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

AALBORG
UNIVERSITY

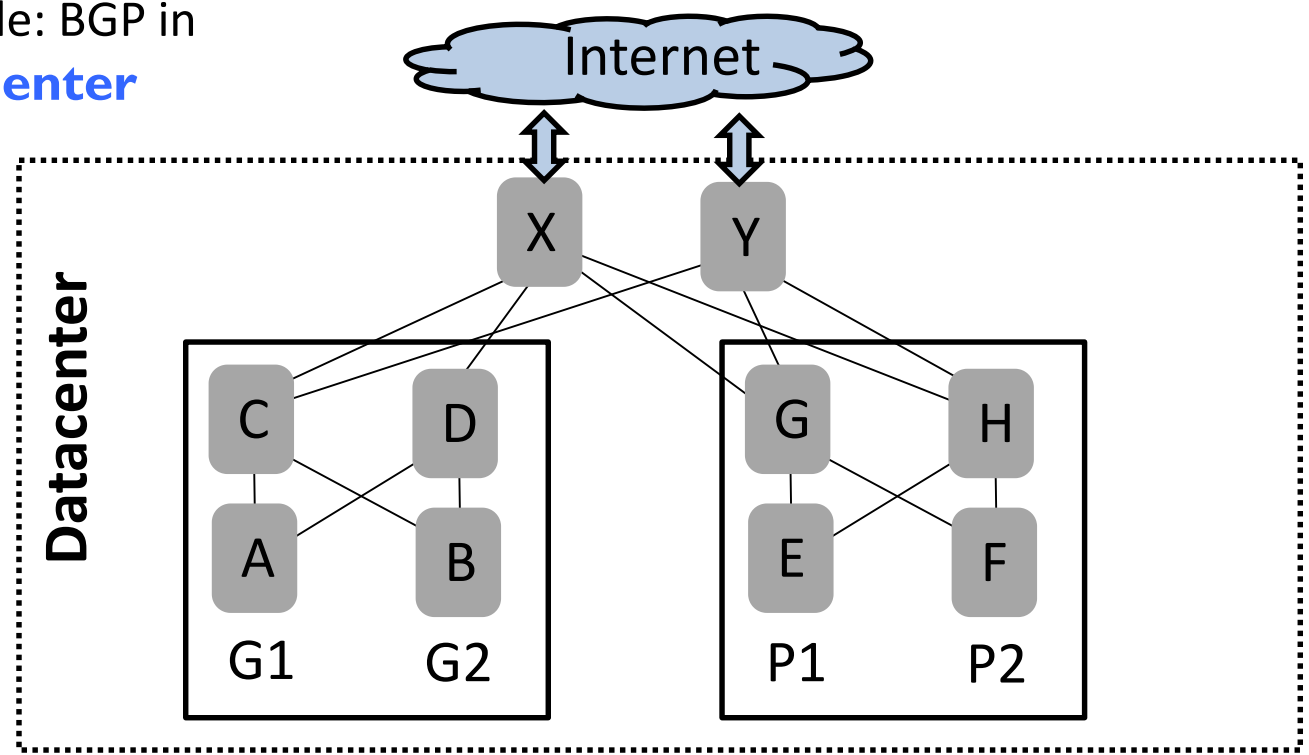
Networks Are Complex

- 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

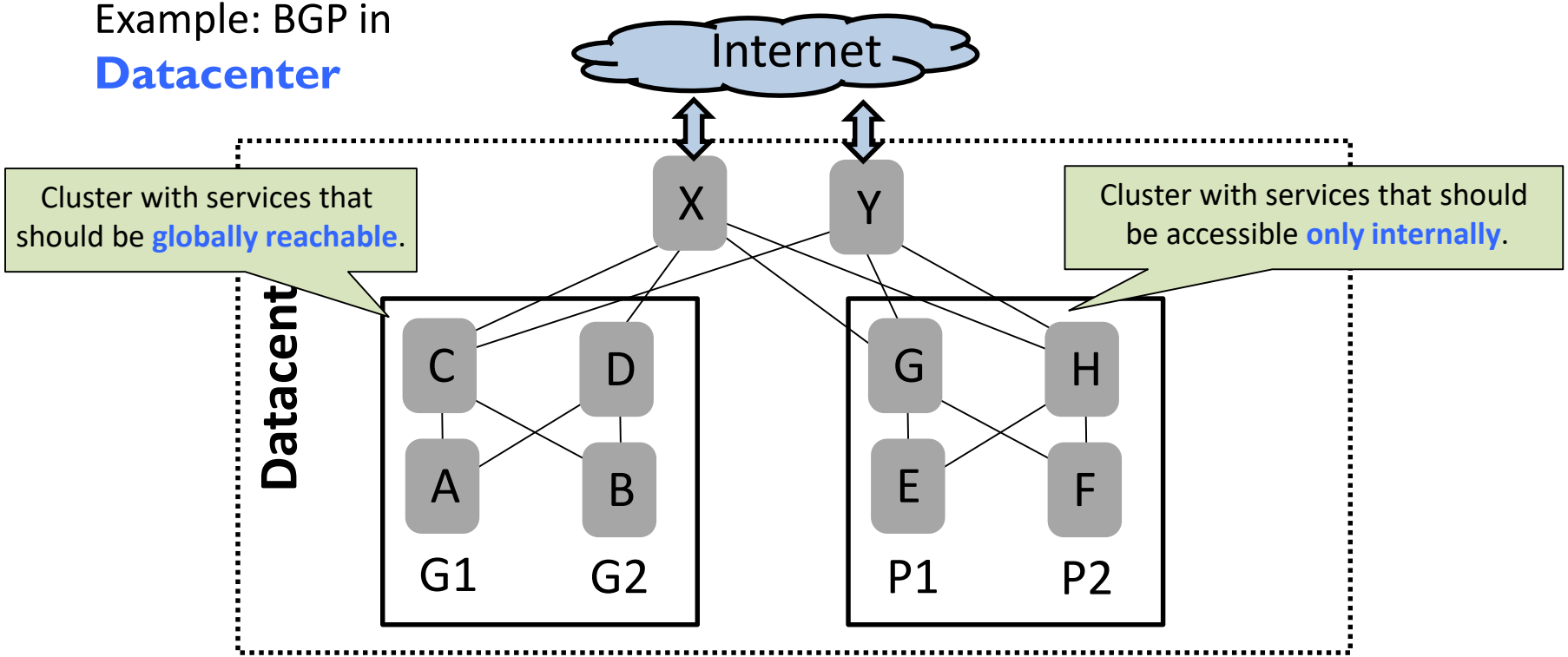
Example: BGP in **Datacenter**



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

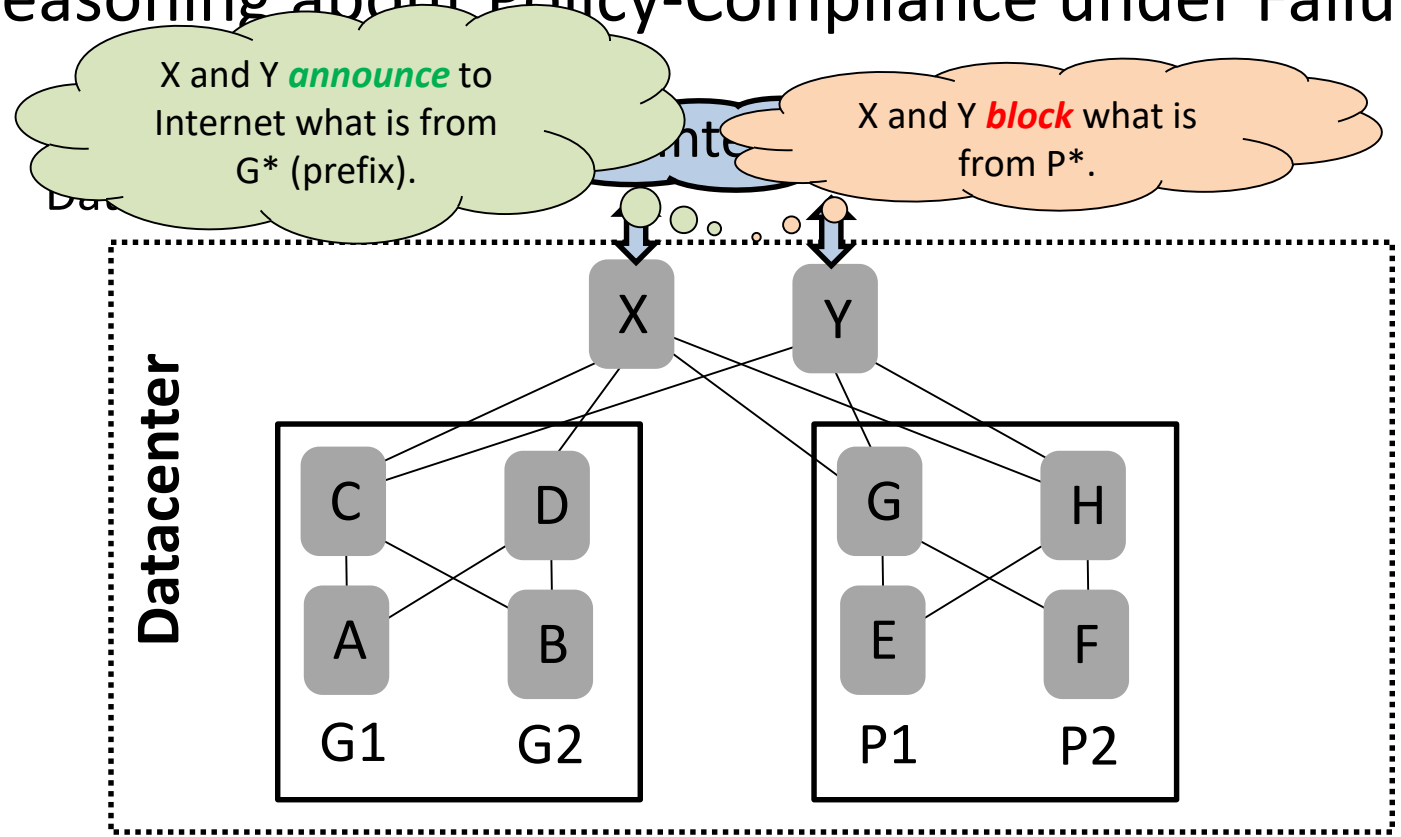
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in **Datacenter**



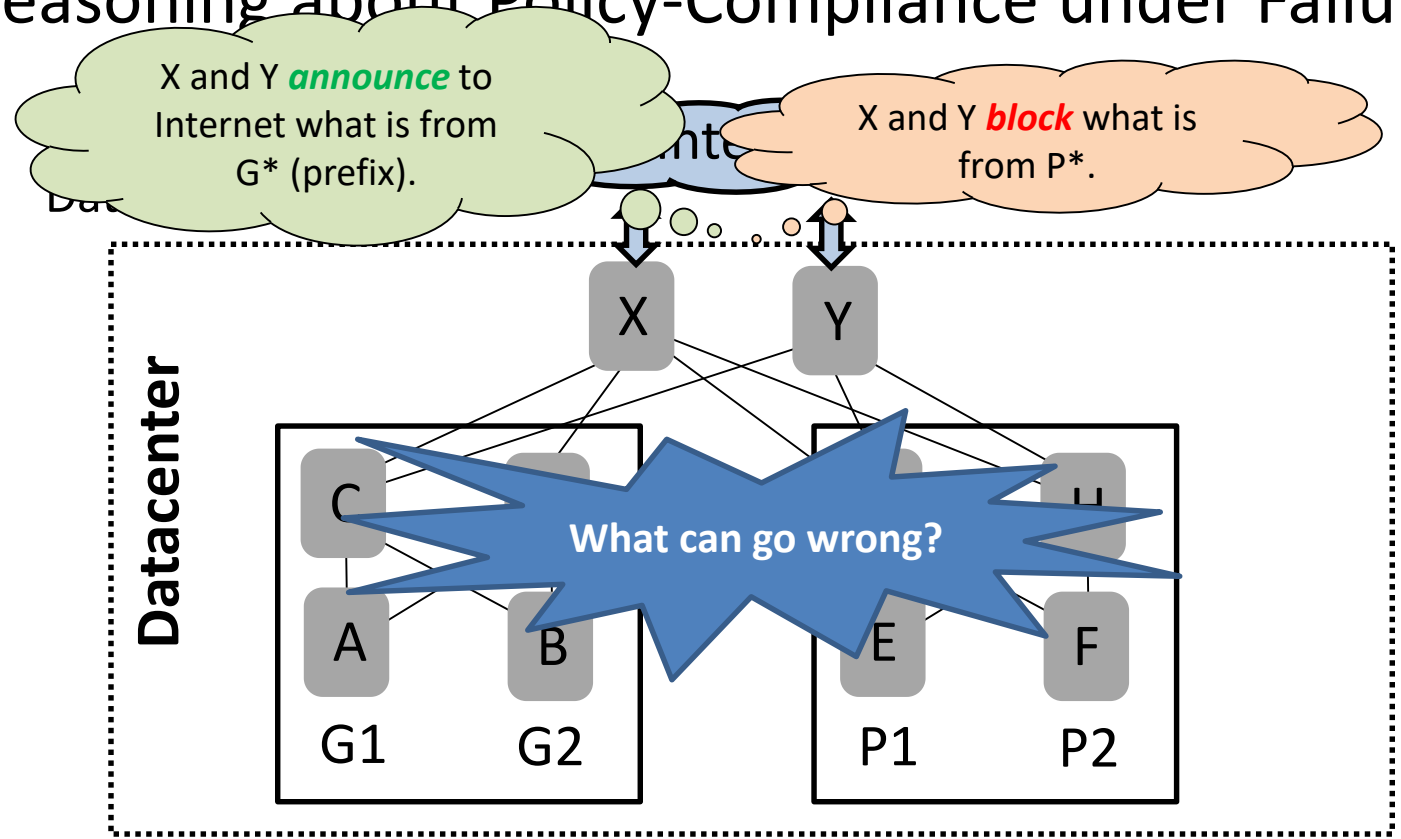
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



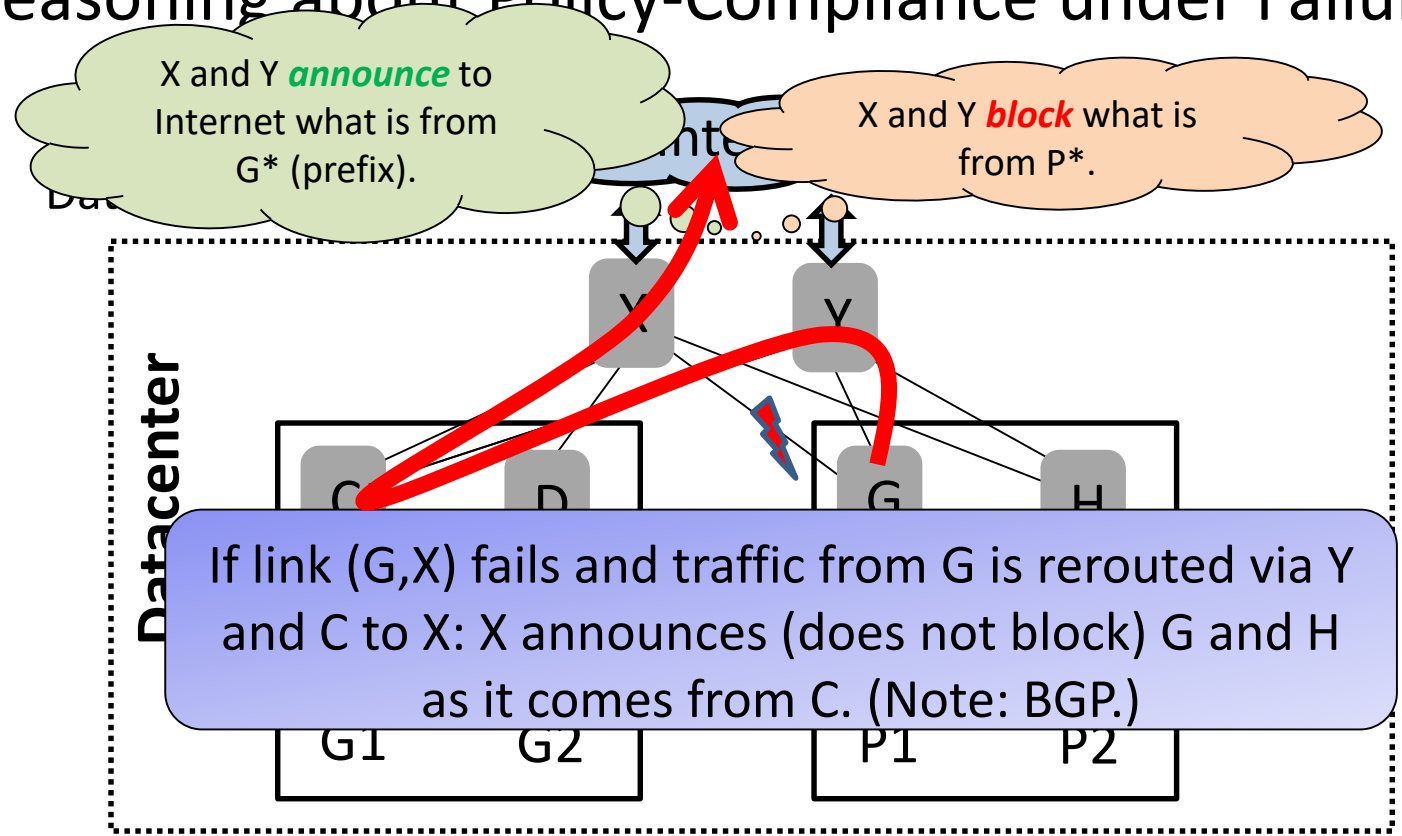
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



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

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



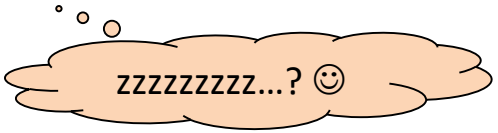
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

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** for MPLS Networks (INFOCOM 2018, ACM CoNEXT 2018, ACM CoNEXT 2019, TACAS 2021)

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

Roadmap

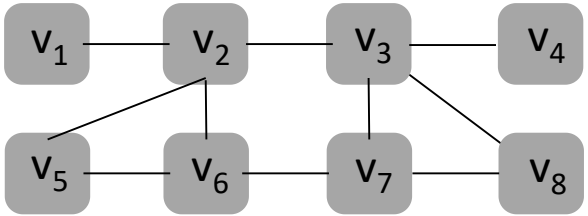
- **A Static Problem:** Policy Compliance *Under Failures*
  - AalWiNes: Fast Automated **What-if Analysis** for MPLS Networks (INFOCOM 2018, ACM CoNEXT 2018, ACM CoNEXT 2019, TACAS 2021)

And SR…

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

How (MPLS) Networks Work

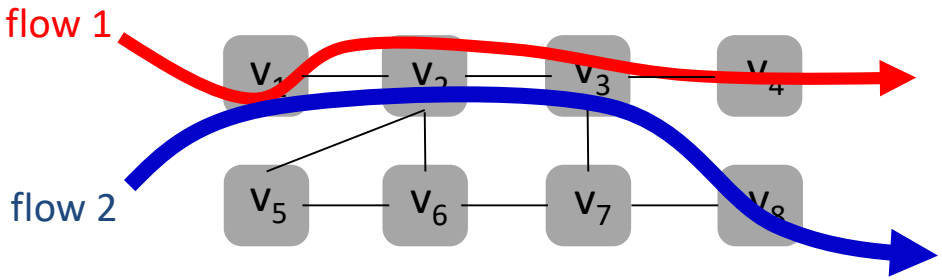
- Forwarding based on **top label** of label **stack**



Default routing of two flows

How (MPLS) Networks Work

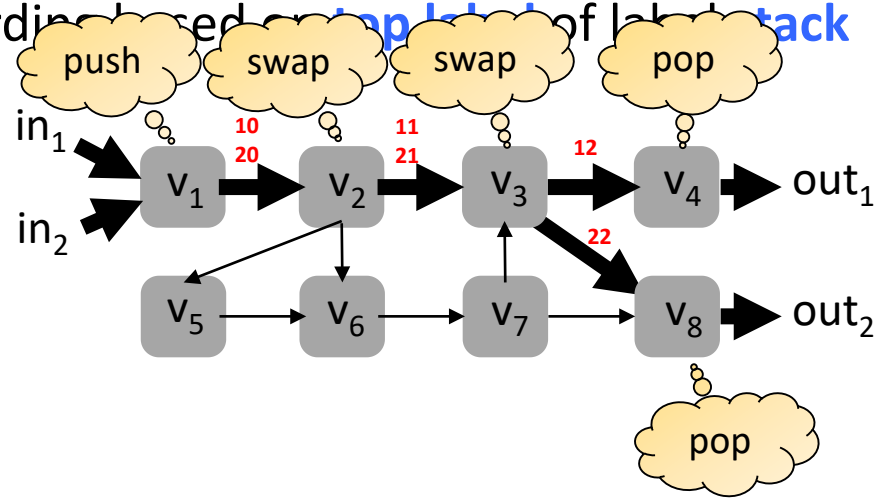
- Forwarding based on **top label** of label **stack**



Default routing of two flows

How (MPLS) Networks Work

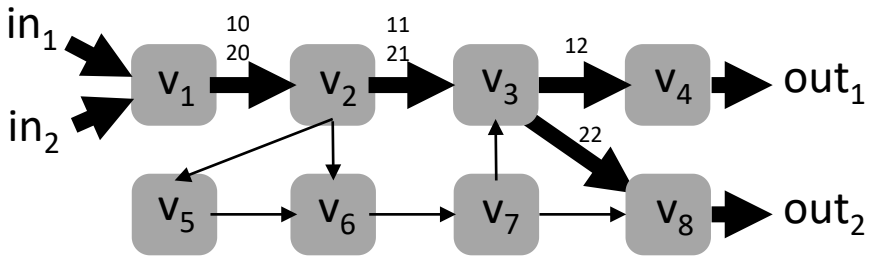
- Forwarding based on top label of label stack



Default routing of two flows

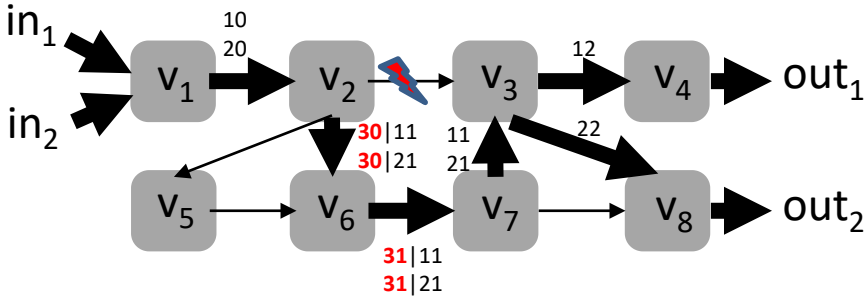
Fast Reroute Around *1 Failure*

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



Default routing of two flows

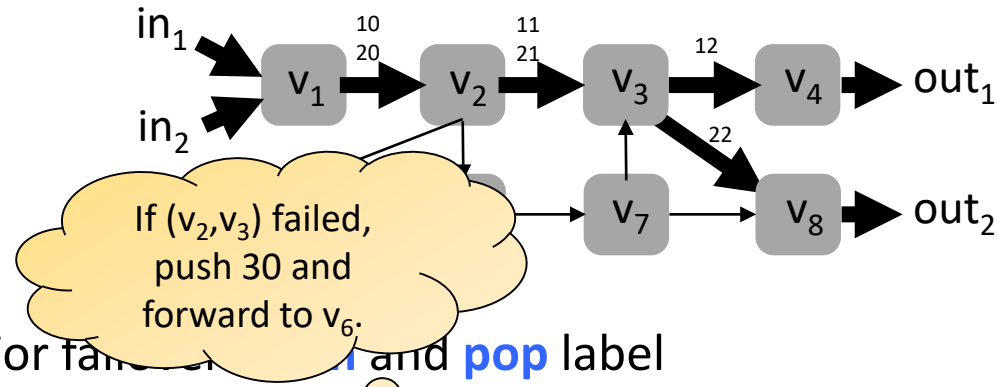
- For failover: **push** and **pop** label



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

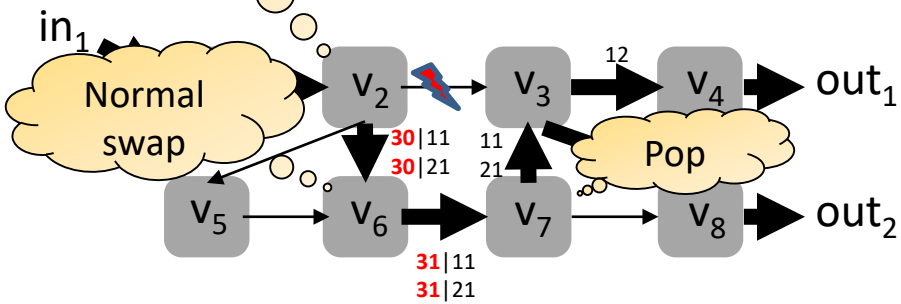
Fast Reroute Around *1 Failure*

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



Default routing of two flows

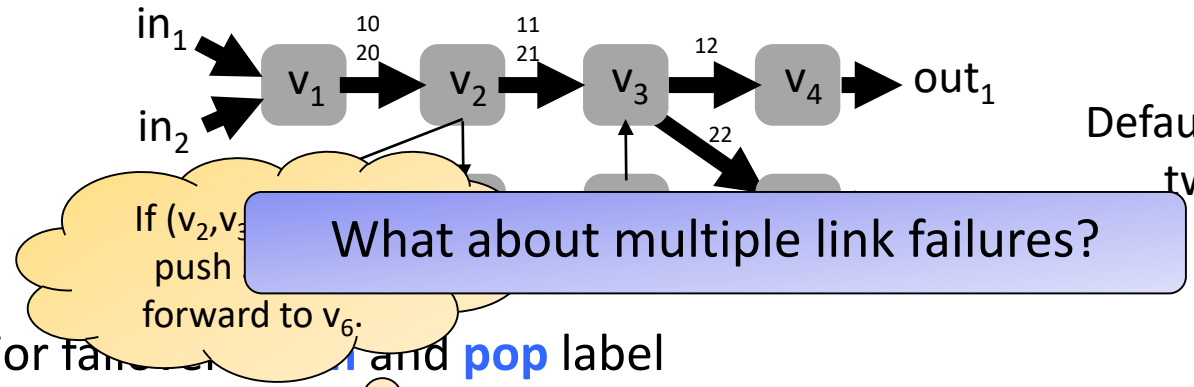
- For fai[illegible] and **pop** label



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

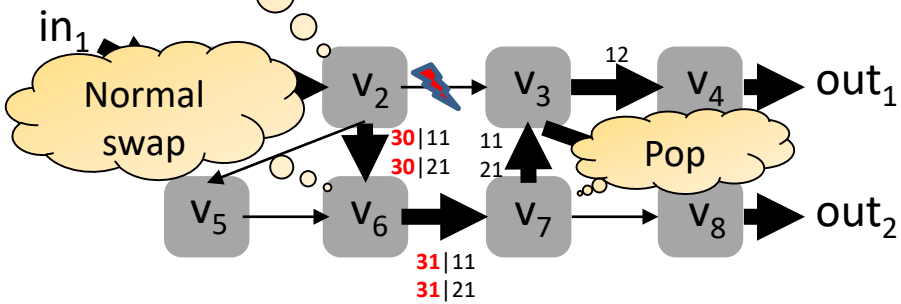
Fast Reroute Around *1 Failure*

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



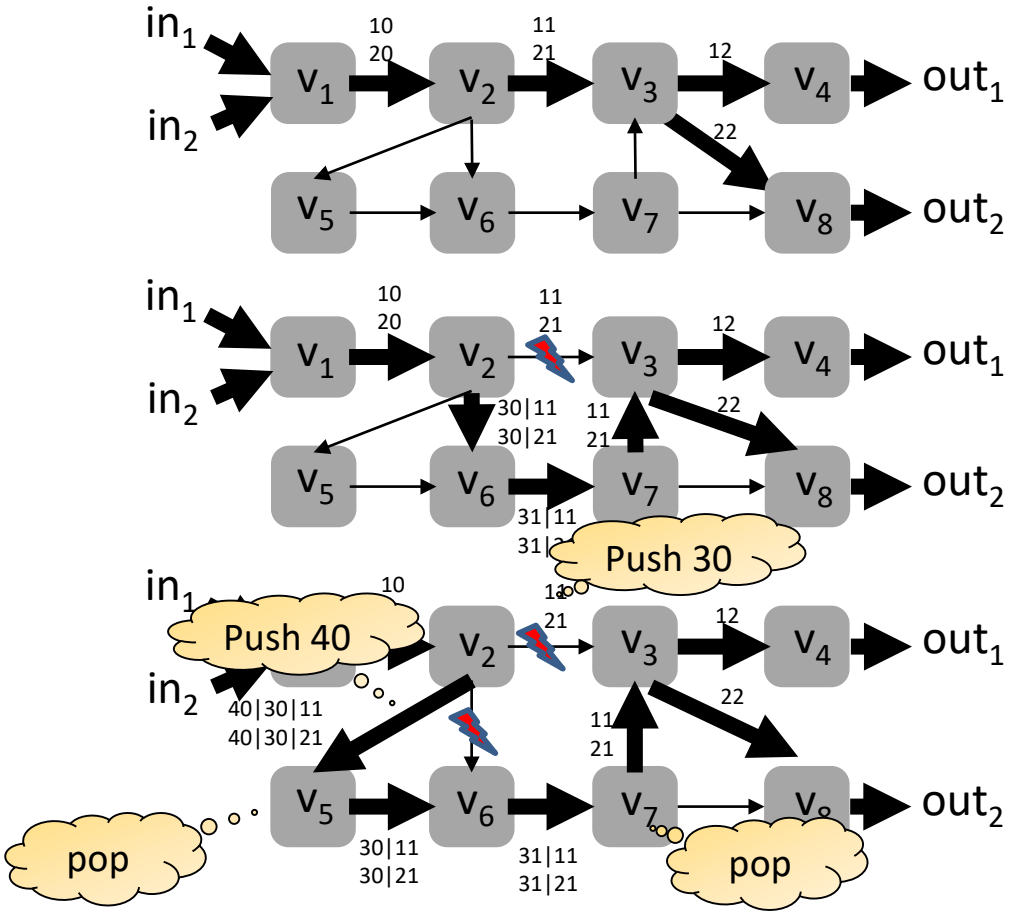
Default routing of two flows

- For fai and **pop** label



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

2 Failures: Push *Recursively*



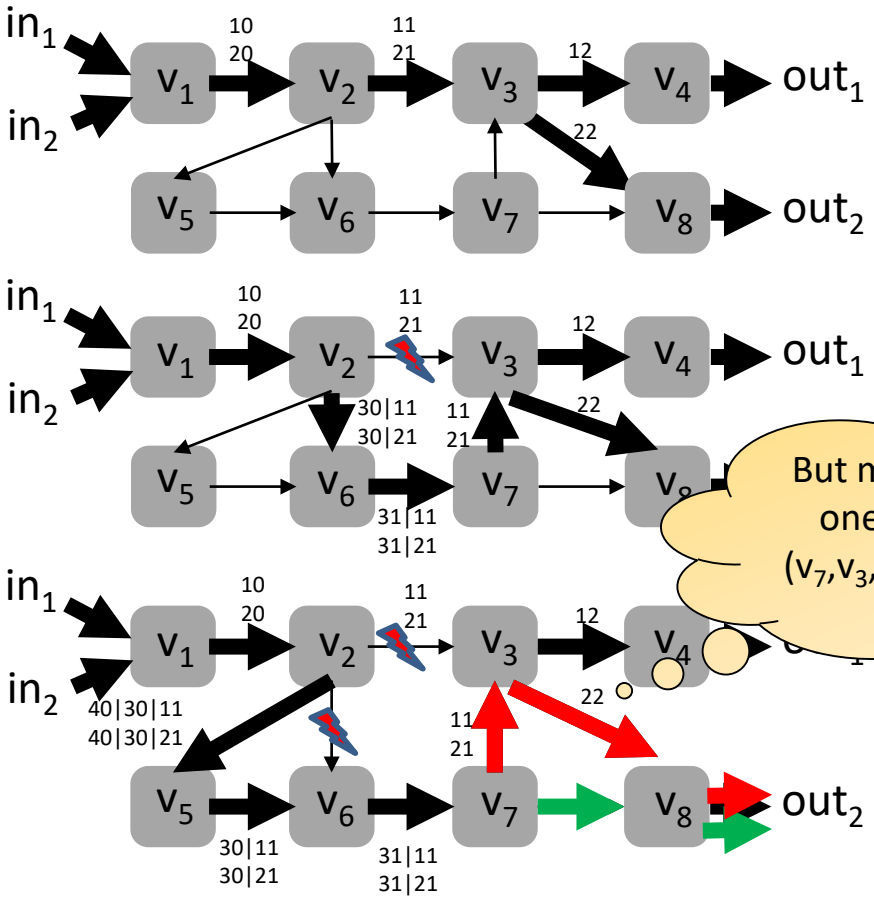
Original Routing

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

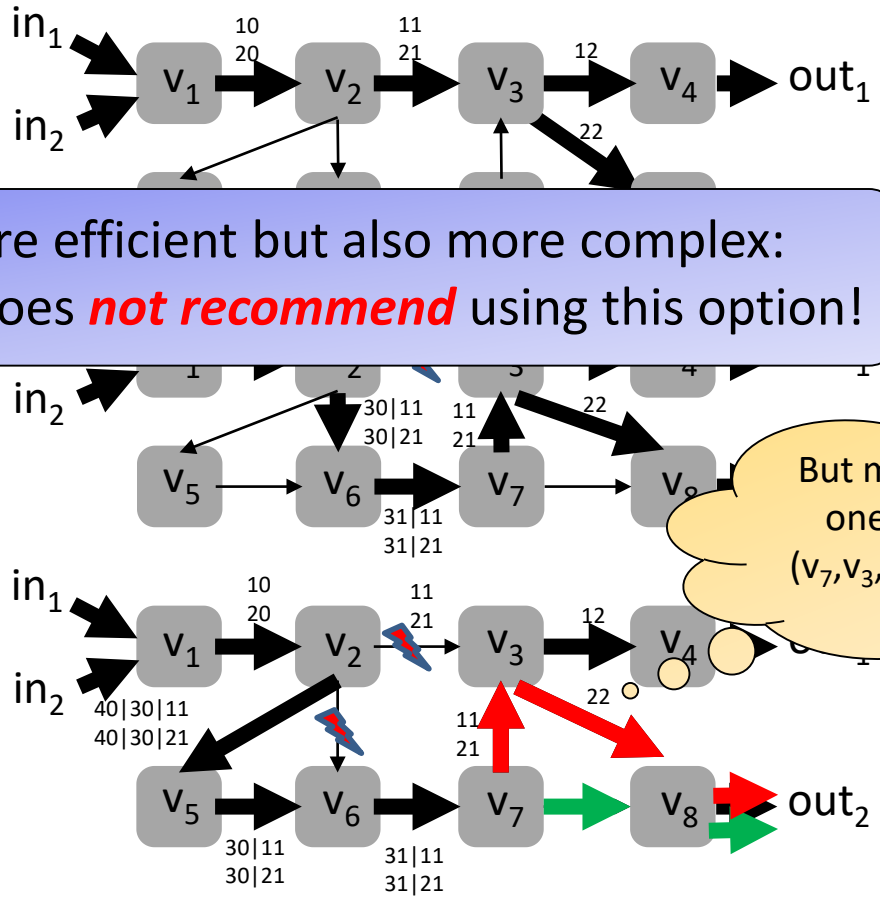
One failure: push 30:

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

…sh 30: route around (v_2,v_3)

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!

Original Routing

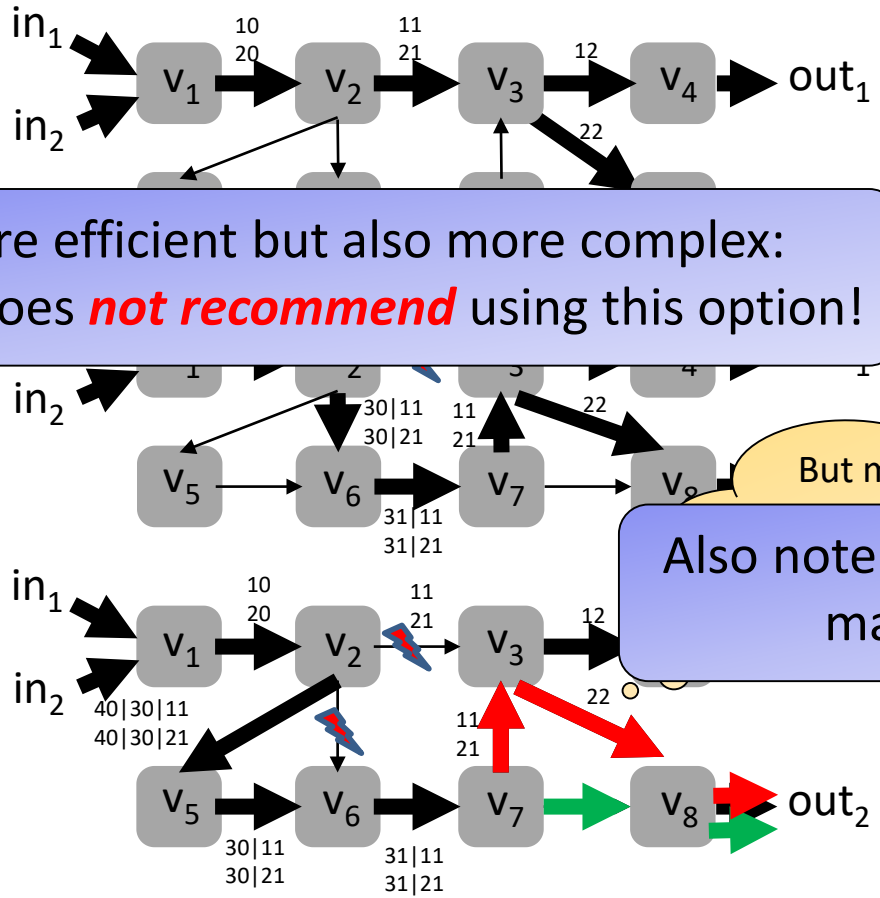
One failure: push 30:

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

…sh 30: route around (v_2,v_3)

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!

Original Routing

One failure: push 30:

But masking links one-by-

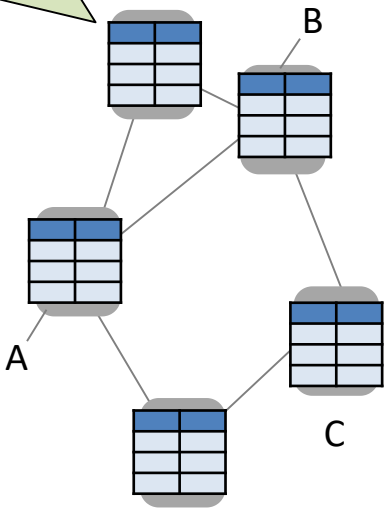
around (v_2,v_3)

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

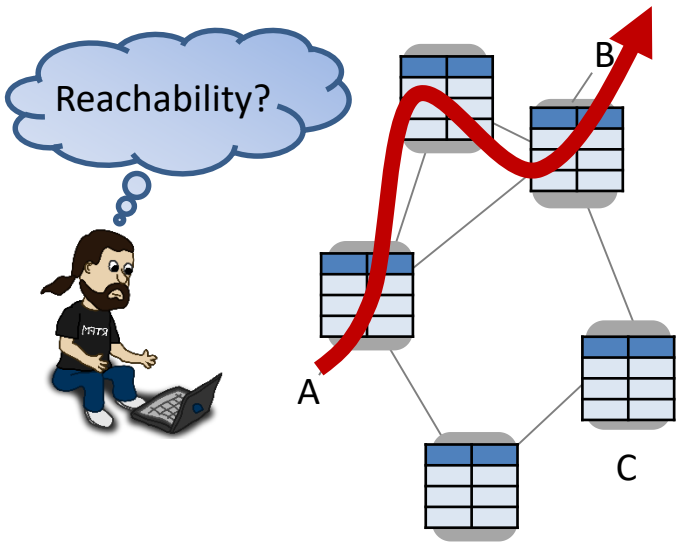
Push recursively 40:
route around (v_2,v_6)

Responsibilities of a Sysadmin

Routers and switches store list of forwarding rules, and conditional failover rules.



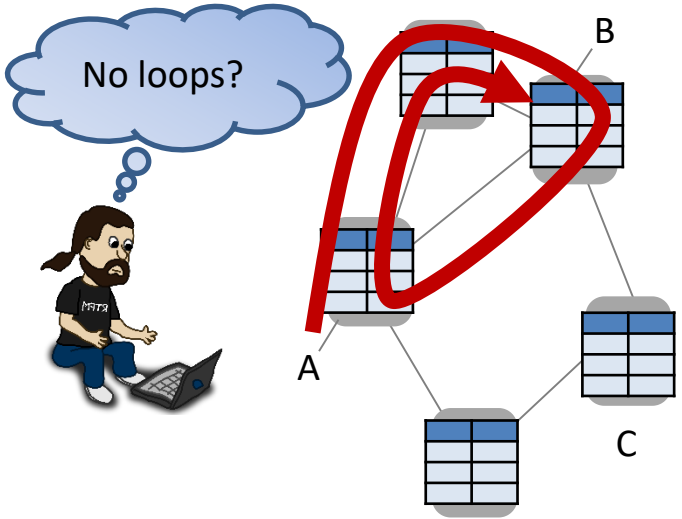
Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?

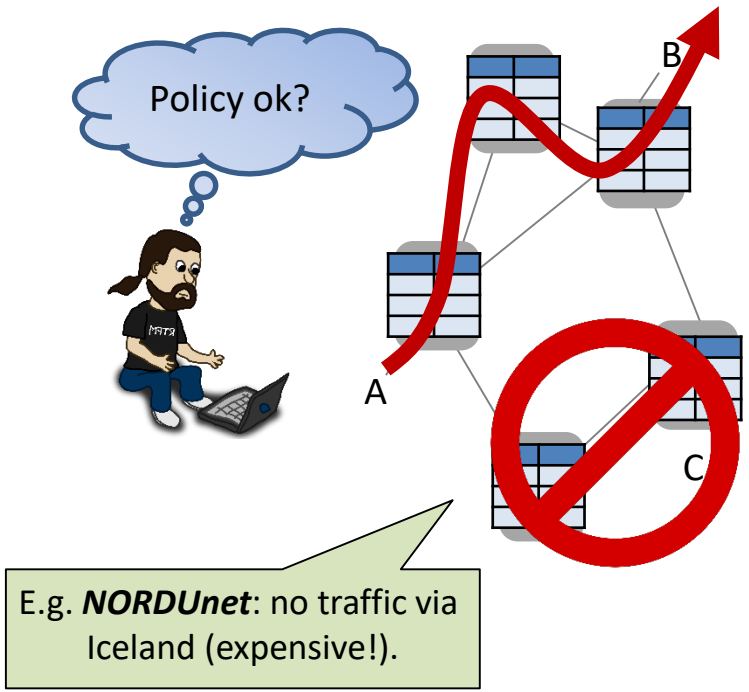
Responsibilities of a Sysadmin



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?

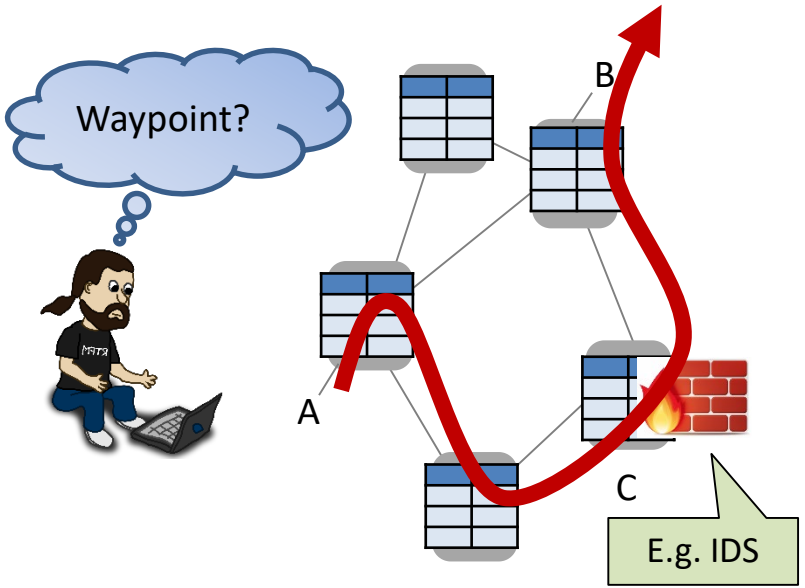
Responsibilities of a Sysadmin



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?

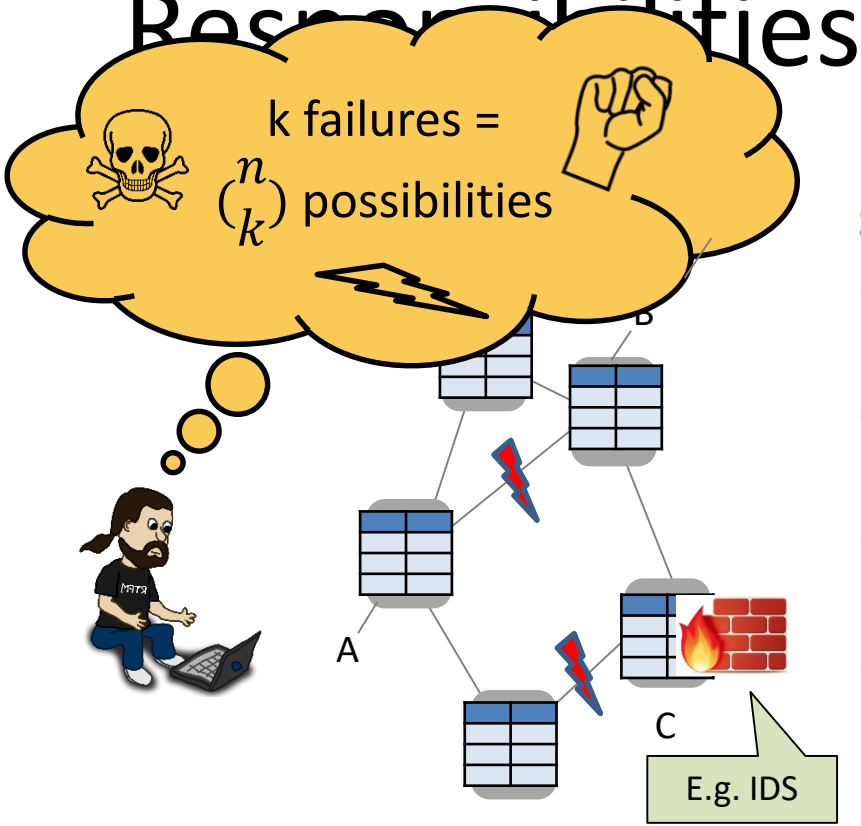
Responsibilities of a Sysadmin



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)?

Responsibilities of a Sysadmin

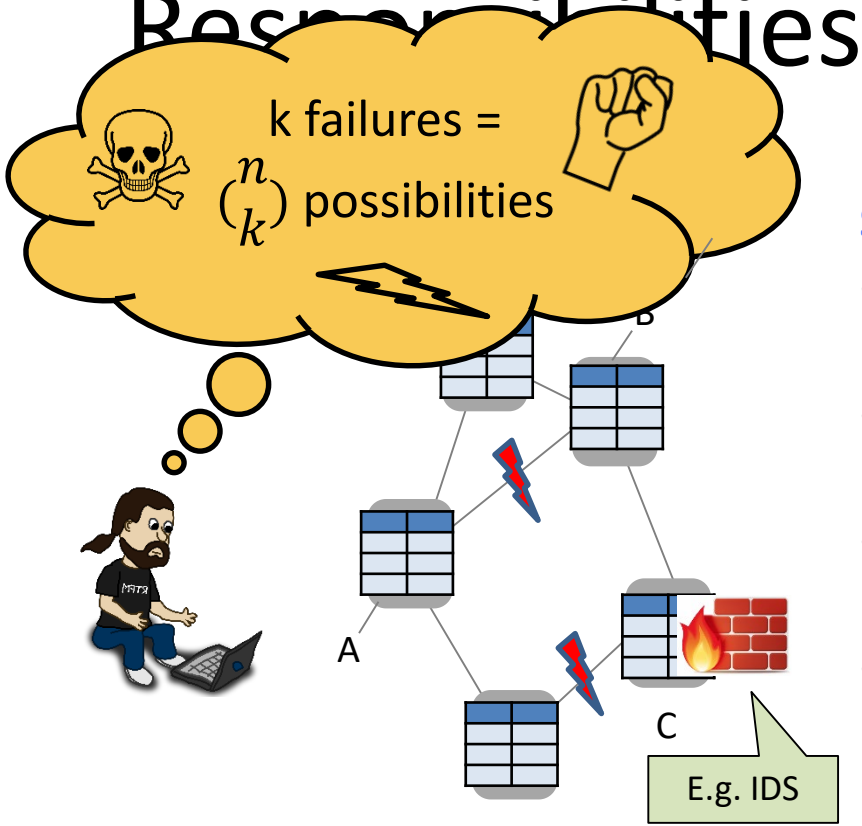


Sysadmin responsible for:

- **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?!

Responsibilities of a Sysadmin



Sysadmin responsible for:

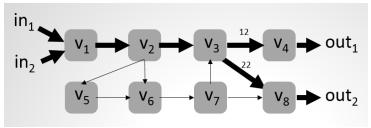
- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed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

… and everything even under multiple failures?!

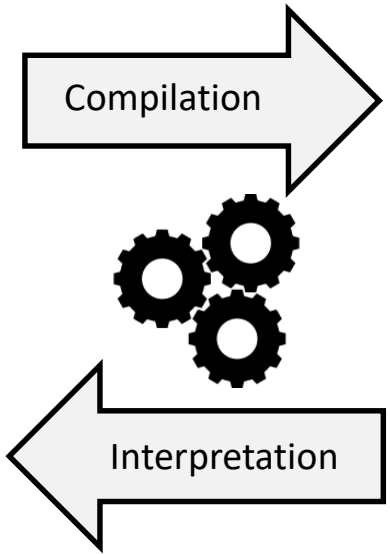
Generalization: service chaining!

Approach: Automation and Formal Methods

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$



$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

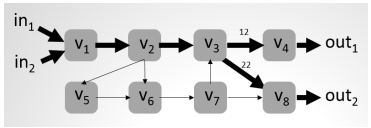
Router **configurations**
(Cisco, Juniper, etc.)

Pushdown Automaton and
Prefix Rewriting Systems

Approach: Automa… …ods

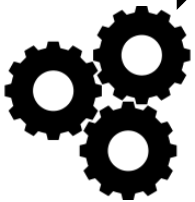
Use cases: Sysadmin ***issues queries*** to test certain properties, or do it on a ***regular basis*** automatically!

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

Compilation



Interpretation

$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

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

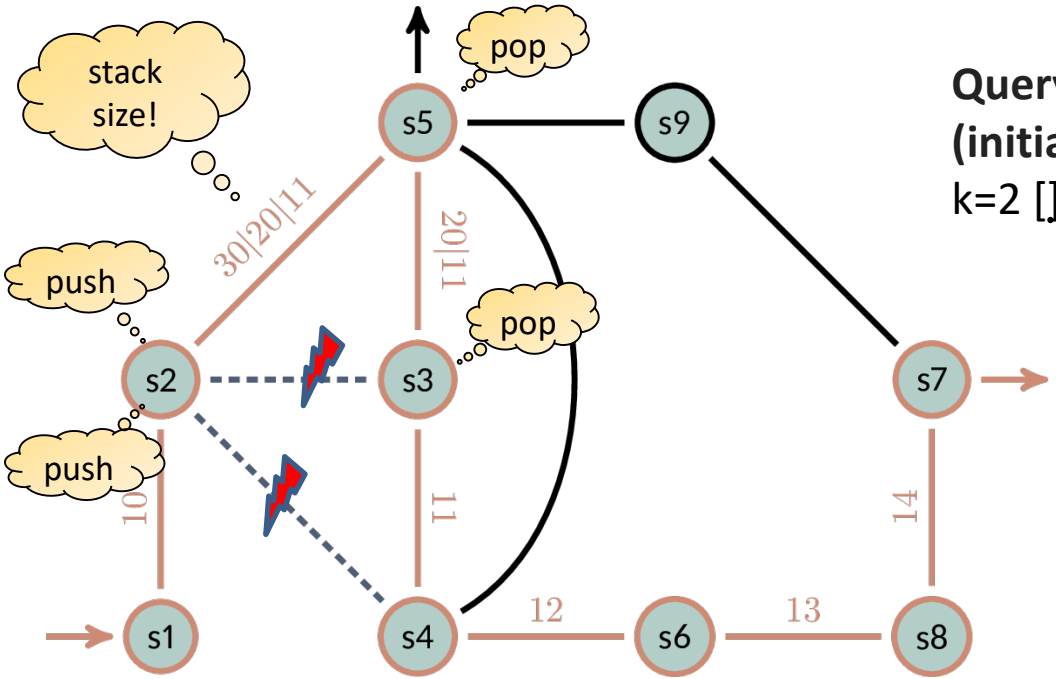
Pushdown Automaton and **Prefix Rewriting Systems**

AalWiNes

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 []

2 failures

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

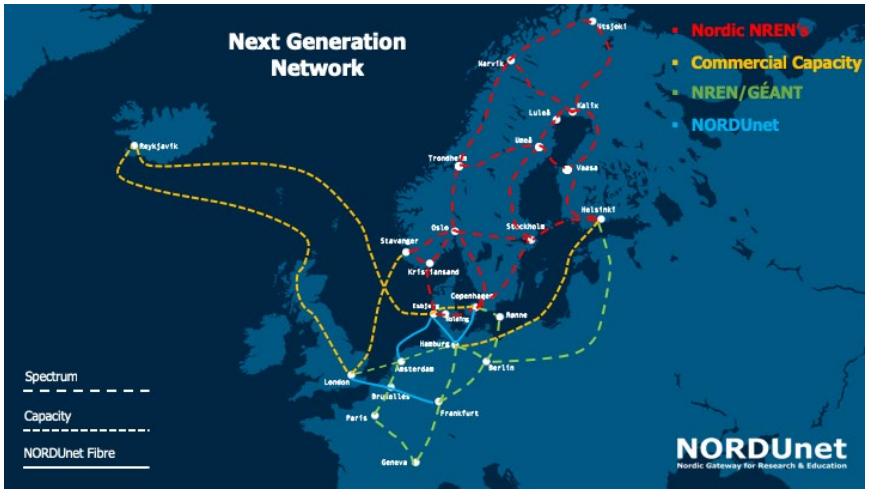
Julius Richard Büchi

1924-1984

Swiss logician

Case Study: NORDUnet

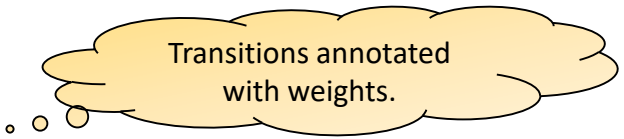
- Regional service provider
- **24 MPLS routers** geographically distributed across several countries
- Running **Juniper** operating system
- More than *30,000 labels*
- Ca. **1 million** forwarding rules in our model
- For most queries of operators: answer *within seconds*



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

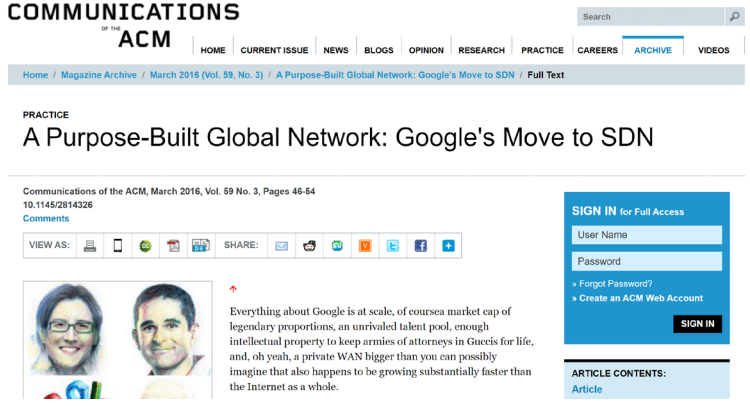
- **A Static Problem:** Policy Compliance *Under Failures*
  - AalWiNes: Fast Automated **What-if Analysis** for MPLS Networks (INFOCOM 2018, ACM CoNEXT 2018, ACM CoNEXT 2019, TACAS 2021)

- **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
  - …


COMMUNICATIONS of the ACM

Search

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | ARCHIVE | VIDEOS

Home / Magazine Archive / March 2016 (Vol. 59, No. 3) / A Purpose-Built Global Network: Google's Move to SDN / Full Text

PRACTICE

A Purpose-Built Global Network: Google's Move to SDN

Communications of the ACM, March 2016, Vol. 59 No. 3, Pages 46-54
10.1145/2814326
Comments

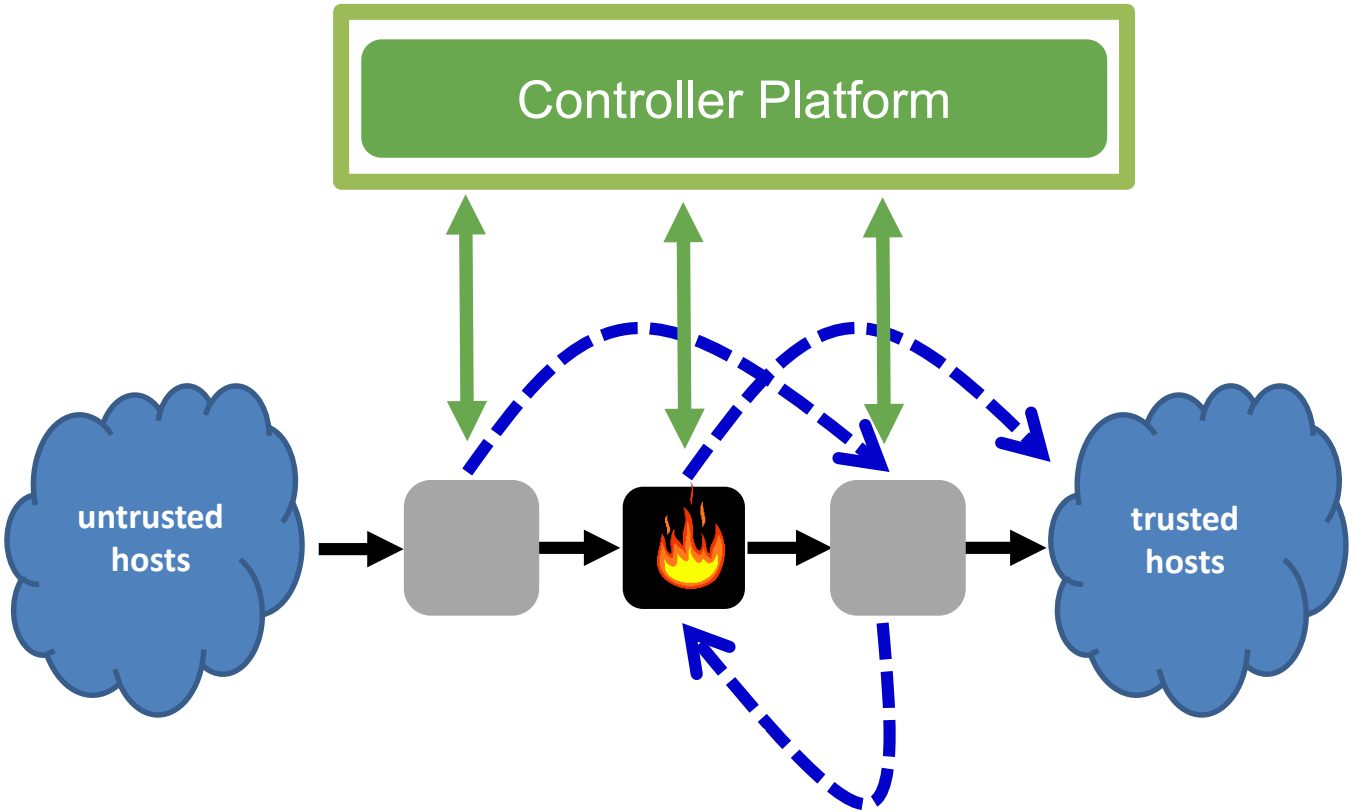
VIEW AS: SHARE:

SIGN IN for Full Access
User Name
Password
» Forgot Password?
» Create an ACM Web Account
SIGN IN

ARTICLE CONTENTS:
Article

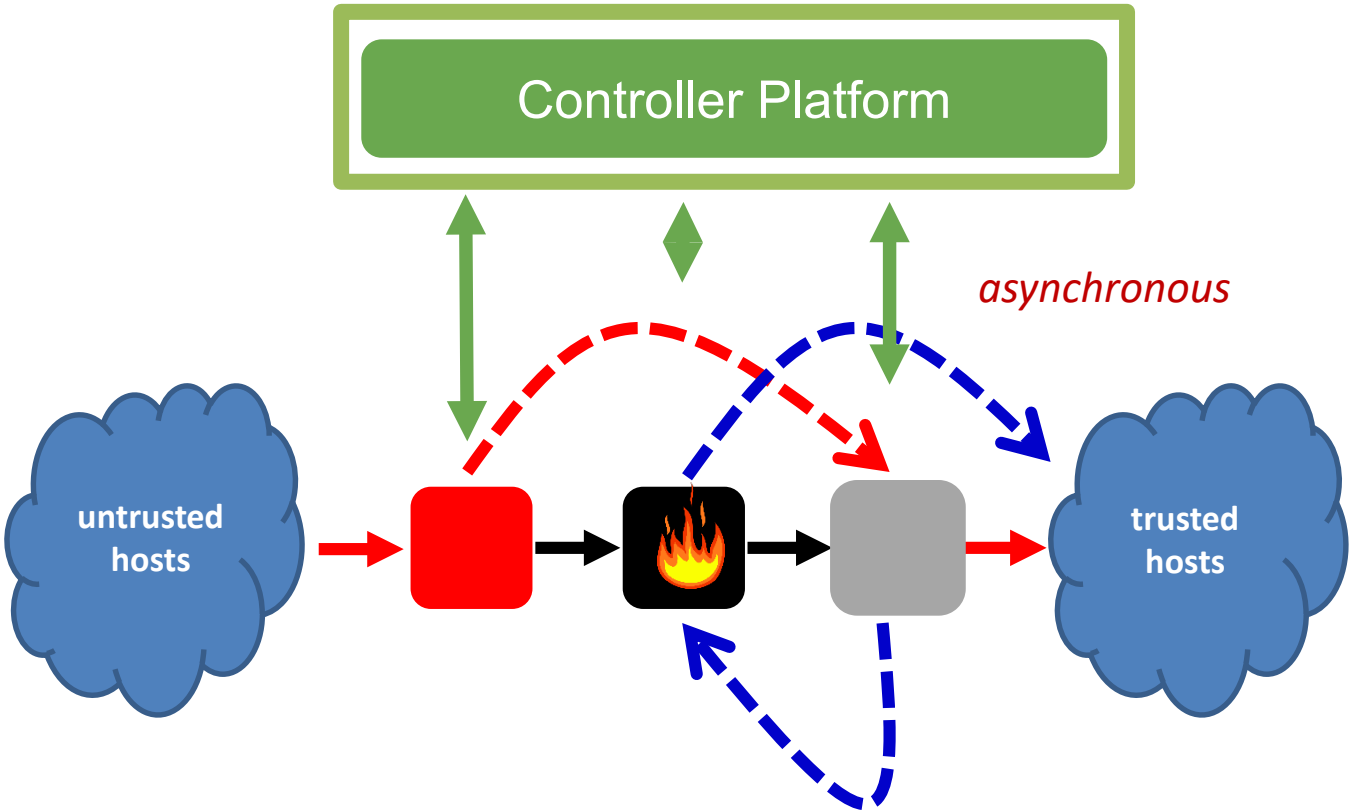
Everything about Google is at scale, of coursea market cap of legendary proportions, an unrivaled talent pool, enough intellectual property to keep armies of attorneys in Guccis for life, and, oh yeah, a private WAN bigger than you can possibly imagine that also happens to be growing substantially faster than the Internet as a whole.

Introduces a New Challenge: Scheduling Updates



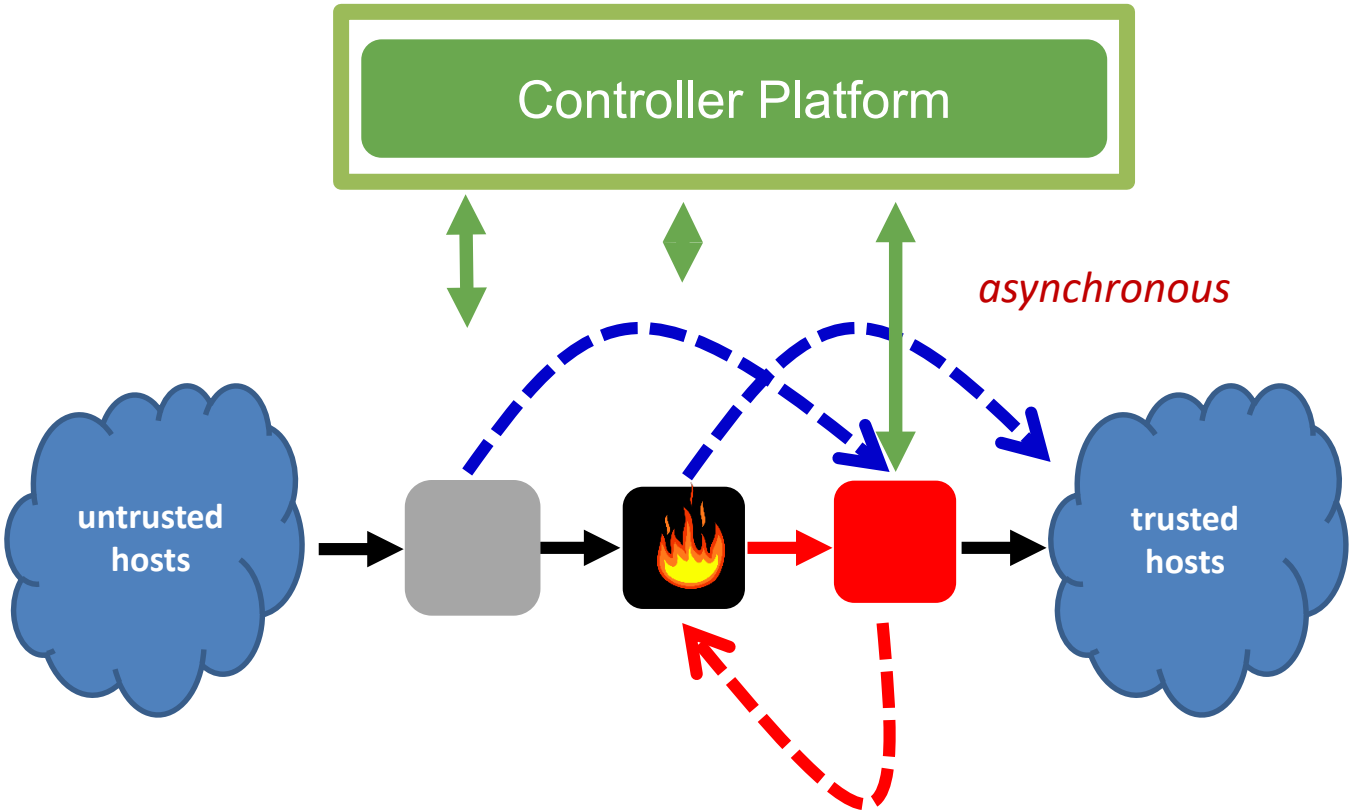
Invariant: Traffic from untrusted hosts to trusted hosts via **firewall**!

Introduces a New Challenge: Scheduling Updates



Invariant: Traffic from untrusted hosts to trusted hosts via **firewall**!

Introduces a New Challenge: Scheduling Updates



Invariant: Traffic from untrusted hosts to trusted hosts via **firewall**!

Latte: Synthesis of Shortest Consistent Update Schedules

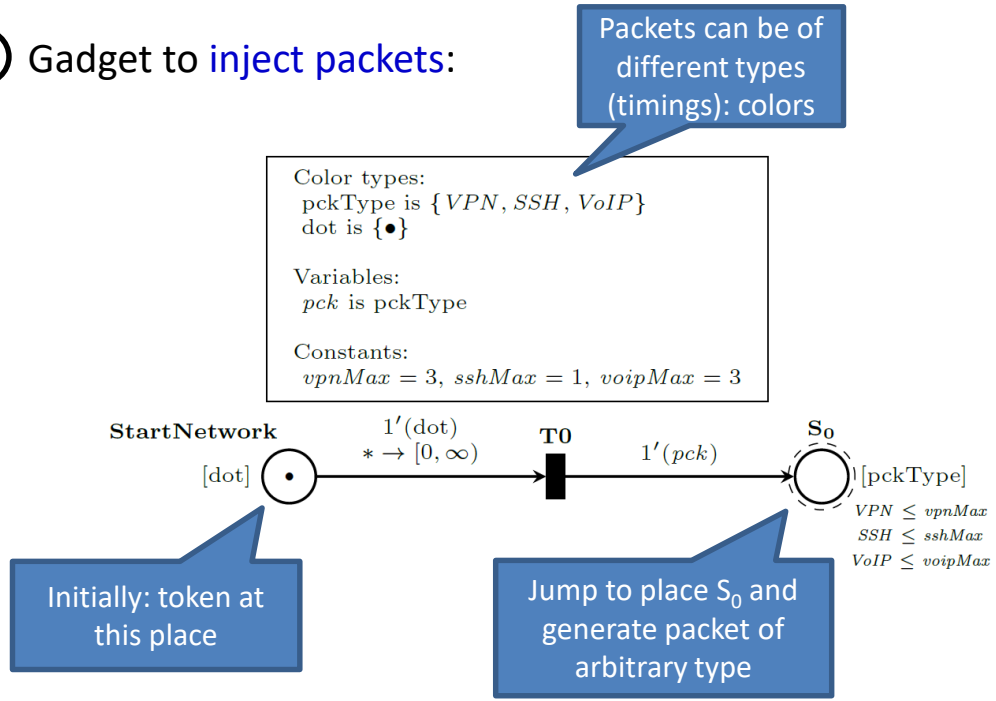
- Much work on the design of efficient algorithms for consistent network updates
- Our goal: automated synthesis of fast updates accounting for temporal properties
  - E.g., different packet types have different requirements and *processing times*
  - Builds upon *NetSynth* (gives fixed update order)
- 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

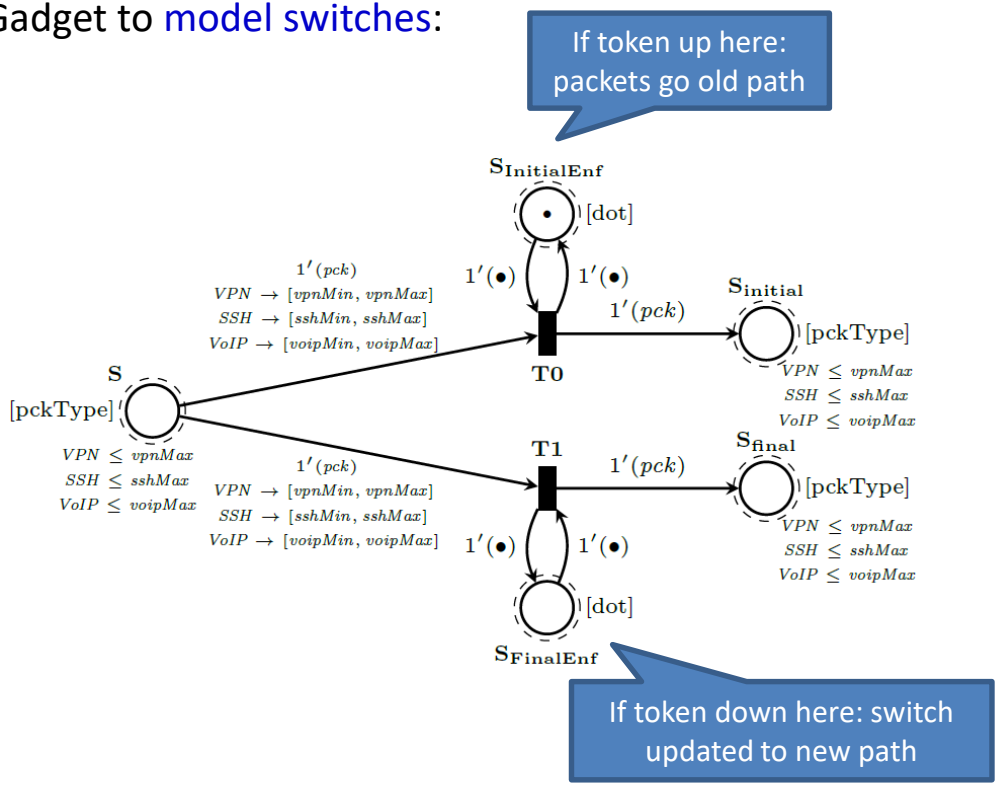
Example: Encoding Network Updates in TACPNs

① Gadget to inject packets:



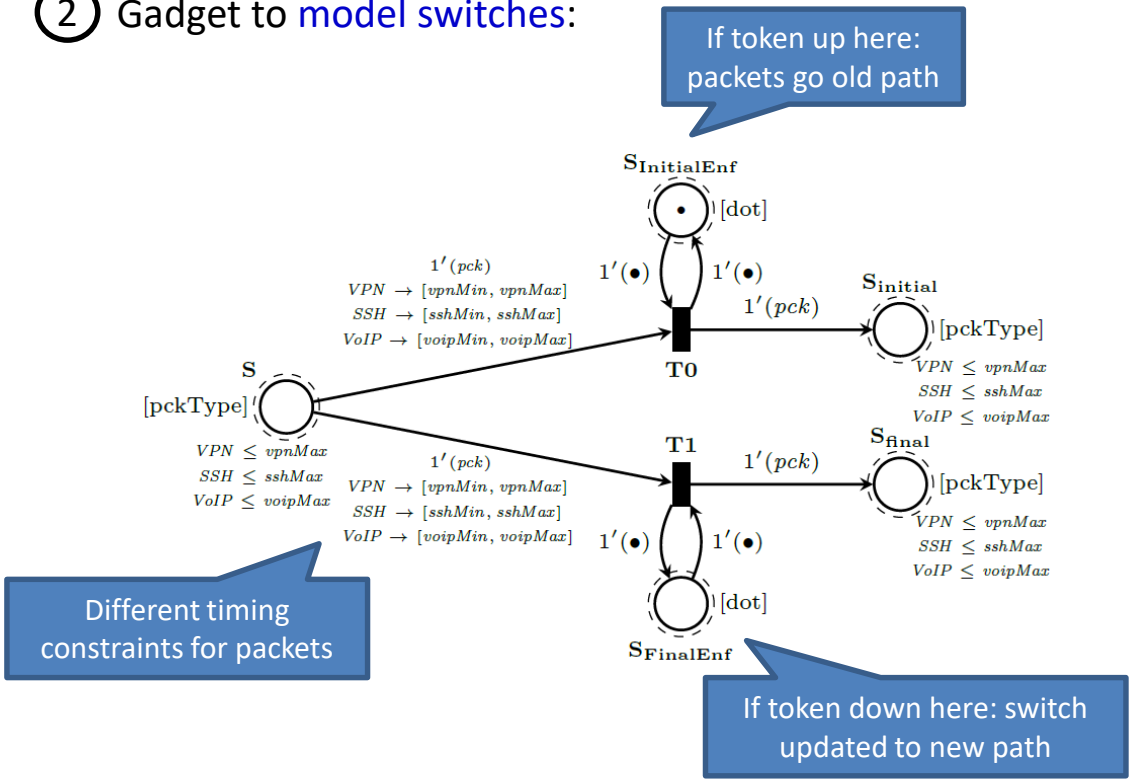
Example: Encoding Network Updates in TACPNs

② Gadget to model switches:



Example: Encoding Network Updates in TACPNs

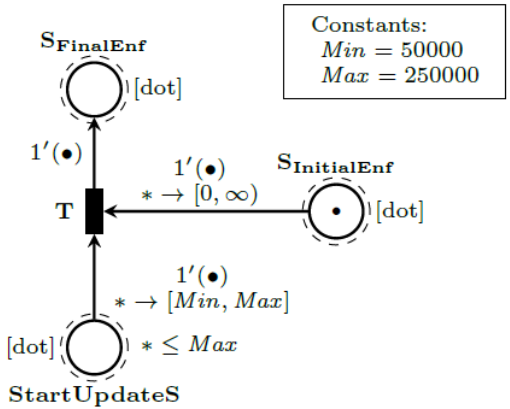
② Gadget to model switches:



Example: Encoding Network Updates in TACPNs

③ Gadget to model switch update:

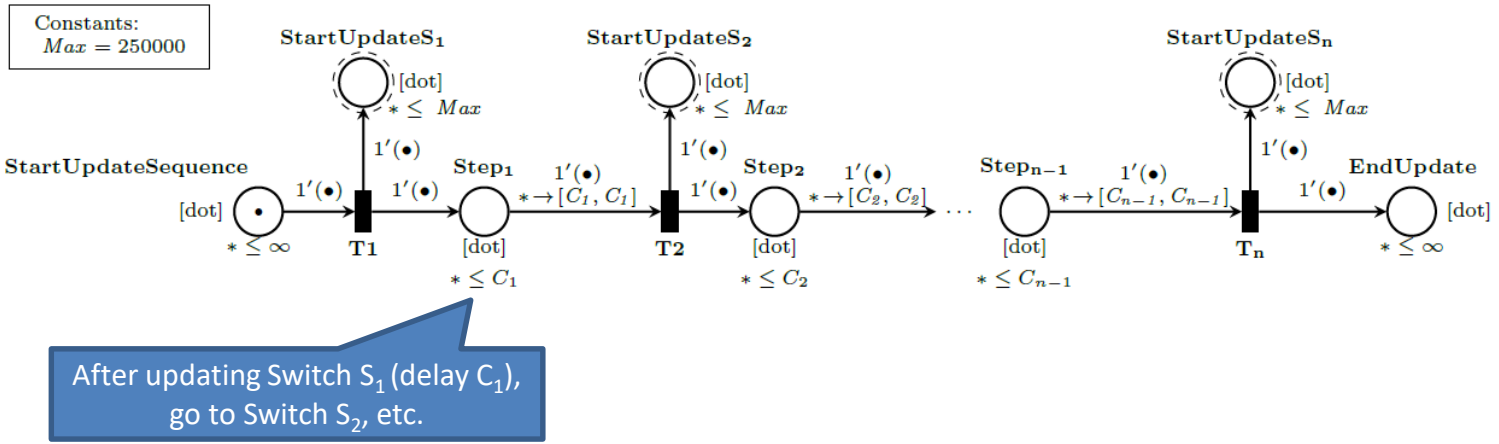
How to change between initial and final switch configuration



Starting here, the update can take time between min and max

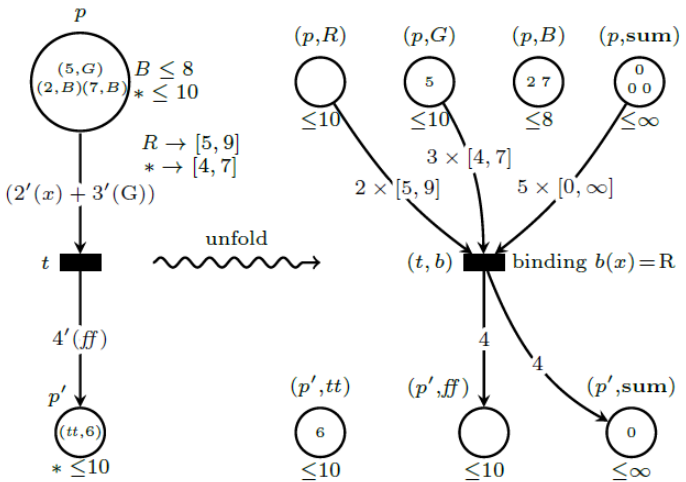
Example: Encoding Network Updates in TACPNs

④ Connecting the pieces: initialization of update sequence for all *n* switches



Analysis

The constructed nets can be analyzed efficiently via their *unfolding* into existing *timed-arc Petri nets*.



Preserves bisimilarity!

Improved Latency of Update Schedules

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

Network	Route length	Verification time[s]	Default update time [s]	Optimized update time [s]	Improvement [%]
TLex	4	0.74	3.58	0.25	92.30%
HiberniaIreland	5	1.02	6.05	0.28	95.50%
Harnet	6	1.42	9.08	0.28	96.97%
UniC	7	1.49	12.65	0.28	97.83%
Oxford	8	2.02	16.78	0.28	98.36%
Xeex	10	5.86	26.68	0.28	98.97%
Sunet	11	10.23	32.45	0.28	99.15%
SwitchL3	12	18.88	38.78	0.28	99.29%
Psinet	14	89.67	53.01	0.28	99.48%
Uunet	15	211.86	61.05	0.28	99.55%
Renater2010	16	480.52	69.58	0.28	99.60%
Missouri	25	timeout	171.05	67.10	60.77%
Syringa	35	timeout	336.05	295.35	12.11%
VtlWavenet2011	35	timeout	336.06	295.35	12.11%

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

Improved Latency of Update Schedules

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

Network	Route length	Verification time[s]	Default update time [s]	Optimized update time [s]	Improvement [%]
TLex	4	0.74	3.58	0.25	92.30%
HiberniaIreland	5	1.02	6.05	0.28	95.50%
Harnet	6	1.42	9.08	0.28	96.97%
UniC	7	1.49	12.65	0.28	97.83%
Oxford	8	2.02	16.78	0.28	98.36%
Xeex	10	5.86	26.68	0.28	98.97%
Sunet	11	10.23	32.45	0.28	99.15%
		18.88	38.78	0.28	99.29%
		89.67	53.01	0.28	99.48%
		211.86	61.05	0.28	99.55%
		480.52	69.58	0.28	99.60%
Missouri	25	timeout	171.05	67.10	60.77%
Syringa	35	timeout	336.05	295.35	12.11%
VtlWavenet2011	35	timeout			12.11%

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

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

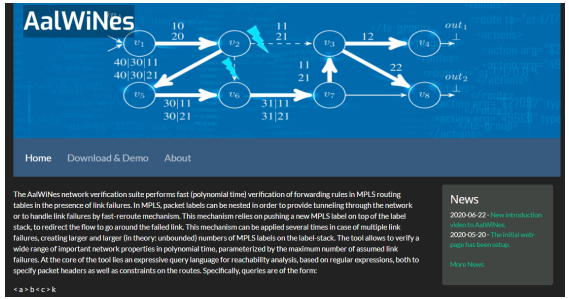
- Network topologies from the To
- Experiments run on a 64-bit Ubuntu 18.04 laptop

Conclusion

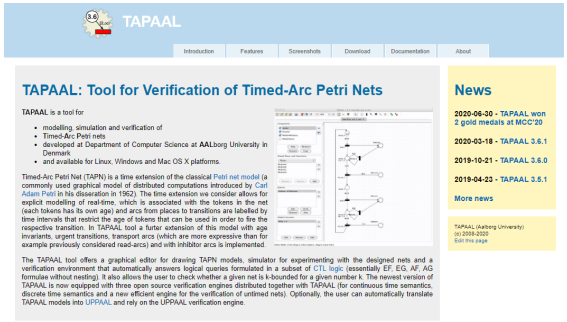
- Finally: networks are moving from manual to more automated operations
- Supported by emerging programmable networks and their solid theoretical foundations and languages
- Automata-theoretical approaches can be used to perform fast what-if analysis of the policy compliance (e.g., P-Rex, *AalWiNes*, 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/



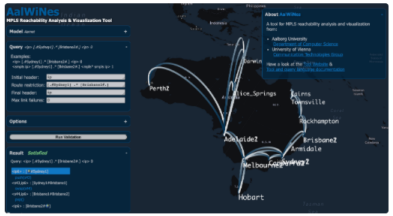
TAPAAL.net



Netverify.fun

RESEARCH, NETWORK, VERIFICATION

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



Stefan Schmid View
Jul 20, 2020 · 6 mins read

Jiri Srba View
Jul 20, 2020 · 6 mins read

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 Schnepf and Jiri Srba.
27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**), Virtual Conference, March 2021.

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.
38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (**PERFORMANCE**) and ACM Performance Evaluation Review (**PER**), Milan, Italy, November 2020.

P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures
Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen.
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?