Modular Control Plane Verification via Temporal Invariants

Timothy Alberdingk Thijm
Princeton University
Aarti Gupta
Princeton University
Ryan Beckett
Microsoft Research
David Walker
Princeton University

PLDI 2023
What is the Control Plane?
What is the Control Plane?

Goal: determine routes to use to forward traffic
What is the Control Plane?

Goal: determine routes to use to forward traffic

Send initial route announcements
What is the Control Plane?

Goal: determine routes to use to forward traffic

Send initial route announcements

Receive announcements, process according to configs

Diagram:

- Node a
- Node b
- Node c
- Node d

Arcs indicate connections and directions.
**What is the Control Plane?**

Goal: determine routes to use to forward traffic

Send initial route announcements

Receive announcements, process according to **configs**

Distributed, written in vendor-specific, low-level language
What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to configs
- Select best announcement

Distributed, written in vendor-specific, low-level language
What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to configs
- Select best announcement
- Broadcast selected route to neighbors

Distributed, written in vendor-specific, low-level language
What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to configs
- Select best announcement
- Broadcast selected route to neighbors
- Converge to a stable state

Distributed, written in vendor-specific, low-level language
What is the Control Plane?

Goal: determine routes to use to forward traffic

- Send initial route announcements
- Receive announcements, process according to configs
- Select best announcement
- Broadcast selected route to neighbors
- Converge to a stable state

Policies for security, traffic engineering, fault tolerance, cost, etc.

Distributed, written in vendor-specific, low-level language
How Do We Verify Control Planes?

Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, etc.
How Do We Verify Control Planes?

Analyze all configurations together to find property violations using a control plane verifier (e.g., Batfish, ARC, Minesweeper, Bagpipe, Tiramisu, Plankton, Hoyan)

Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, etc.
How Do We Verify Control Planes?

Analyze all configurations together to find property violations using a control plane verifier (e.g., Batfish, ARC, Minesweeper, Bagpipe, Tiramisu, Plankton, Hoyan)

Network configuration files

Policies for security, traffic engineering, fault tolerance, cost, etc.

Repeat when configurations change
Many networks are too big and too complex to verify monolithically!
Scaling Control Plane Verification

modular verification to the rescue!

Many networks are too big and too complex to verify monolithically!
Our Contributions

demonstrate why naive stable states analysis is unsuitable for modular verification

present time-based theory for modular control plane analysis, with SMT-based verification procedure

verify properties of 2000-node data centers and complex wide-area networks in seconds!
How (Not) to Modularly Verify
Modular Network Verification
Modular Network Verification

sound modular analysis: captures all monolithic routing behavior
Modular Network Verification

sound modular analysis: captures all monolithic routing behavior

split the network up into node-local components to verify independently
Modular Network Verification

- **sound modular analysis**: captures all monolithic routing behavior
- split the network up into **node-local components** to verify independently
- represent cross-component dependencies using **interfaces**
Interfaces

User

\[u_1 \leq 2 \leq u_2 \leq 3 \leq u_3 > 3\]
interface $A$ over-approximates the converged states of the node $v$ with a set of states $A(v)$
Bear’s Modular Verification Procedure

Verification condition:
If $v$ receives any routes satisfying $A(u_1), A(u_2), \ldots, A(u_m)$, does its selected route satisfy $A(v)$?
An Example Network

$n$ \rightarrow \text{filter} \rightarrow u \rightarrow v
An Example Network

routes: $\langle l_p : \mathbb{N}, \text{len} : \mathbb{N} \rangle$ or $\infty$
select by highest $l_p$, lowest $\text{len}$
An Example Network

routes: \( (lp : \mathbb{N}, len : \mathbb{N}) \) or \( \infty \)
select by highest \( lp \), lowest \( len \)

filter external routes
An Example Network

routes: \( \langle lp : \mathbb{N}, \text{len} : \mathbb{N} \rangle \) or \( \infty \)
select by highest \( lp \), lowest \( \text{len} \)

filter external routes

\[ \langle 100, 0 \rangle \]
An Example Network

filter external routes

routes: \( \langle l_p : \mathbb{N}, l_{en} : \mathbb{N} \rangle \) or \( \infty \)
select by highest \( l_p \), lowest \( l_{en} \)

\( n \) -> filter -> \( u \) -> \( v \)

\( \langle 100, 0 \rangle \) -> \( \infty \)
An Example Network

- Filter external routes

\[ \langle l_p : \mathbb{N}, l_e : \mathbb{N} \rangle \text{ or } \infty \]

Select by highest \( l_p \), lowest \( l_e \)
Execution Interference

\[ s \cdot \text{lp} = 200 \]

\[ \langle 100, 0 \rangle \]  \[ \langle 100, 1 \rangle \]
Execution Interference

\[ v \text{ sends a route with } \text{lp} = 200, \]
\[ \text{so } u \text{ has } \text{lp} = 200 \]

\[ s \cdot \text{lp} = 200 \]

\[ \langle 100, 0 \rangle \]

\[ \langle 100, 1 \rangle \]
Execution Interference

\[ s \cdot lp = 200 \]

\[ \langle 100, 0 \rangle \]

\[ \langle 100, 1 \rangle \]
Execution Interference

\( v \) sends a route with \( lp = 200 \), so \( u \) has \( lp = 200 \)

\( u \) sends a route with \( lp = 200 \), so \( v \) has \( lp = 200 \)

\( n \)

Filter

\( s \cdot lp = 200 \)

\( v \)'s converged route does NOT have \( lp = 200 \)!
Execution Interference

Interfaces are **unsound**: exclude the legitimate converged routes, but the checks pass!
Timepiece’s Temporal Model
interface $A$ over-approximates the converged states of the node $v$ with a set of states $A(v)$
temporal interface $A$ over-approximates the states of the node $v$ at time $t$ with a set of states $A(v)(t)$
Temporal Interfaces

Temporal interface $A$ over-approximates the states of the node $v$ at time $t$ with a set of states $A(v)(t)$.
Temporal Interfaces

Temporal interface $A$ over-approximates the states of the node $v$ at time $t$ with a set of states $A(v)(t)$.

**base check:** interfaces at time 0 holds on initial routes

**inductive check:** for all times $t$, interfaces at time $t+1$ holds given interfaces from time $t$
Preventing Interference

\[ G(\text{true}) \]

\[ G(s \cdot lp = 200) \]

\[ G(s \cdot lp = 200) \]

\[ n \rightarrow \text{filter} \rightarrow u \rightarrow v \]

\[ \langle 100, 0 \rangle \rightarrow \infty \]
Preventing Interference

\( \mathcal{GP} \text{ (globally } P \text{): at every point in time, the predicate } P \text{ holds} \)

\( \mathcal{G}(\text{true}) \quad \mathcal{G}(s \cdot \text{lp} = 200) \quad \mathcal{G}(s \cdot \text{lp} = 200) \)

\( n \rightarrow \text{filter} \rightarrow u \rightarrow v \)

\( \langle 100, 0 \rangle \quad \infty \)
Preventing Interference

\( GP(\text{globally } P) \): at every point in time, the predicate \( P \) holds

\( G(\text{true}) \)

\( G(s \cdot lp = 200) \)

\( G(s \cdot lp = 200) \)

Base checks fail: interfaces \( A(u) \) and \( A(v) \) do not hold for initial routes at time 0
Proving Path Length

\[ \mathcal{G}(true) \]

\[ \mathcal{G}(s.lp = 100 \land s.len = 0) \]
Proving Path Length

\( \mathcal{G}(\text{true}) \)

\[ s = \infty \quad \mathcal{U}^1 \quad s . \text{lp} = 100 \land s . \text{len} \leq 1 \]

\( \mathcal{G}(s . \text{lp} = 100 \land s . \text{len} = 0) \)

\( \mathcal{G} \)
Proving Path Length

\[ P \cup^t Q (P \text{ until } Q \text{ at } t): \text{ until time } t, P \text{ holds; at and after } t, Q \text{ holds} \]

\[ \mathcal{G}(\text{true}) \]

\[ s = \infty \ \cup^1 s.\text{lp} = 100 \land s.\text{len} \leq 1 \]

\[ \mathcal{G}(s.\text{lp} = 100 \land s.\text{len} = 0) \]
Proving Path Length

\[ P \mathcal{U}^t Q \ (P \text{ until } Q \text{ at } t) : \text{ until time } t, P \text{ holds; at and after } t, Q \text{ holds} \]

\[ \mathcal{G}(true) \]

\[ s = \infty \ \mathcal{U}^1 \ s.p = 100 \land s.len \leq 1 \]

\[ \mathcal{G}(s.p = 100 \land s.len = 0) \]
Proving Path Length

\[ P \mathcal{U}^t Q \text{ (P until Q at t): until time } t, P \text{ holds; at and after } t, Q \text{ holds} \]

\[ \mathcal{G}(\text{true}) \]

\[ s = \infty \mathcal{U}^1 s.\text{lp} = 100 \land s.\text{len} \leq 1 \]

\[ \mathcal{G}(s.\text{lp} = 100 \land s.\text{len} = 0) \]

\[ v \text{ has no route until time 1} \]

\[ \text{at time 1, } v \text{ has a route with } \text{lp} = 100 \text{ and } \text{len} \leq 1 \]
Soundness Theorem
Soundness Theorem

If interface $A$ satisfies the base and inductive checks for all nodes, then $A$ includes all states computable via (monolithic) simulation.
Soundness Theorem

If interface $A$ satisfies the base and inductive checks for all nodes, then $A$ includes all states computable via (monolithic) simulation.

Proof by induction on time.
How to Use Timepiece
How to Use Timepiece

define network semantics in C# or via configurations (via Batfish)
How to Use Timepiece

- define network semantics in C# or via configurations (via Batfish)
- write interfaces using C# library of temporal operators ($\mathcal{G}, \mathcal{U}', \mathcal{F}'$)
How to Use Timepiece

- Define network semantics in C# or via configurations (via Batfish)
- Write interfaces using C# library of temporal operators ($G, U', F'$)
- Check VCs in parallel on every node using Satisfiability Modulo Theories (SMT) solver
Evaluation
Evaluation
Evaluation

does Timepiece scale to large networks?

does Timepiece handle complex policies?

how easy is it to write invariants for different properties?
Fat-tree data center networks
C# model of eBGP routing protocol
20–2000 nodes

how easy is it to write invariants for different properties?

does Timepiece handle complex policies?
Evaluation

Fat-tree data center networks
C# model of eBGP routing protocol
20–2000 nodes

Internet2 wide-area network
102,753 lines of Juniper configuration code
263 nodes (10 internal, 253 external)

how easy is it to write invariants for different properties?
Evaluation

Fat-tree data center networks
C# model of eBGP routing protocol
20–2000 nodes

Reachability
Valley freedom

Path length
Hijack filtering

Internet2 wide-area network
102,753 lines of Juniper configuration code
263 nodes (10 internal, 253 external)

No transit
Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM
Evaluation

Benchmark

- Reachability
- Path length
- Valley freedom
- Hijack filtering
- No transit

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM
## Evaluation

Evaluate on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
</tr>
</tbody>
</table>
## Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
<td>3</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
<td>7</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
<td>12</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
<td>21</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
<td>5</td>
</tr>
</tbody>
</table>
## Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
<td>7</td>
<td>2000</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
<td>12</td>
<td>2000</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
<td>21</td>
<td>2000</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
<td>5</td>
<td>263</td>
</tr>
</tbody>
</table>
## Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Nodes</th>
<th>Monolithic time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
<td>3</td>
<td>2000</td>
<td>14s</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
<td>7</td>
<td>2000</td>
<td>&gt;2h</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
<td>12</td>
<td>2000</td>
<td>&gt;2h</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
<td>21</td>
<td>2000</td>
<td>&gt;2h</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
<td>5</td>
<td>263</td>
<td>&gt;2h</td>
</tr>
</tbody>
</table>
# Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Nodes</th>
<th>Monolithic time</th>
<th>Modular time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
<td>3</td>
<td>2000</td>
<td>14s</td>
<td>28s</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
<td>7</td>
<td>2000</td>
<td>&gt;2h</td>
<td>1204s</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
<td>12</td>
<td>2000</td>
<td>&gt;2h</td>
<td>398s</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
<td>21</td>
<td>2000</td>
<td>&gt;2h</td>
<td>142s</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
<td>5</td>
<td>263</td>
<td>&gt;2h</td>
<td>38s</td>
</tr>
</tbody>
</table>
## Evaluation

on Microsoft Azure D96s VM with 96 vCPUs and 384GB RAM

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Nodes</th>
<th>Monolithic time</th>
<th>Modular time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>81</td>
<td>3</td>
<td>2000</td>
<td>14s</td>
<td>28s</td>
</tr>
<tr>
<td>Path length</td>
<td>88</td>
<td>7</td>
<td>2000</td>
<td>&gt;2h</td>
<td>1204s</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>89</td>
<td>12</td>
<td>2000</td>
<td>&gt;2h</td>
<td>398s</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>146</td>
<td>21</td>
<td>2000</td>
<td>&gt;2h</td>
<td>142s</td>
</tr>
<tr>
<td>No transit</td>
<td>88 (+102,753)</td>
<td>5</td>
<td>263</td>
<td>&gt;2h</td>
<td>38s</td>
</tr>
</tbody>
</table>
Fat-tree Hijack Filtering

BGP misconfiguration/attack:
a “hijacker” node announces it has a path to a prefix it
doesn’t own, misleading others to route through the hijacker
Fat-tree Hijack Filtering

BGP misconfiguration/attack: a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker.
Fat-tree Hijack Filtering

BGP misconfiguration/attack:
a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker.

Sends a route with symbolic prefix $p$.
Fat-tree Hijack Filtering

BGP misconfiguration/attack: a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker

Sends a route with symbolic prefix $p$

Sends a symbolic route $h$ with tag true
Fat-tree Hijack Filtering

BGP misconfiguration/attack: a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker.

Sends a symbolic route $h$ with tag true

Filter routes for prefix $p$

Sends a route with symbolic prefix $p$
Fat-tree Hijack Filtering

Converged routes for prefix $p$ should not come from $h$

$$P(v) \equiv \text{true} \land \forall \mathcal{A}_4 \mathcal{S} \cdot \text{prefix} = p \land \neg \text{tag}$$

Sends a symbolic route $h$ with tag true

Filter routes for prefix $p$

BGP misconfiguration/attack:

a “hijacker” node announces it has a path to a prefix it doesn’t own, misleading others to route through the hijacker

Sends a route with symbolic prefix $p$
Fat-tree Hijack Filtering

Converged routes for prefix $p$ should not come from $h$

$$P(v) \equiv \text{true } \forall^4 s. \text{ prefix } = p \land \neg s \land \text{ tag}$$

Interface *composes* an “eventual invariant” with a “safety invariant”
Fat-tree Hijack Filtering

Converged routes for prefix $p$ should not come from $h$

$$P(v) \equiv \text{true} \land \forall^4 s. \text{prefix} = p \land \neg s. \text{tag}$$

Interface \textit{composes} an “eventual invariant” with a “safety invariant”

All nodes’ interfaces are parameterized by their distance $\text{dist}(v)$ from $e_{19}$

$$\text{dist}(c_3) = 2$$
Fat-tree Hijack Filtering

Converged routes for prefix $p$ should not come from $h$

$$P(v) \equiv \text{true } \forall^4 s . \text{prefix } = p \land \neg s . \text{tag}$$

Interface composes an “eventual invariant” with a “safety invariant”

All nodes’ interfaces are parameterized by their distance $\text{dist}(v)$ from $e_{19}$

Nodes are eventually “internally reachable”

$$\text{true } \forall^{\text{dist}(v)} s . \text{prefix } = p \land \neg s . \text{tag}$$

Converged routes for prefix $p$ should not come from $h$

$$P(v) \equiv \text{true } \forall^4 s . \text{prefix } = p \land \neg s . \text{tag}$$

Interface composes an “eventual invariant” with a “safety invariant”

All nodes’ interfaces are parameterized by their distance $\text{dist}(v)$ from $e_{19}$

Nodes are eventually “internally reachable”

$$\text{true } \forall^{\text{dist}(v)} s . \text{prefix } = p \land \neg s . \text{tag}$$
Fat-tree Hijack Filtering

Converged routes for prefix $p$ should not come from $h$

\[ P(v) \equiv \text{true } \mathcal{U}^{4} s. \text{prefix } = p \land \neg s. \text{tag} \]

Interface \textit{composes} an “eventual invariant” with a “safety invariant”

All nodes’ interfaces are parameterized by their distance $\text{dist}(v)$ from $e_{19}$

Nodes are \textit{eventually} “internally reachable”

\[ \text{true } \mathcal{U}^{\text{dist}(v)} s. \text{prefix } = p \land \neg s. \text{tag} \]

Nodes \textit{never} use hijacking routes

\[ \mathcal{G}(s. \text{prefix } = p \rightarrow \neg s. \text{tag}) \]
Fat-tree Hijack Filtering

The graph shows the verification time (in seconds) for different node counts. The verification time is represented on a logarithmic scale, with the y-axis ranging from $10^{-1}$ to $10^4$ seconds.

- **TIMEPIECE**: The blue line represents the verification time for the TIMEPIECE approach.
- **TIMEPIECE median**: The orange triangles indicate the median verification time for TIMEPIECE.
- **TIMEPIECE 99th p.**: The green squares represent the 99th percentile verification time for TIMEPIECE.
- **Monolithic**: The red diamonds signify the verification time for the monolithic approach.

The graph highlights that TIMEPIECE is more efficient than the monolithic approach, especially as the number of nodes increases. The t/o (time out) mark indicates the threshold beyond which the verification process is not completed.

**Nodes** range from 0 to 2,000 on the x-axis.
Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes

Verification time [s]

Nodes

TIMEPIECE
TIMEPIECE median
TIMEPIECE 99th p.
Monolithic

Monolithic verification times out (>2h) at 80 nodes
Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes

max. wall clock time: ~2.2 minutes
Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes

Monolithic verification times out (>2h) at 80 nodes

max. wall clock time: ~2.2 minutes

99% of nodes complete checks in <5 seconds
Fat-tree Hijack Filtering

Monolithic verification times out (>2h) at 80 nodes

Monolithic verification times proportional to node’s degree

Verification time [s]

Veriﬁcation time proportional to node’s degree

max. wall clock time: ~2.2 minutes

TIMEPIECE
TIMEPIECE median
TIMEPIECE 99th p.

Monolithic

99% of nodes complete checks in <5 seconds

Nodes
Takeaways
Takeaways

Big, complex control planes need modular tools
Takeaways

Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**
Takeaways

Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**

Scale to thousands of nodes & complex policies
Big, complex control planes need modular tools

Temporal invariants provide a **correct basis for modular verification**

Scale to thousands of nodes & complex policies

Read the paper to learn more!
I’m looking for a job!
cs.princeton.edu/~tthijm

Our paper

Thank You!

Tim Alberdingk Thijm
Princeton

Ryan Beckett
Microsoft Research

Aarti Gupta
Princeton

Dave Walker
Princeton
Extra slides
Closed Completeness Theorem
Closed Completeness Theorem

Starting from fixed initial routes, if $\sigma(v)(t)$ is the (monolithic) state of node $v$ at time $t$, then the interface $A(v)(t) = \{\sigma(v)(t)\}$ satisfies the base and inductive checks for all nodes.
Closed Completeness Theorem

Starting from fixed initial routes, if \( \sigma(v)(t) \) is the (monolithic) state of node \( v \) at time \( t \), then the interface \( A(v)(t) = \{ \sigma(v)(t) \} \) satisfies the base and inductive checks for all nodes.

Proof by induction on time.
Evaluation
Evaluation

Benchmark

- Reachability
- Path length
- Valley freedom
- Hijack filtering
- No transit
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>20-2000</td>
</tr>
<tr>
<td>Path length</td>
<td>20-2000</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>20-2000</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>20-2000</td>
</tr>
<tr>
<td>No transit</td>
<td>263</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>Network LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>20-2000</td>
<td>81</td>
</tr>
<tr>
<td>Path length</td>
<td>20-2000</td>
<td>88</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>20-2000</td>
<td>89</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>20-2000</td>
<td>146</td>
</tr>
<tr>
<td>No transit</td>
<td>263</td>
<td>88 (+102,753)</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>20-2000</td>
<td>81</td>
<td>3</td>
</tr>
<tr>
<td>Path length</td>
<td>20-2000</td>
<td>88</td>
<td>7</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>20-2000</td>
<td>89</td>
<td>12</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>20-2000</td>
<td>146</td>
<td>21</td>
</tr>
<tr>
<td>No transit</td>
<td>263</td>
<td>88</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+102,753)</td>
<td></td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Monolithic hits 2h timeout?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>20-2000</td>
<td>81</td>
<td>3</td>
<td>No (fixed dest.) 80 nodes (symbolic dest.)</td>
</tr>
<tr>
<td>Path length</td>
<td>20-2000</td>
<td>88</td>
<td>7</td>
<td>80 nodes (fixed dest.) 20 nodes (symbolic dest.)</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>20-2000</td>
<td>89</td>
<td>12</td>
<td>180 nodes</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>20-2000</td>
<td>146</td>
<td>21</td>
<td>80 nodes (fixed dest.) 20 nodes (symbolic dest.)</td>
</tr>
<tr>
<td>No transit</td>
<td>263 (+102,753)</td>
<td>88</td>
<td>5</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>Network LoC</th>
<th>Annotation LoC</th>
<th>Monolithic hits 2h timeout?</th>
<th>Modular verification time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>20-2000</td>
<td>81</td>
<td>3</td>
<td>No (fixed dest.)</td>
<td>28s (fixed 2000 nodes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80 nodes (symbolic dest.)</td>
<td>336s (symbolic 2000 nodes)</td>
</tr>
<tr>
<td>Path length</td>
<td>20-2000</td>
<td>88</td>
<td>7</td>
<td>80 nodes (fixed dest.)</td>
<td>1204s (fixed 2000 nodes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 nodes (symbolic dest.)</td>
<td>3953s (symbolic 2000 nodes)</td>
</tr>
<tr>
<td>Valley freedom</td>
<td>20-2000</td>
<td>89</td>
<td>12</td>
<td>180 nodes</td>
<td>398s (fixed 2000 nodes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3506s (symbolic 1280 nodes)</td>
</tr>
<tr>
<td>Hijack filtering</td>
<td>20-2000</td>
<td>146</td>
<td>21</td>
<td>80 nodes (fixed dest.)</td>
<td>142s (2000 nodes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 nodes (symbolic dest.)</td>
<td>2196s (symbolic 2000 nodes)</td>
</tr>
<tr>
<td>No transit</td>
<td>263 (+102,753)</td>
<td>88</td>
<td>5</td>
<td>Yes</td>
<td>38s</td>
</tr>
</tbody>
</table>
Related Work
Related Work
Related Work
Related Work

Satisfiability Modulo Theories (SMT)-based verification

Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver

Konstantin Woritz  Dong Woon Emna Tolk
Michael D. Ernst  Arvind Krishnamurthy  Zachary Tatlock
University of Washington, USA
(satisfaction, issues, issues, interest, interest, interest@us华盛顿.edu)

A General Approach to Network Configuration Verification

Ryan Beckert
Princeton University
Ratul Mahajan
Microsoft Research & International
Aarti Gupta
Princeton University
David Walker
Princeton University

simulation-based verification

Plankton: Scalable network configuration verification through model checking
University of Illinois at Urbana-Champaign

Tiramisu: Fast Multilayer Network Verification
Anishkumar Abhishek, Aaron Gember-Jacobson, Aditya Akella
University of Wisconsin - Madison, Columbia University

modular SMT-based verification

LIGHTYEAR: Using Modularity to Scale BGP Control Plane Verification
Alan Tang
University of California, Los Angeles
Todd Millstein
UCLA / Investments
Ryan Beckert
Microsoft
Karthik Jayaraman
Microsoft

Kirigami, the Verifiable Art of Network Cutting
Timothy Atkinson
Princeton University
Francesco, USA
Chris@cs.princeton.edu
Ryan Beckert
Microsoft
Redmond, USA
rbeckert@microsoft.com
Aarti Gupta
Princeton University
aarti@cs.princeton.edu
David Walker
Princeton University
Princeton, USA
djw@cs.princeton.edu
Challenges

finding the correct invariants

synchronous network semantics