and 2 initially forward all to the internal ports ($\pi_0$)
• $\pi_1$ installed by $p_1$ and $\pi_2$ installed by $p_2$, $\pi_2$ refines $\pi_1$ (higher precedence, same domain)
• $\pi_1$ and $\pi_2$ propose different paths
• $p_1$ changes port 1 and is just about to change 2 (with a composition of $\pi_0$ and $\pi_1$), $p_2$ takes no steps
• $p_2$ wakes up and installs of $\pi_0 \pi_1 \pi_2$, $p_1$ takes no steps
• $p_1$ changes port 2 with $\pi_0 \pi_1$: $\pi_2$ is forgotten!
CPC implementation: model v2

- Controllers access ports with atomic read-modify-write ops RMW(f,g,v):
  - read the state v’
  - write f(v,v’)
  - return g(v,v’)

- Intuition: do not update if conflicts with currently installed policy
Upper bound: FixTag algorithm

Operation:

1. Unique tag per path
2. Broadcast policy $\pi$ to all other controllers
3. Update ingress ports in predefined order
4. ... add rule to tag all packets matching $\text{dom}(\pi)$ with the tag corresponding to the path($\pi, i$) for ingress port $i$

Upsides: wait-free (tolerates all failure patterns)

Downsides: overhead can be huge
   - Super-exponential in the size of the network
Can we do better?

- No, if we get no feedback from the network
  - Tag \( t \) cannot be reused if a packet tagged with \( t \) is still “in flight”

- Suppose, we can correctly evaluate the set of active tags
  - Correct (but asynchronous) oracle

- Single-controller scenario: one bit is enough!
  - Upon policy update \( \pi_i \) wait until \( (i \mod 2) \)-traffic is over, and use tag \( i \mod 2 \)

- Two or more controllers: inherent price of concurrency?
  - Between constant and super-exponential?
  - Yes, if controllers coordinate use of tags
ReuseTag: linear complexity

• Proportional to the level of resilience:
  – Up to f failures: f+2 tags needed (proved optimal)

• Controllers use consensus instances (eventual synchrony or «eventual leader»)
  – Replicated state machine that imposes a global order on the policy updates and ensure coordinated use and reuse of tags

• All requests are serialized
  – Even non-conflicting ones
  – Can we do better?
Summary

• Framework for concurrent and consistent policy composition in distributed SDN

• Transactional interface to manipulate the network as though there is no concurrency
  – Policies compose or conflict (and abort)
Acknowledgements

Petr Kuznetsov   Dan Levin   Stefan Schmid

Supported by ARC grant 13/18-054 from Communauté française de Belgique
References

Software Transactional Networking: Concurrent and Consistent Policy Composition
HotSDN’13

STN: A Robust and Distributed SDN Control Plane
ONS’14

A Distributed SDN Control Plane for Consistent Policy Updates
ArXiv technical report 1305.7429