Enforcing Customizable Consistency Properties in Software-Defined Networks

Wenxuan Zhou, Dong Jin, Jason Croft, Matthew Caesar, Brighten Godfrey
Keeping network correct consistently over time.

-- Network Consistency

Network changes

• control applications,
• changes in traffic load,
• system upgrades,
• …
1. Correctness at every step
2. Customizable properties
3. With efficient update installation

What is Correctness?
- firewall traversal,
- access control,
- balanced load,
- loop freedom,
- …
Problem Statement

1. Consistency at every step
2. Customizable consistency properties
3. Efficient updates installation

Is it possible to efficiently ensure customizable correctness properties as the network evolves?
Prior Work

Network Verification

Dionysus

Fixed Consistency Property

Consistent Updates

Dynamic Scheduling of Network Updates

Abstractions for Network Update
Ideally given *arbitrary* invariants, a sequence with *minimized* overhead is produced.
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** → **Verification**

Diagram:

- **Controller**
  - **Stream of Updates**
  - **Network Model**
    - **CCG**
    - **Buffer of pending updates**
      - **Verification Engine**
        - **Confirmations**
          - **Pass**
          - **Fail**

Verification outcomes:

- No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...

Diagram:

Network topology with labeled nodes and connections.
Our design: **Customizable Consistency Generator**

Challenges:

- Greedy algorithm may get stuck
  - identify the scope of cases that guarantees no deadlock
  - For other cases, a more heavyweight update technique as a fallback, triggered rarely in practice

- Distributed nature of networks (uncertainty)
  - compact *uncertain forwarding graph*
  - verification optimization
Network Uncertainty

The “uncertainty” of an observation point tasked with instilling updates in knowing the current network state.

May deviate network behavior away from desired properties.
Uncertainty-aware Modeling Basis: VeriFlow
Uncertainty-aware Modeling Basis: VeriFlow

Equivalence class: Packets experiencing the same forwarding actions throughout the network.

Forwarding graphs:
Uncertainty-aware Modeling

Naively, represent every possible network state $O(2^n)$

Uncertain graph: represent all possible combinations

When to change “uncertain” to “certainty”?

How to verify the network under “uncertainty”?

The model captures packets’ view of the network, assuming controller initiates changes.
Consistency under Uncertainty

Enforcing consistency with max parallelism heuristically

Waypoint Properties: flows are required to traverse a set of waypoints

- connectivity,
- waypointing,
- access control,
- service chaining, ...

Theorem: Segment independent properties is guaranteed by the heuristic.
Consistency under Uncertainty

Even with FB triggered, CCG achieves better efficiency than using FB alone.
System Structure

Controller

Uncertainty-aware Network Model

Stream of Updates

Buffer of pending

Fail

Verification Engine

Pass

CCG

No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak,

Fallback Mechanism

Verify

Stream of Updates

Confirmations

Controller

Uncertainty-aware Network Model

Buffer of pending

Verification Engine

Deadlock free, Resource isolation,
Evaluation

Can CCG verify network invariants in *real time*?

Can CCG achieve *performance gain* during network transitions with its algorithm for maximizing the parallelism of applying updates?

- Segment-independent Policies
- Non-segment-independent Policies
- Emulations
- Testbed experiments
Simulated network: BGP RIBs and update trace from RouteViews injected into 172-router AS 1755 topology, checking reachability invariant

15X less memory overhead (540MB vs. 9GB)
Emulation: Segment-independent Policies

Controller-switch delay = network delay + processing delay

- Local (4ms)
- Wide area (100ms)

Measure: path completion time

Controller-switch delay = network delay + processing delay
Emulation: Segment-independent Policies

No fallback triggered
No additional memory

Wide area

Local

Fraction of trials

Millissecond

Optimal
CCG
CCG-waypoint
Dionysus
Consistent Updates
Incremental CU
Emulation: Non-segment-independent Policies

Traces from a enterprise network with 200+ layer-3 devices.

One day, one snapshot per hour, 24 transitions, 4ms delay.

- New rules were added first, then old rules deleted.

Rules overlapped with longest prefix match, not segment-independent.

Fallbacks happened rarely.

Overhead close to Immediate Update, with no transient connectivity violations.
Conclusion

Uncertainty problem with network control

Uncertainty-aware network model

GCC, a system that

- enforces customizable network consistency properties with
- heuristically optimized efficiency.

Ongoing work:

- Study the generality of segment independency
- Test with more data traces, and compare against the original implementation of Dionysus
- Handle changes initiated from the network.