Fast Automated What-if Analysis and Updates for Policy-Compliant Networks Even Under Failures

Stefan Schmid (TU Berlin)
Fast Automated What-if Analysis and Updates for Policy-Compliant Networks Even Under Failures

Stefan Schmid (TU Berlin)

In collaboration with Jiri Srba’s team at Aalborg University, Denmark
Networks Are Complex

• Many outages are due to network configuration errors = *human errors*

• Examples (see Ratul Mahajan’s NetVerify.Fun blog):
  – The December 2018 CenturyLink outage
  – The June 2020 T-Mobile outage
  – The July 2020 Cloudflare outage
  – The August 2020 CenturyLink outage
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in Datacenter

Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in Datacenter

Cluster with services that should be **globally reachable**.

Cluster with services that should be accessible **only internally**.

Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in Datacenter

X and Y *announce* to Internet what is from G* (prefix).

X and Y *block* what is from P*.

Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

X and Y *announce* to Internet what is from G* (prefix).

X and Y *block* what is from P*.

What can go wrong?

Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in Datacenter

X and Y *announce* to Internet what is from G* (prefix).

X and Y *block* what is from P*.

If link (G,X) fails and traffic from G is rerouted via Y and C to X: X announces (does not block) G and H as it comes from C. (Note: BGP.)

Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.
The Hope: Automation

- Can we automate the verification of the policy-compliance of configurations? Even under failures? Or even synthetize them?
The Hope: Automation

- Can we automate the verification of the policy-compliance of configurations? Even under failures? Or even synthetize them?

- A main challenge: should be fast as network configurations are not only complex for humans but also computers (many problems PSPACE-hard).
Roadmap

• **A Static Problem:** Policy Compliance
  *Under Failures*
  
  – AalWiNes: Fast Automated *What-if Analysis*

• **A Dynamic Problem:** *Scheduling*
  *Consistent Network Updates*
  
  – Latte and quantitative extensions (PODC 2015, ICALP 2018, PERFORMANCE 2021)
Roadmap

• **A Static Problem:** Policy Compliance
  *Under Failures*
  And SR...

• **A Dynamic Problem:** Scheduling
  Consistent Network *Updates*
    - Latte and quantitative extensions (PODC 2015, ICALP 2018, PERFORMANCE 2021)
How (MPLS) Networks Work

- Forwarding based on **top label** of label **stack**

![Diagram showing network with vertices labeled v1 to v8 and default routing of two flows]
How (MPLS) Networks Work

- Forwarding based on top label of label stack

Default routing of two flows
How (MPLS) Networks Work

- Forwarding is based on the top label of label stack

Default routing of two flows
Fast Reroute Around 1 Failure

- Forwarding based on **top label** of label **stack** (in packet header)

  ![Diagram](image)

  Default routing of two flows

- For failover: **push** and **pop** label

  ![Diagram](image)

  One failure: **push 30**: route around \((v_2,v_3)\)
Fast Reroute Around 1 Failure

- Forwarding based on **top label** of label **stack** (in packet header)

![Diagram]

- For failure, **push** and **pop** label

**If** \((v_2, v_3)\) failed, **push 30** and forward to \(v_6\).

**Default routing of two flows**

**One failure**: **push 30**: route around \((v_2, v_3)\)
Fast Reroute Around 1 Failure

- Forwarding based on top label of label stack (in packet header)

- For failure, push and pop label

- One failure: push 30: route around (v2,v3)

What about multiple link failures?
2 Failures: Push *Recursively*

- **Original Routing**
- **One failure**: push 30: route around \((v_2, v_3)\)
- **Two failures**: first push 30: route around \((v_2, v_3)\)
  - Push recursively 40: route around \((v_2, v_6)\)
2 Failures: Push *Recursively*

---

**Original Routing**

---

**One failure: push 30:** route around \((v_2,v_3)\)

---

But masking links one-by-one can be inefficient: \((v_7,v_3,v_8)\) could be shortcut to \((v_7,v_8)\).

---

**Push recursively 40:** route around \((v_2,v_6)\)
2 Failures: Push *Recursively*

More efficient but also more complex: Cisco does **not recommend** using this option!

But masking links one-by-one can be inefficient: \((v_7,v_3,v_8)\) could be shortcut to \((v_7,v_8)\).

**Push recursively** 40: route around \((v_2,v_6)\)
2 Failures: Push *Recursively*

More efficient but also more complex: Cisco does *not recommend* using this option!

Also note: due to push, *header size* may grow arbitrarily!

Push recursively 40:
route around \((v_2, v_6)\)

But masking links one-by-one can be inefficient:
\((v_7, v_3, v_8)\) could be shortcut to \((v_7, v_8)\).
Responsibilities of a Sysadmin

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Non-reachability:** Is it ensured that traffic originating from A never reaches B?
- **Waypoint ensurance:** Is it ensured that traffic from A to B is always routed via a node C (e.g., a firewall)?

Routers and switches store list of **forwarding rules**, and conditional **failover rules**.
Responsibilities of a Sysadmin

Sysadmin responsible for:

• **Reachability:** Can traffic from ingress port A reach egress port B?
Responsibilities of a Sysadmin

**Sysadmin** responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
Responsibilities of a Sysadmin

Sysadmin responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Policy**: Is it ensured that traffic from A to B never goes via C?

E.g. **NORDUnet**: no traffic via Iceland (expensive!).
Responsibilities of a Sysadmin

**Sysadmin** responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Policy**: Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement**: Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?
Sysadmin responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Policy**: Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement**: Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?

... and everything even under multiple failures?!
Responsibilities of a Sysadmin

**Sysadmin** responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Policy**: Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement**: Is it ensured that traffic from A to B is always routed via a node C (e.g., intrusion detection system or a firewall)?

... and everything even under multiple failures?!
Approach: Automation and Formal Methods

Router configurations (Cisco, Juniper, etc.)

Pushdown Automaton and Prefix Rewriting Systems

Compilation

\[
\begin{align*}
pX & \Rightarrow qXX \\
pX & \Rightarrow qYX \\
qY & \Rightarrow rYY \\
rY & \Rightarrow r \\
rX & \Rightarrow pX
\end{align*}
\]

Interpretation
Approach: Automation and Formal Methods

Compilation

Interpretation

Use cases: Sysadmin *issues queries* to test certain properties, or do it on a *regular basis* automatically!

Router *configurations* (Cisco, Juniper, etc.)

Pushdown Automaton and *Prefix Rewriting Systems*

What if...?!
AalWiNes

A tool for MPLS reachability analysis and visualization from:

- Aalborg University
- Department of Computer Science
- University of Vienna Communication Technologies Group

Have a look at the Tool Website & Tool and query language documentation

Query: regular expression

Witness

Dozens of networks

Online demo: https://demo.aalwines.cs.aau.dk/
Source code: https://github.com/DEIS-Tools/AalWiNes
Example

Can traffic starting with [] go through s5, under up to k=2 failures?

Query: 3 regular expressions (initial and final header, route)
k=2 [] s1 >> s5 >> s7 []

YES
(Polynomial time!)
Why AalWiNes is Fast (Polytime): Automata Theory

• For fast verification, we can use the result by Büchi: the set of all reachable configurations of a pushdown automaton $a$ is regular set

• We hence simply use Nondeterministic Finite Automata (NFAs) when reasoning about the pushdown automata

• The resulting regular operations are all polynomial time
Case Study: NORDUnet

- Regional service provider
- **24 MPLS routers** geographically distributed across several countries
- Running **Juniper** operating system
- More than **30,000 labels**
- Ca. **1 million** forwarding rules in our model
- For most queries of operators: answer **within seconds**
Generalizes to Quantitative Properties

• AalWiNes can also be used to test **quantitative properties**

• If query is satisfied, find trace that minimizes:
  • **Hops**
  • Latency (based on a latency value per link)
  • Tunnels

• Approach: **weighted** pushdown automata
  • Fast *poly-time algorithms* exist also for weighted pushdown automata (area of dataflow analysis)
  • Indeed, experiments show: **acceptable overhead** of weighted (quantitative) analysis
Roadmap

• **A Static Problem**: Policy Compliance
  
  *Under Failures*
  

• **A Dynamic Problem**: Scheduling
  
  Consistent Network *Updates*
  
  – Latte and quantitative extensions (PODC 2015, ICALP 2018, PERFORMANCE 2021)
More Adaptable Networks

• Automation and programmability also enables networks to be more adaptable

• Attractive for:
  – Fine-grained traffic engineering (e.g., at Google)
  – Accounting for changes in the demand (spatio-temporal structure)
  – Security policy changes
  – Service relocation
  – Maintenance work
  – Link/node failures
  – …
Introduces a New Challenge: Scheduling Updates

**Invariant:** Traffic from untrusted hosts to trusted hosts via **firewall**!
Introduces a New Challenge: Scheduling Updates

Controller Platform

Invariant: Traffic from untrusted hosts to trusted hosts via firewall!
Introduces a New Challenge: Scheduling Updates

Invariant: Traffic from untrusted hosts to trusted hosts via firewall!
Latte: Synthesis of Shortest Consistent Update Schedules

• Much work on the design of efficient algorithms for consistent network updates

• Our goal: automated synthesis of fast updates accounting for temporal properties
  – E.g., different packet types have different requirements and processing times
  – Builds upon NetSynth (gives fixed update order)

• A classic tool to reason about asynchronous distributed systems: petri nets
  – Configurations: tokens located at places

• Our extension: Timed-Arc Colored Petri Nets (TACPN)
  – Tokens also contain: color information (e.g., different packet types) and time information (e.g., modeling age)
  – Places and input arcs have time constraints for each color
Latte: Synthesis of Shortest Consistent Update Schedules

- Much work on the design of efficient algorithms for consistent network updates

- Our goal: automated synthesis of fast updates accounting for temporal properties
  - E.g., different packet types have different requirements and processing times
  - Builds upon NetSynth (gives fixed update order)

- A classic tool to reason about asynchronous distributed systems: petri nets
  - Configurations: tokens located at places

- Our extension: Timed-Arc Colored Petri Nets (TACPN)
  - Tokens also contain: color information (e.g., different packet types) and time information (e.g., modeling age)
  - Places and input arcs have time constraints for each color
Example: Encoding Network Updates in TACPNs

1. Gadget to inject packets:

Packets can be of different types (timings): colors

Initially: token at this place

Jump to place $S_0$ and generate packet of arbitrary type
Example: Encoding Network Updates in TACPNs

2. Gadget to model switches:

- If token up here: packets go old path.
- If token down here: switch updated to new path.
Example: Encoding Network Updates in TACPNs

2 Gadget to model switches:

- If token up here: packets go old path
- Different timing constraints for packets
- If token down here: switch updated to new path
Example: Encoding Network Updates in TACPNs

3 Gadget to model switch update:

How to change between initial and final switch configuration

Starting here, the update can take time between min and max
Example: Encoding NetworkUpdates in TACPNs

Connecting the pieces: initialization of update sequence for all $n$ switches

Constants: $M_{ax} = 250000$

After updating Switch $S_1$ (delay $C_1$), go to Switch $S_2$, etc.
Analysis

The constructed nets can be analyzed efficiently via their unfolding into existing timed-arc Petri nets. Preserves bisimilarity!
Improved Latency of Update Schedules

- Network topologies from the Topology Zoo
- Experiments run on a 64-bit Ubuntu 18.04 laptop

Compared to conservative delays as produced by NetSynth: over 90% improvement.

<table>
<thead>
<tr>
<th>Network</th>
<th>Route length</th>
<th>Verification time [s]</th>
<th>Default update time [s]</th>
<th>Optimized update time [s]</th>
<th>Improvement [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tlex</td>
<td>4</td>
<td>0.74</td>
<td>3.58</td>
<td>0.25</td>
<td>92.30%</td>
</tr>
<tr>
<td>HiberniaIreland</td>
<td>5</td>
<td>1.02</td>
<td>6.05</td>
<td>0.28</td>
<td>95.50%</td>
</tr>
<tr>
<td>Hornet</td>
<td>6</td>
<td>1.42</td>
<td>9.08</td>
<td>0.28</td>
<td>96.97%</td>
</tr>
<tr>
<td>UniC</td>
<td>7</td>
<td>1.49</td>
<td>12.65</td>
<td>0.28</td>
<td>97.83%</td>
</tr>
<tr>
<td>Oxford</td>
<td>8</td>
<td>2.02</td>
<td>16.78</td>
<td>0.28</td>
<td>98.36%</td>
</tr>
<tr>
<td>Xerox</td>
<td>10</td>
<td>5.86</td>
<td>26.68</td>
<td>0.28</td>
<td>98.97%</td>
</tr>
<tr>
<td>Sunet</td>
<td>11</td>
<td>10.23</td>
<td>32.45</td>
<td>0.28</td>
<td>99.15%</td>
</tr>
<tr>
<td>SwitchL3</td>
<td>12</td>
<td>18.88</td>
<td>38.78</td>
<td>0.28</td>
<td>99.29%</td>
</tr>
<tr>
<td>P3net</td>
<td>14</td>
<td>89.67</td>
<td>53.01</td>
<td>0.28</td>
<td>99.48%</td>
</tr>
<tr>
<td>Unet</td>
<td>15</td>
<td>211.86</td>
<td>61.05</td>
<td>0.28</td>
<td>99.55%</td>
</tr>
<tr>
<td>Renater2010</td>
<td>16</td>
<td>480.52</td>
<td>69.58</td>
<td>0.28</td>
<td>99.60%</td>
</tr>
<tr>
<td>Missouri</td>
<td>25</td>
<td>timeout</td>
<td>171.05</td>
<td>67.10</td>
<td>60.77%</td>
</tr>
<tr>
<td>Syringa</td>
<td>35</td>
<td>timeout</td>
<td>336.05</td>
<td>295.35</td>
<td>12.11%</td>
</tr>
<tr>
<td>Vt/Wavenet2011</td>
<td>35</td>
<td>timeout</td>
<td>336.06</td>
<td>295.35</td>
<td>12.11%</td>
</tr>
</tbody>
</table>
Improved Latency of Update Schedules

- Network topologies from the Topology Zoo
- Experiments run on a 64-bit Ubuntu 18.04 laptop

Up to route length 16, optimal update time can be computed.

Compared to conservative delays as produced by NetSynth: over 90% improvement.

Too many updates concurrently: could be tackled with static analysis (future work).
Conclusion

• Finally: networks are moving from manual to more automated operations

• Supported by emerging programmable networks and their solid theoretical foundations and languages

• Automata-theoretical approaches can be used to perform fast what-if analysis of the policy compliance (e.g., P-Rex, *AalWiNes*, etc.)
  – E.g., MPLS networks, but also Segment Routing networks

• More adaptive network operations further require tools for consistent network update scheduling (e.g., *Latte*)

• Current research focus on:
  – Accounting for *quantitative aspects*
  – Improving performance further *with AI*, without losing formal guarantees (e.g., configuration of CEGAR)
Further Reading

The AalWines project
https://aalwines.cs.aau.dk/

Netverify.fun

Toward Polynomial-Time Verification of Networks with Infinite State Spaces: An Automata-Theoretic Approach

With the increasing scale of communication networks, failures (e.g., link failures) are becoming the norm rather than the exception. Given the critical role such networks play for our digital society, it is important to...
References

**Resilient Capacity-Aware Routing**
Stefan Schmid, Nicolas Schnepp and Jiri Srba.

**AalWiNes: A Fast and Quantitative What-If Analysis Tool for MPLS Networks**
Peter Gjøl Jensen, Morten Konggaard, Dan Kristiansen, Stefan Schmid, Bernhard Clemens Schrenk, and Jiri Srba.
16th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Barcelona, Spain, December 2020.

**Latte: Improving the Latency of Transiently Consistent Network Update Schedules**
Mark Glavind, Niels Christensen, Jiri Srba, and Stefan Schmid.

**P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures**
14th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Heraklion/Crete, Greece, December 2018.

**Congestion-Free Rerouting of Flows on DAGs**
Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht.
45th International Colloquium on Automata, Languages, and Programming (ICALP), Prague, Czech Republic, July 2018.

**Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks**
Stefan Schmid and Jiri Srba.
37th IEEE Conference on Computer Communications (INFOCOM), Honolulu, Hawaii, USA, April 2018.

**DeepMPLS: Fast Analysis of MPLS Configurations Using Deep Learning**
Fabien Geyer and Stefan Schmid.
**IFIP Networking**, Warsaw, Poland, May 2019.
Questions?