Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks

… and Segment Routing!

…

Stefan Schmid
University of Vienna, Austria

Jiri Srba
Aalborg University, Denmark

Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks

Stefan Schmid
University of Vienna, Austria

Jiri Srba
Aalborg University, Denmark

Teaser: Can we verify reachability under k failures without trying exponentially many options?

Yes. *MUCH FASTER!* An Automata-Theoretic Approach.

Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks

Stefan Schmid
University of Vienna, Austria

Jiri Srba
Aalborg University, Denmark

Kudos to collaborators:
Jesper Stenbjerg Jensen, Jonas Sand Madsen, Troels Beck Krøgh
at Aalborg University, Denmark

Configuring Networks is Hard…

Datacenter, enterprise, carrier networks: **mission-critical infrastructures**.
But even **techsavvy** companies struggle to provide reliable operations.

github SOCIAL CODING

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm

amazon web services

Go Daddy.COM

Service outage was due to a series of internal network events that corrupted router data tables

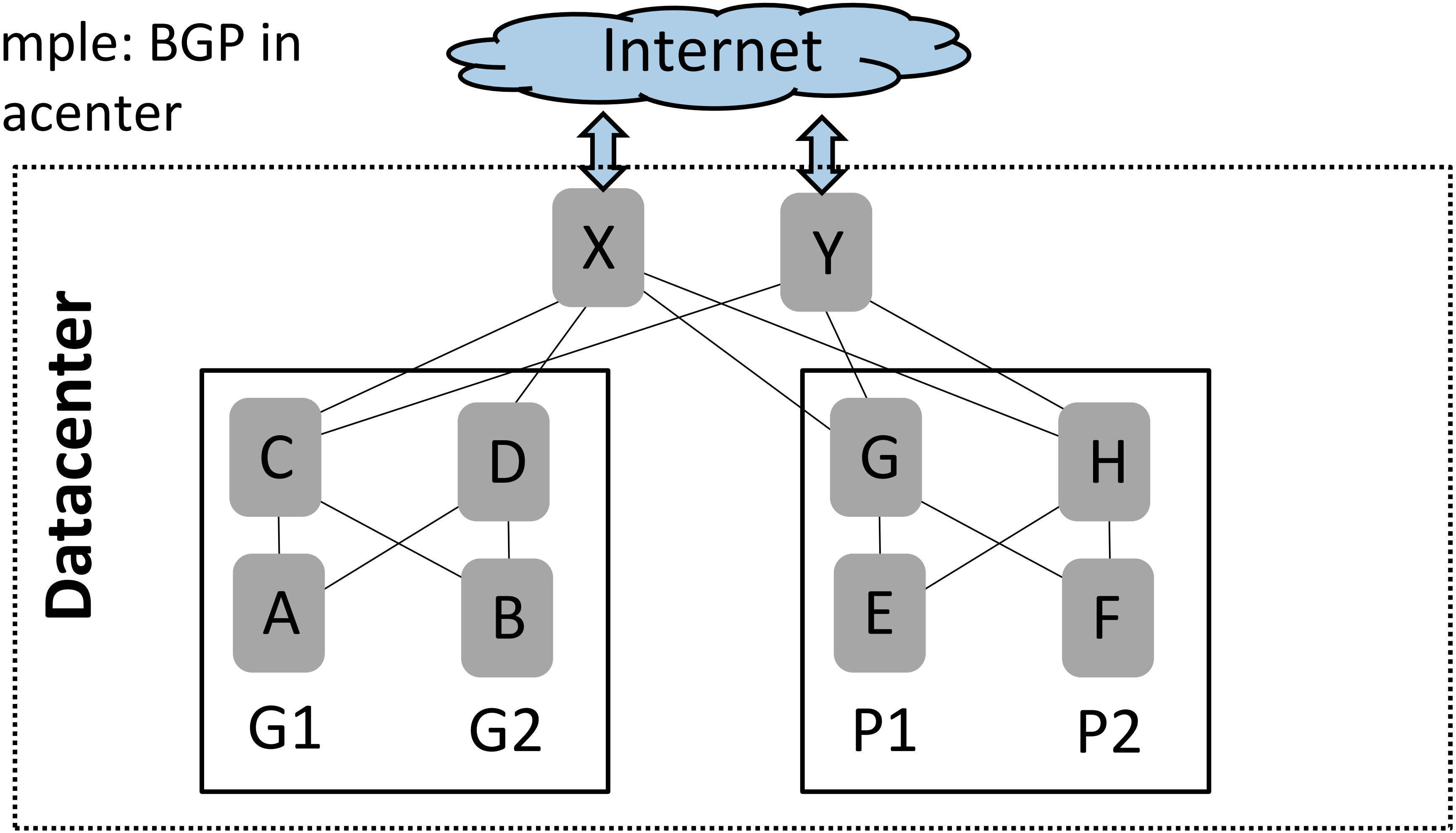
Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems

United Airlines

Credits: Nate Foster

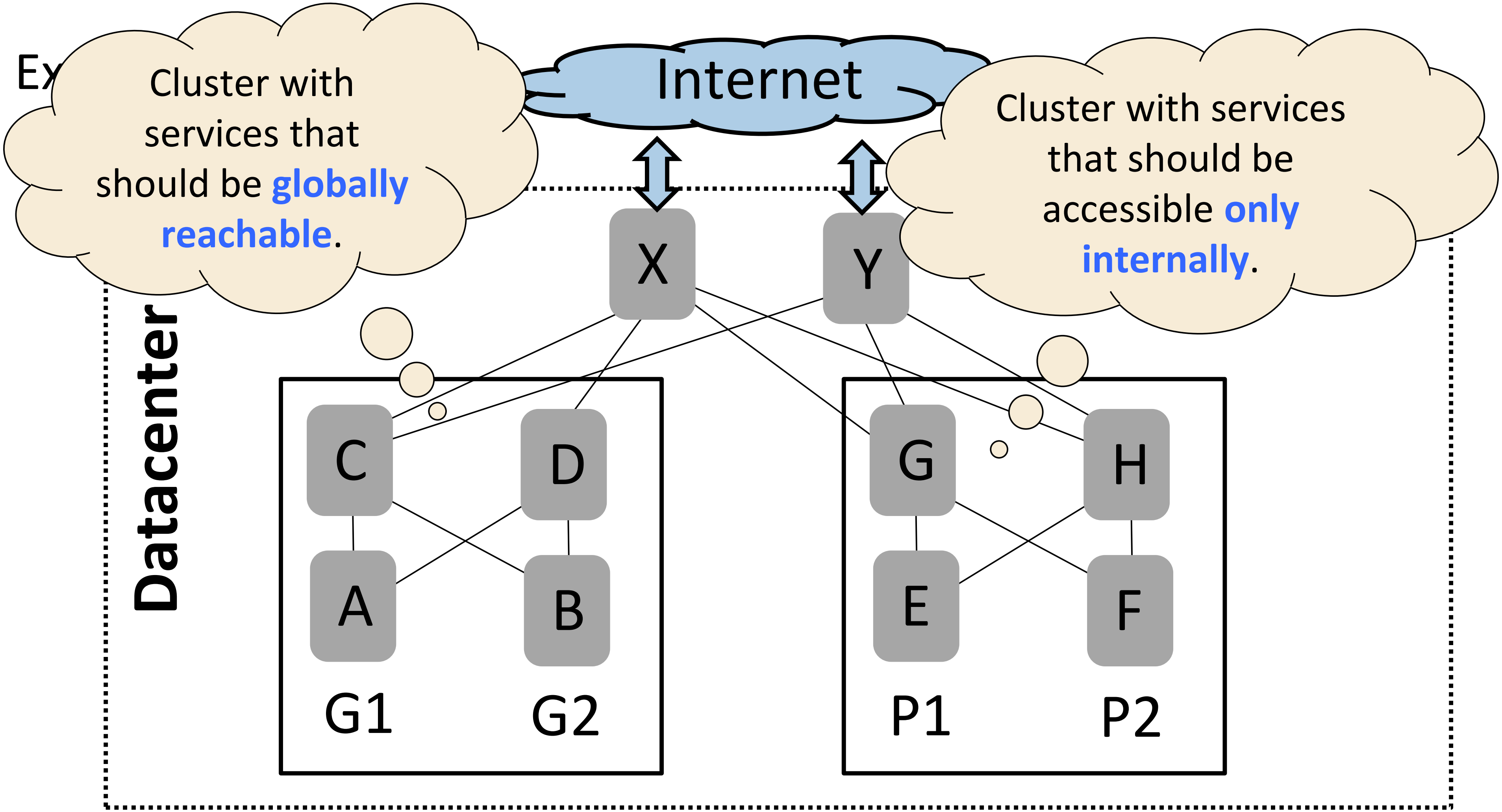
… Especially Under Failures

Example: BGP in Datacenter



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

… Especially Under Failures

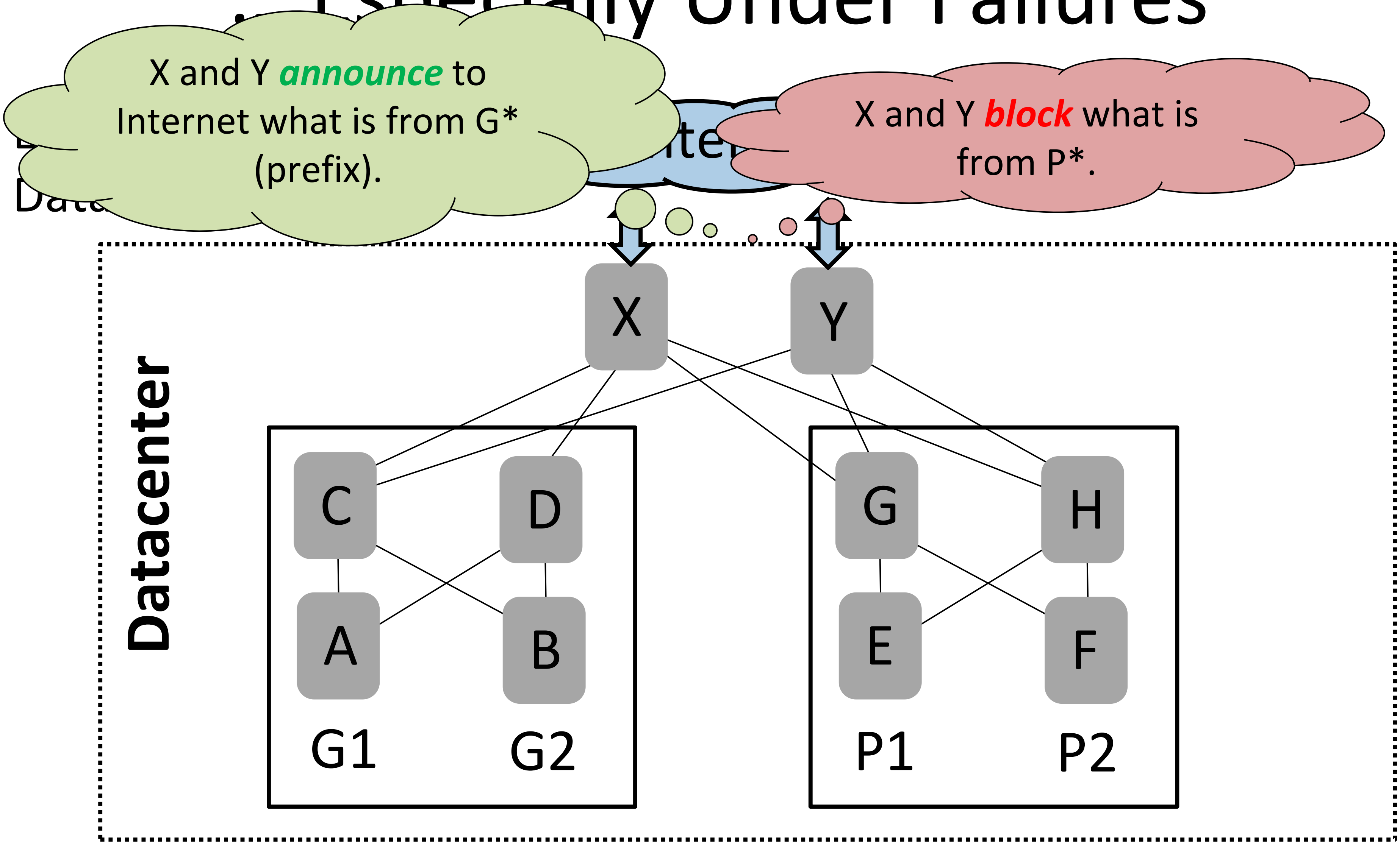


Cluster with services that should be **globally reachable**.

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

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

… Especially Under Failures

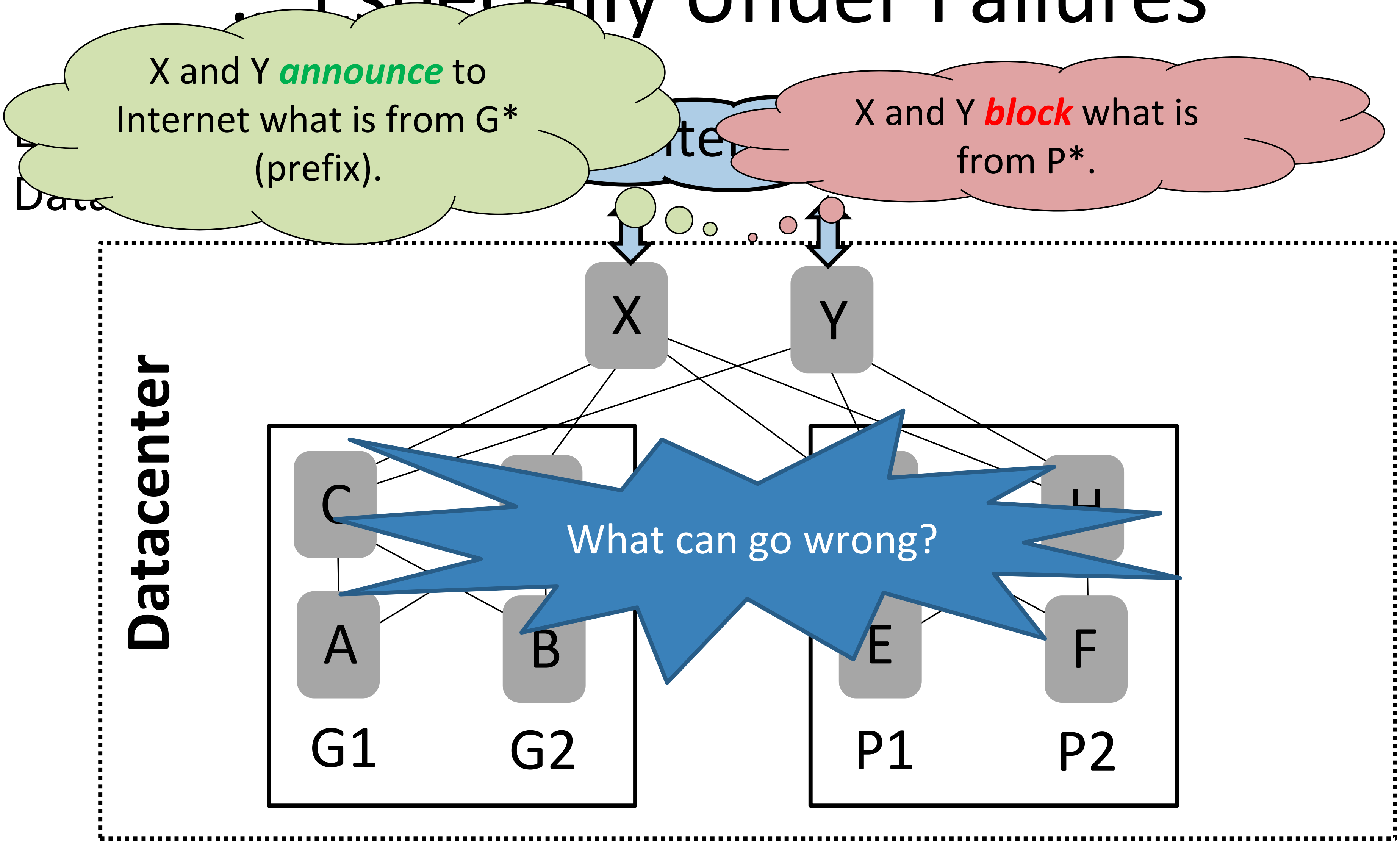


X and Y ***announce*** to Internet what is from G* (prefix).

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

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

… Especially Under Failures



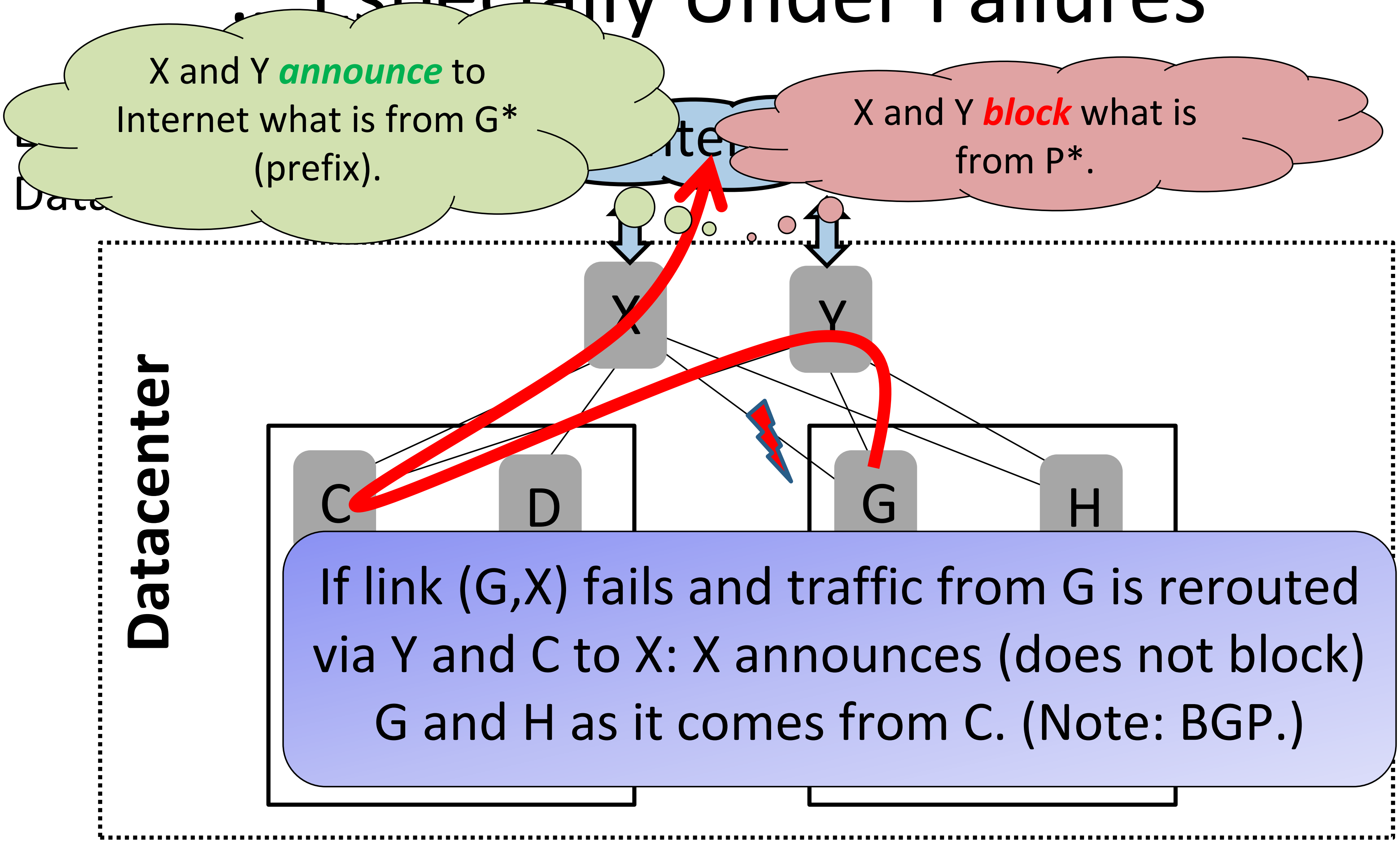
X and Y ***announce*** to Internet what is from G* (prefix).

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.

… Especially Under Failures



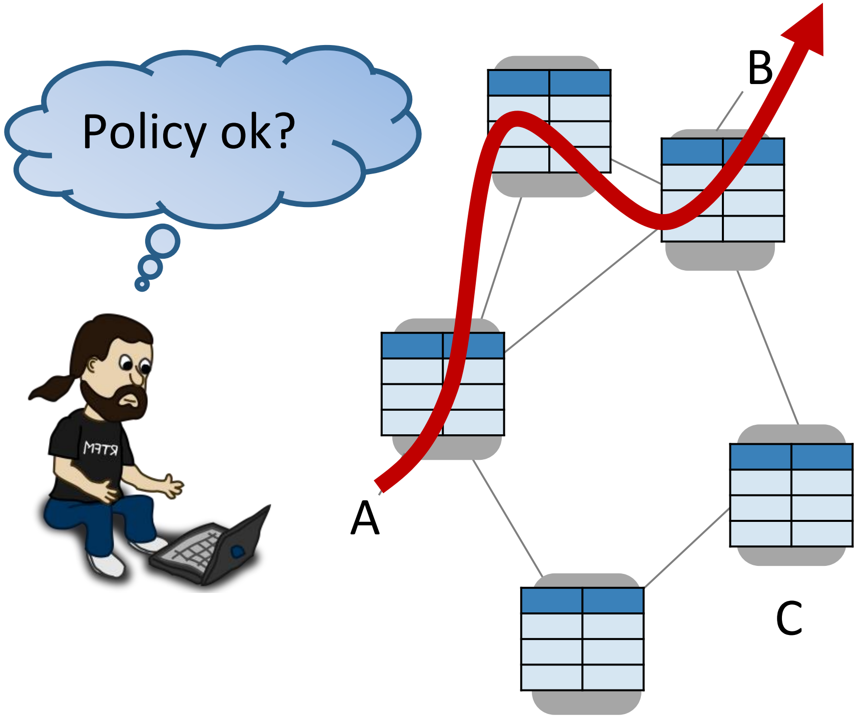
X and Y ***announce*** to Internet what is from G* (prefix).

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.

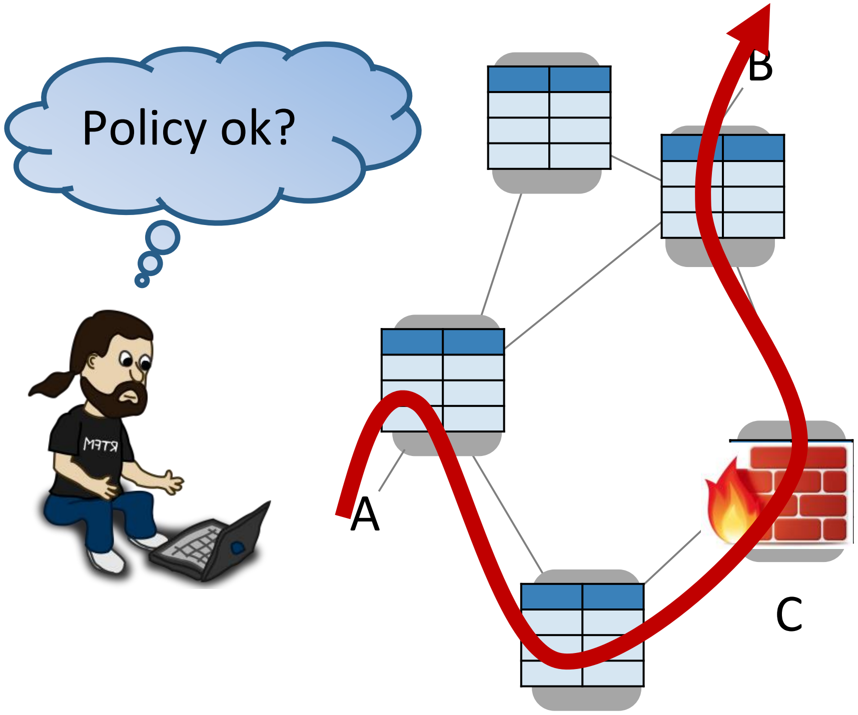
Network Administration Today



- Many **forwarding tables** with many **rules**, **distributed** across network

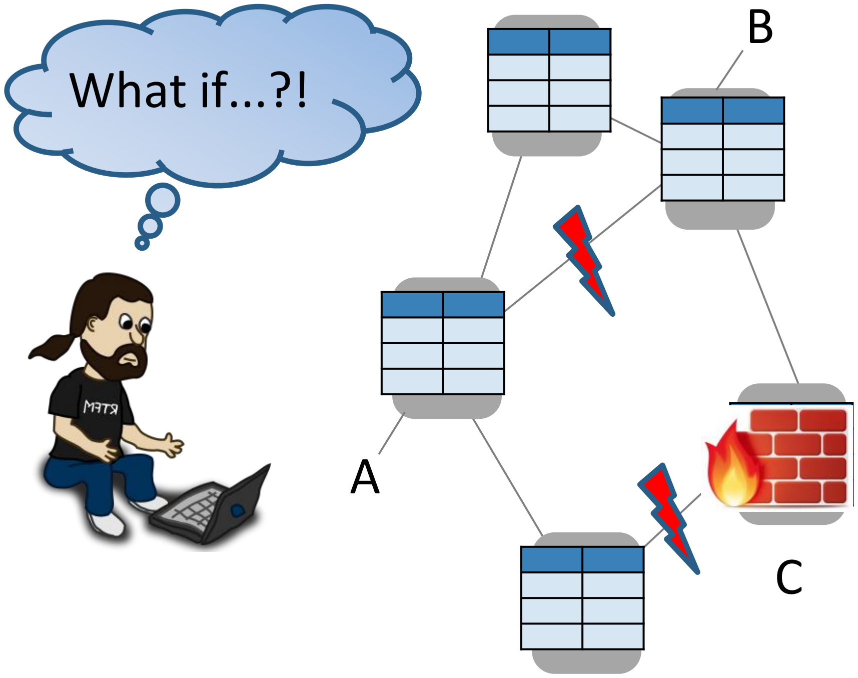
- **Sysadmin** responsible for:
  - **Reachability:** Can traffic from ingress port A reach egress port B?
  - **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
  - **Non-reachability:** Is it ensured that traffic originating from A never reaches B?
  - **Waypoint ensurance:**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

Network Administration Today



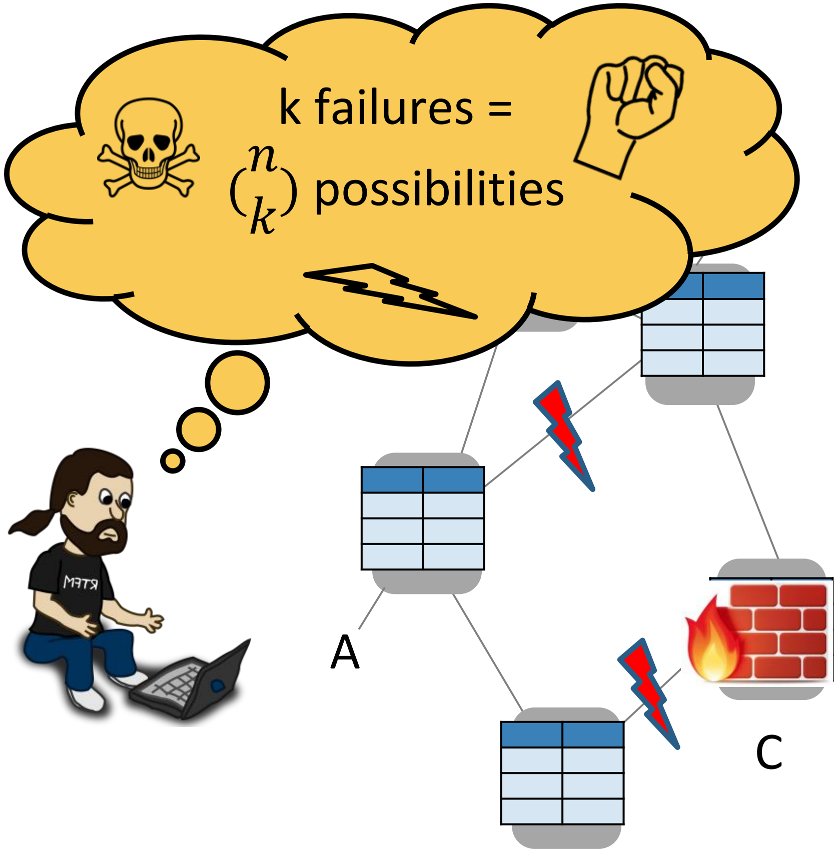
- Many **forwarding tables** with many **rules**, **distributed** across network
- Sysadmin responsible for:
  - **Reachability:** Can traffic from ingress port A reach egress port B?
  - **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
  - **Non-reachability:** Is it ensured 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)?

Network Administration Today



- Many **forwarding tables** with many **rules**, **distributed** across network
- **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?
  - **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)?
  - … **even under (multiple) failures!**

Network Administration Today



- Many **forwarding tables** with many **rules**, **distributed** across network

- **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?
  - **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)?
  - … **even under (multiple) failures!**

The Good News

- Networks are becoming more **programmable** and logically **centralized**, have **open** interfaces, …
- … are based on **formal foundations**…
- … researchers develop high-level specification languages such as **NetKAT**.

Enables a more **automated** network operation and verification!

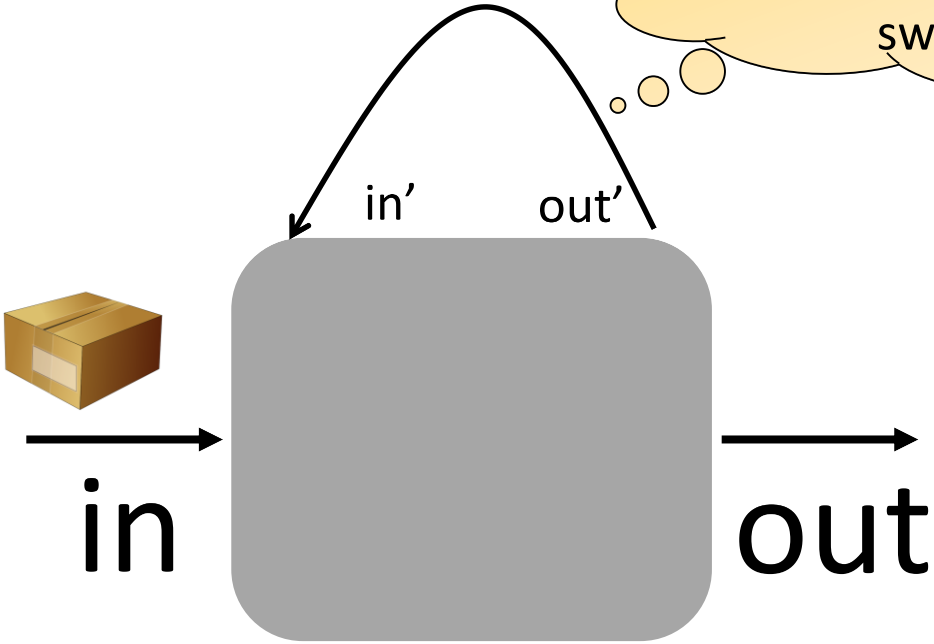
The Bad News

- For many **traditional networks** (still *predominant*!), such benefits are not available yet
- Many existing tools cannot deal with **failures**
- **Super-polynomial** runtime, verification PSPACE-hard
- Other limitations: e.g., fixed header size

Tractability of Verification

Reachability is **undecidable** in SDN:

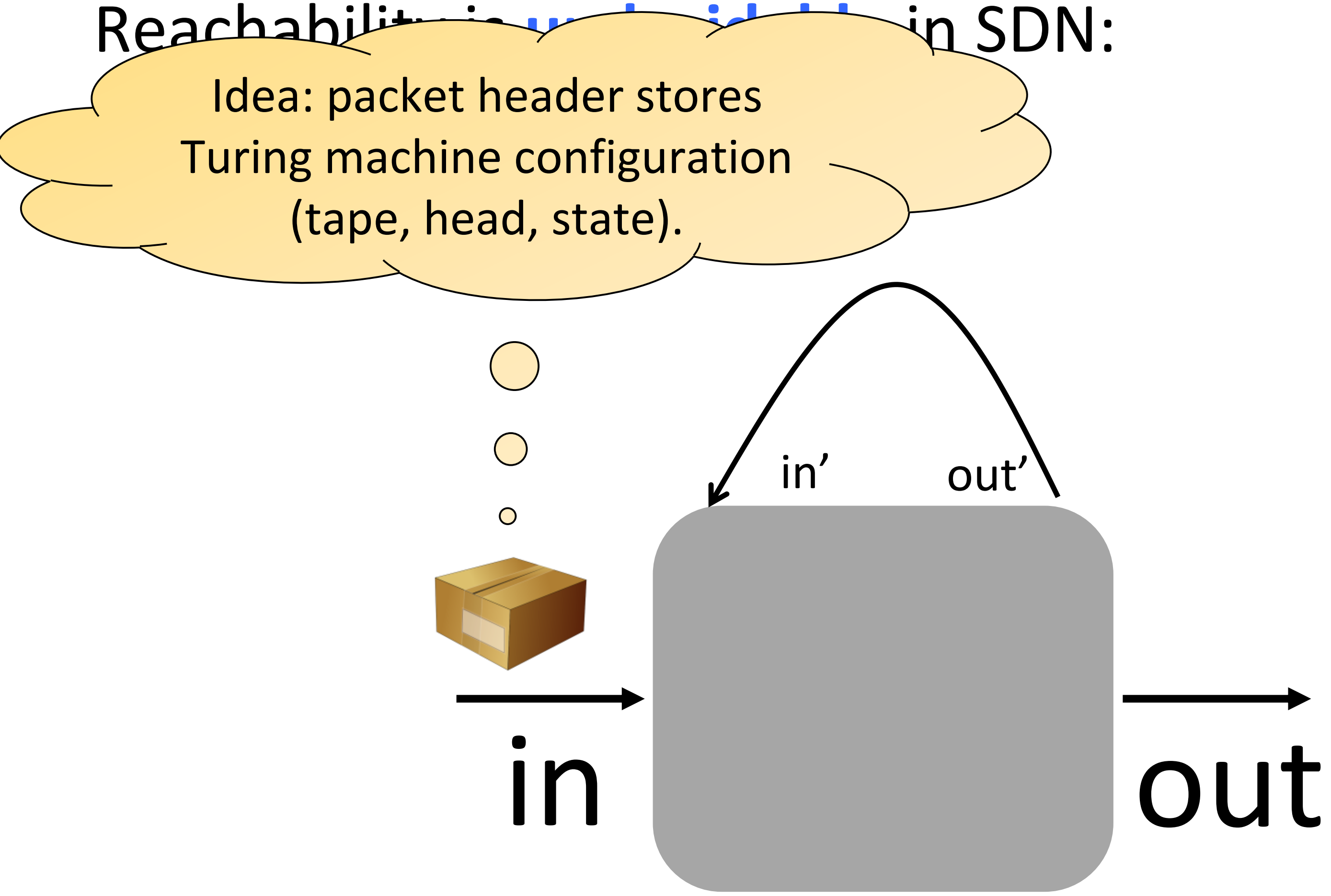
Can emulate a Turing machine.



Self-loop: could be replaced by "dummy switch".

Tractability of Verification

Reachability is undecidable in SDN:

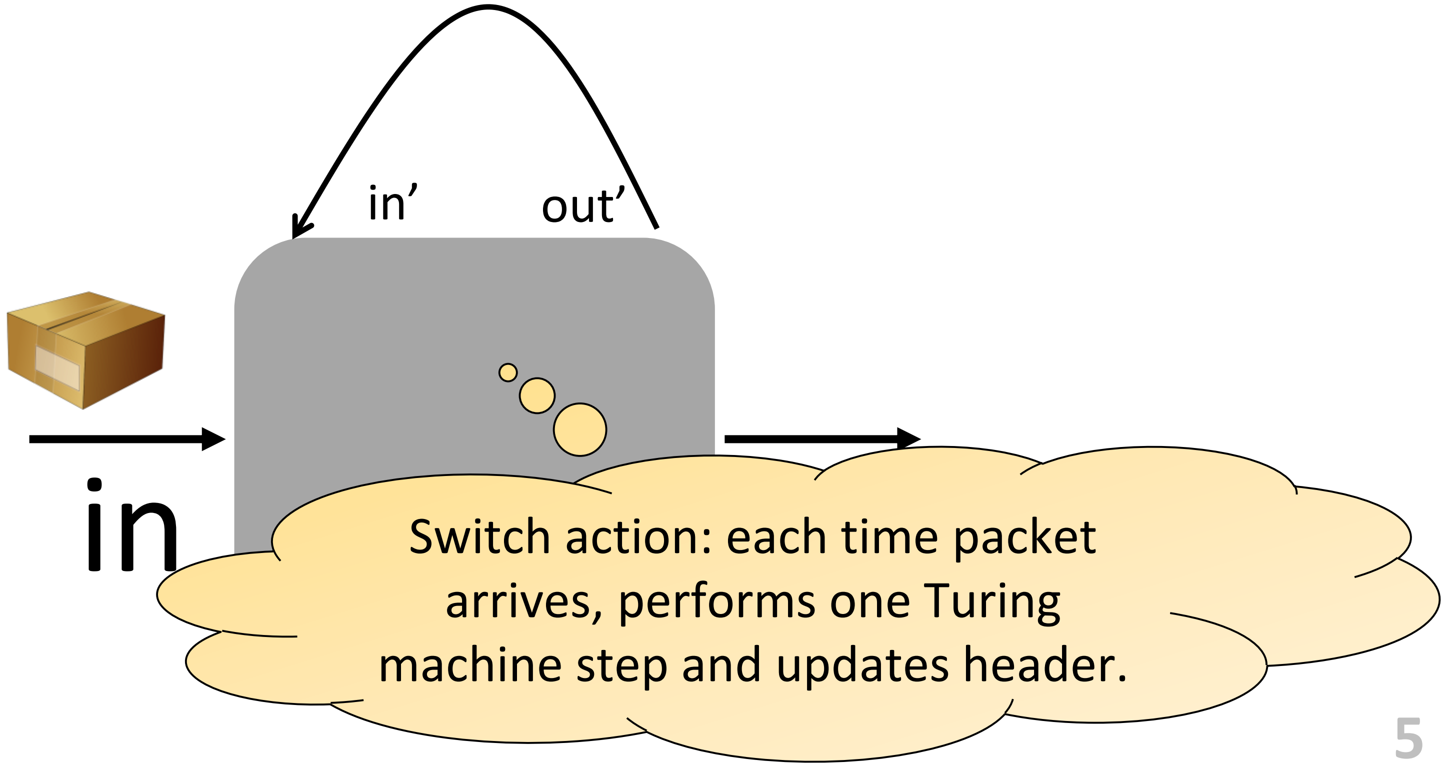


Idea: packet header stores Turing machine configuration (tape, head, state).

Tractability of Verification

Reachability is **undecidable** in SDN:

Can emulate a Turing machine.

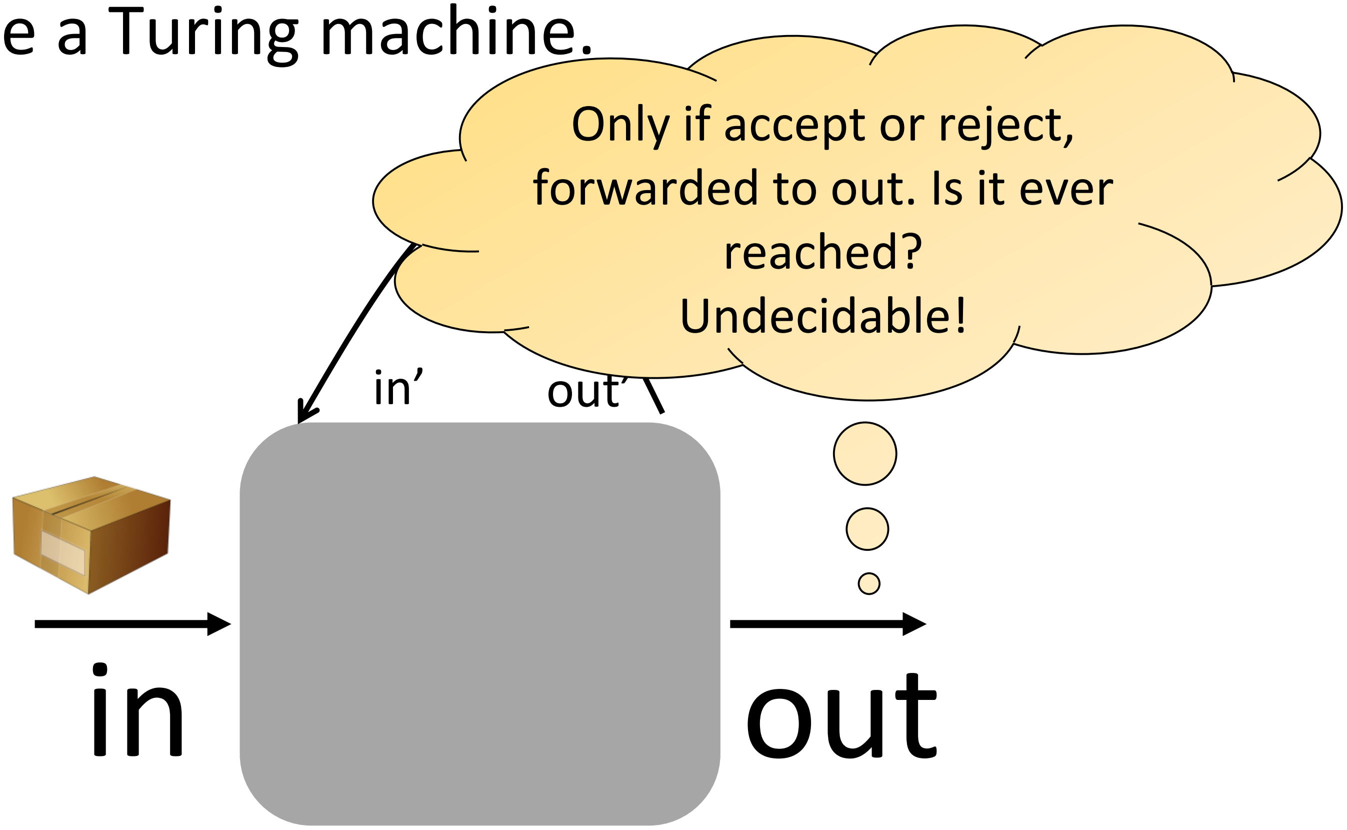


Switch action: each time packet arrives, performs one Turing machine step and updates header.

Tractability of Verification

Reachability is **undecidable** in SDN:

Can emulate a Turing machine.



Only if accept or reject, forwarded to out. Is it ever reached?
Undecidable!
in'
out'
in
out

Our Contribution

Polynomial-Time What-if Analysis
for Prefix Rewriting Networks

Our Contribution

Independently of the number of failures! No need to try combinations.

Reachability, loop-freedom, waypointing, etc.!

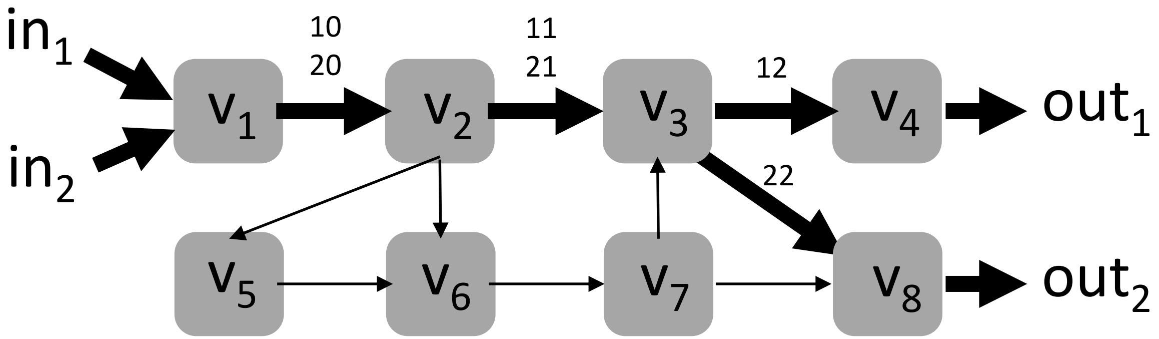
Polynomial-Time What-if Analysis for Prefix Rewriting Networks

e.g., MPLS networks or Segment Routing networks

Support arbitrary header sizes!

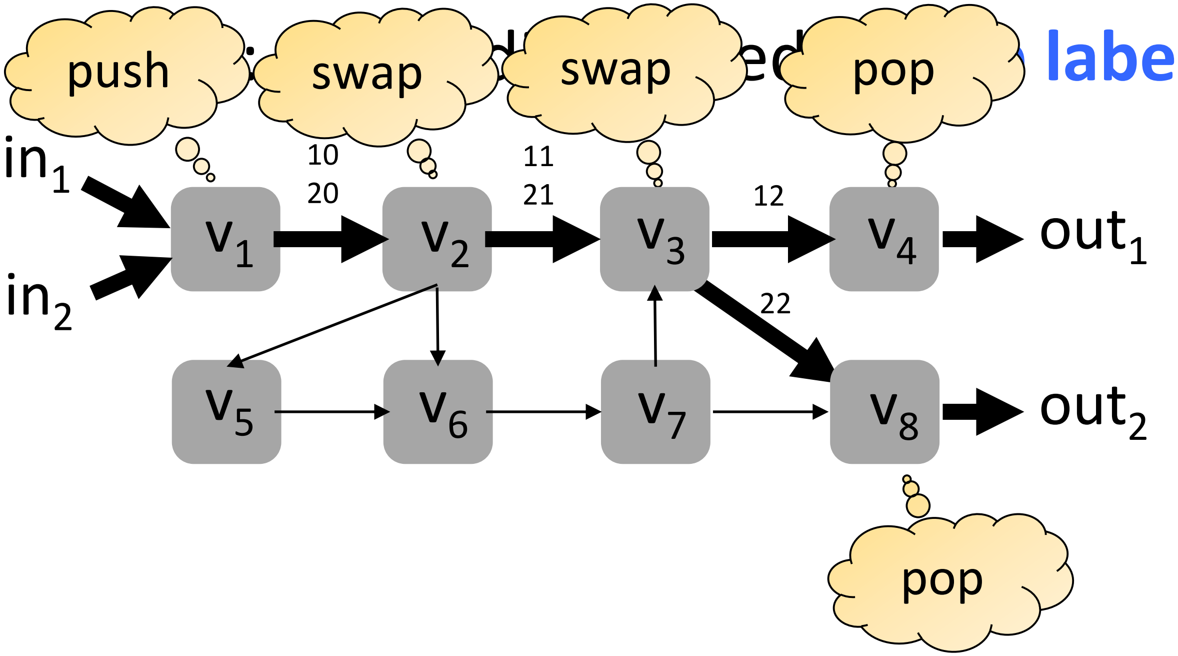
MPLS Networks

- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

MPLS Networks

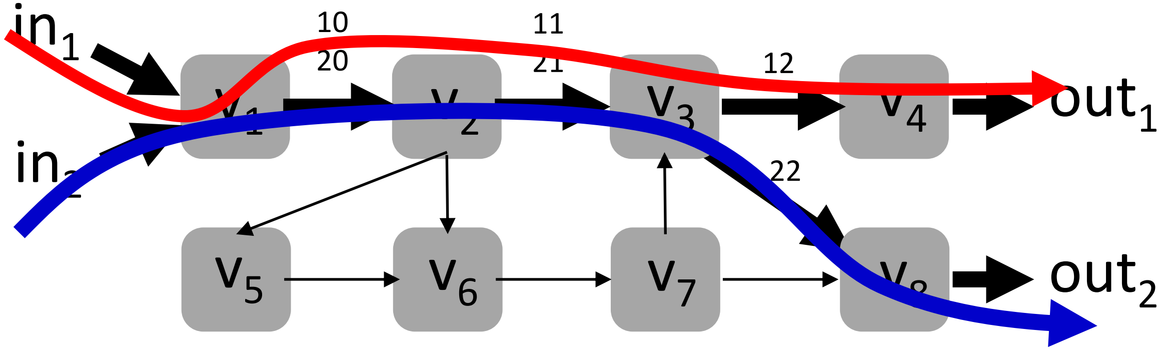


label of label **stack**

Default routing of two flows

MPLS Networks

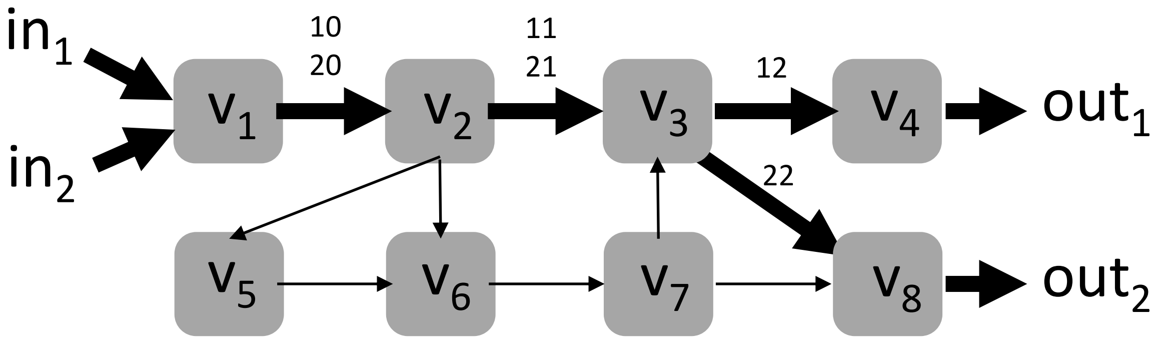
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

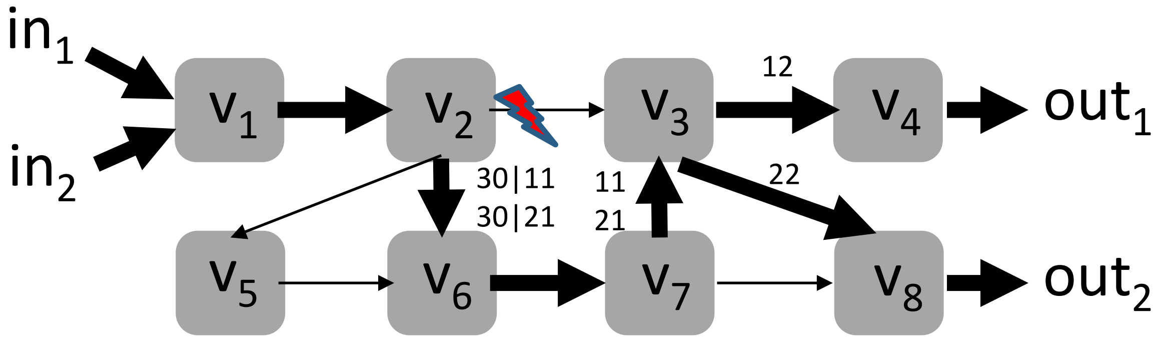
MPLS Networks: 1 Failure

- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

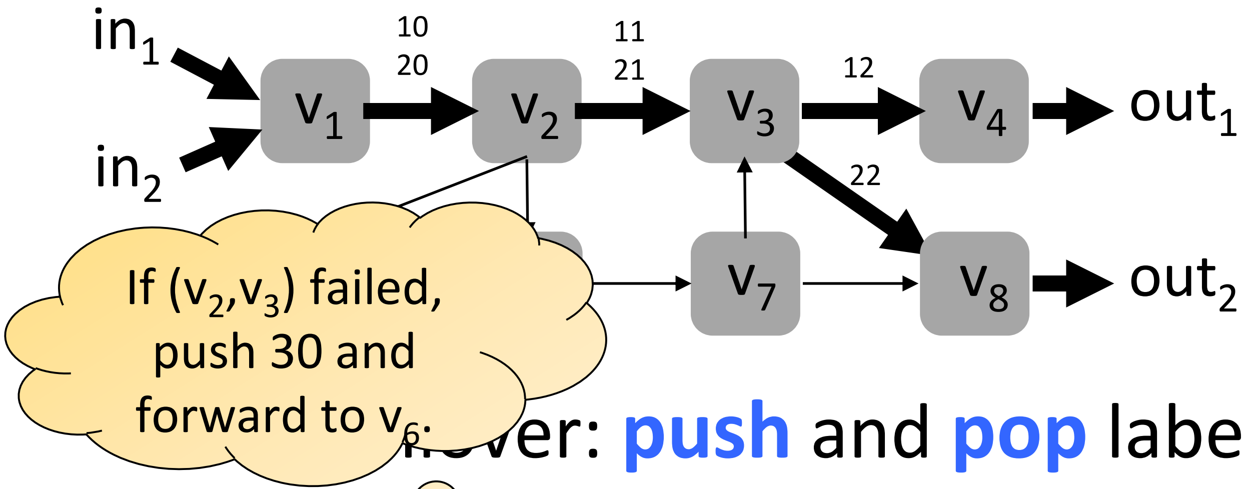
- For failover: **push** and **pop** label



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

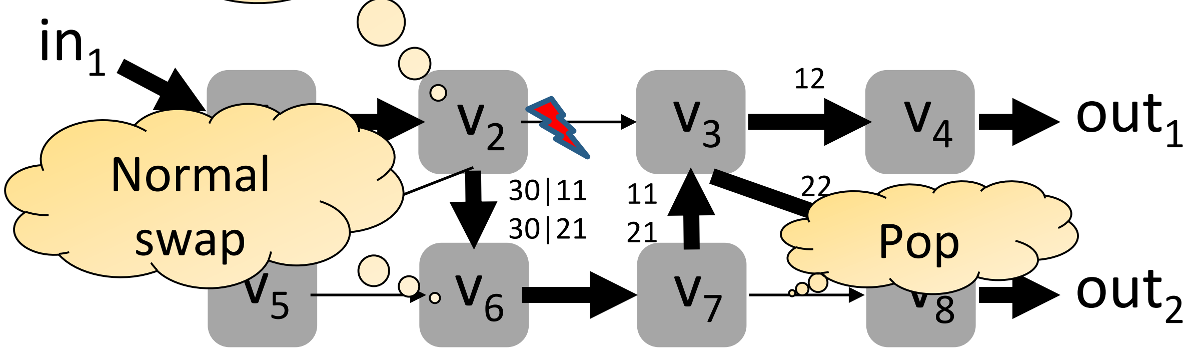
MPLS Networks: 1 Failure

- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

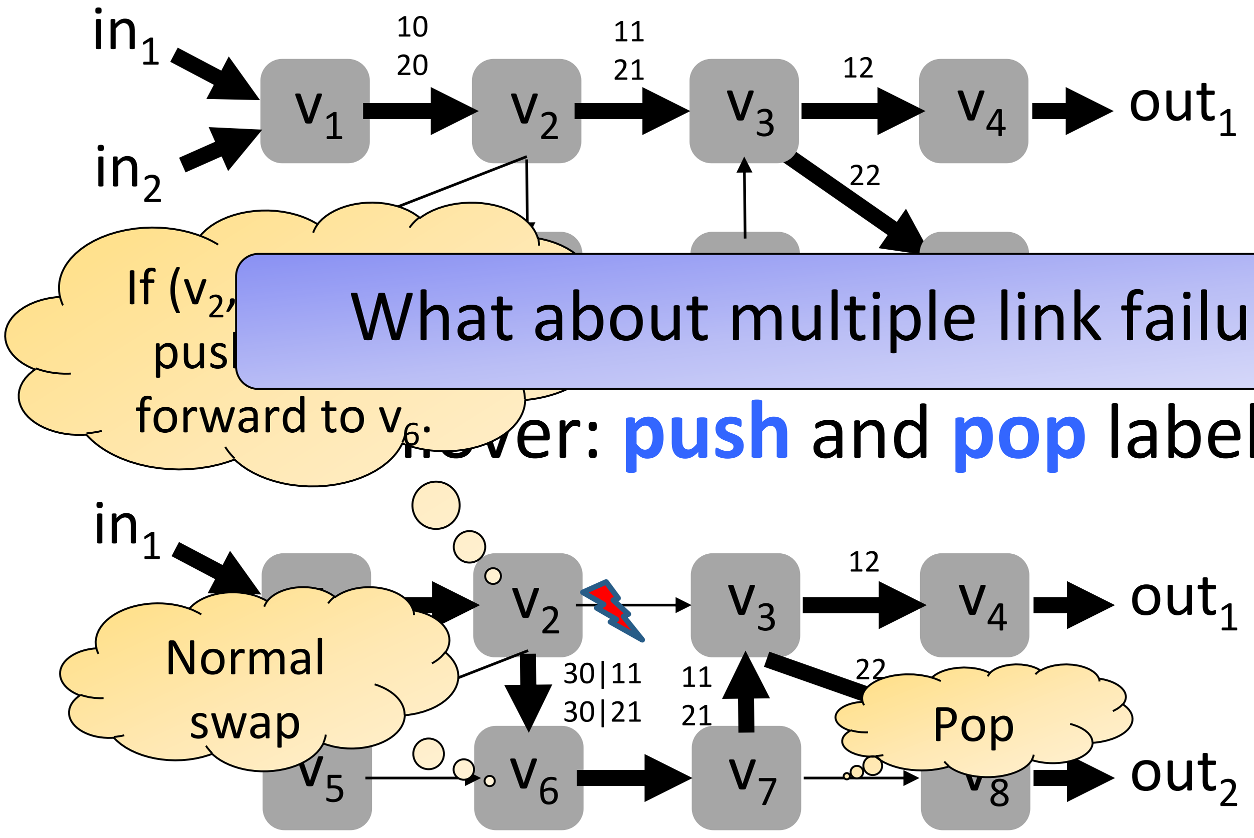
- over: **push** and **pop** label



One failure: push 30: route around (v$_2$,v$_3$)

MPLS Networks: 1 Failure

- MPLS: forwarding based on **top label** of label **stack**



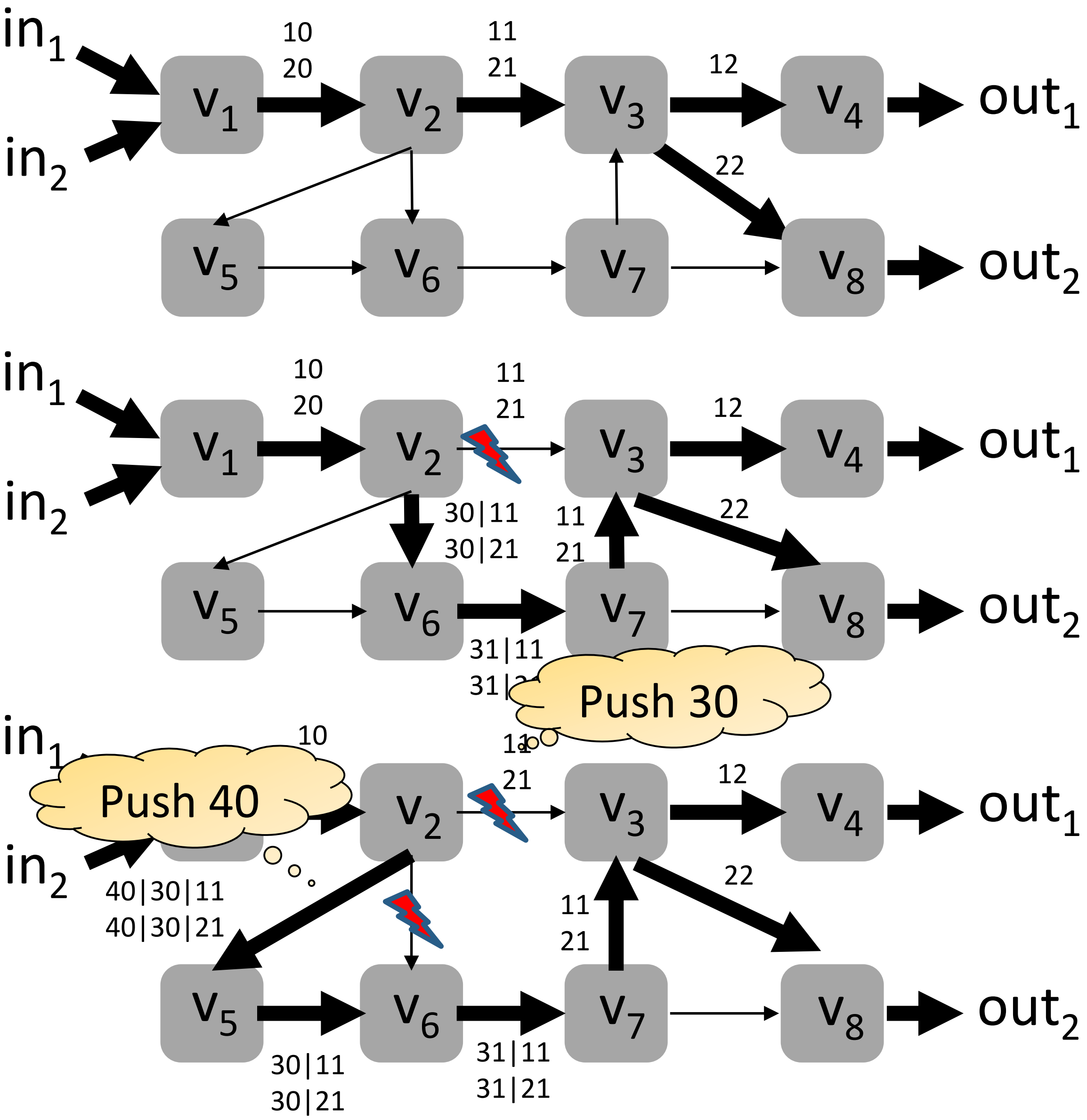
Default routing of two flows

What about multiple link failures?

ver: **push** and **pop** label

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

MPLS Networks: 2 Failures



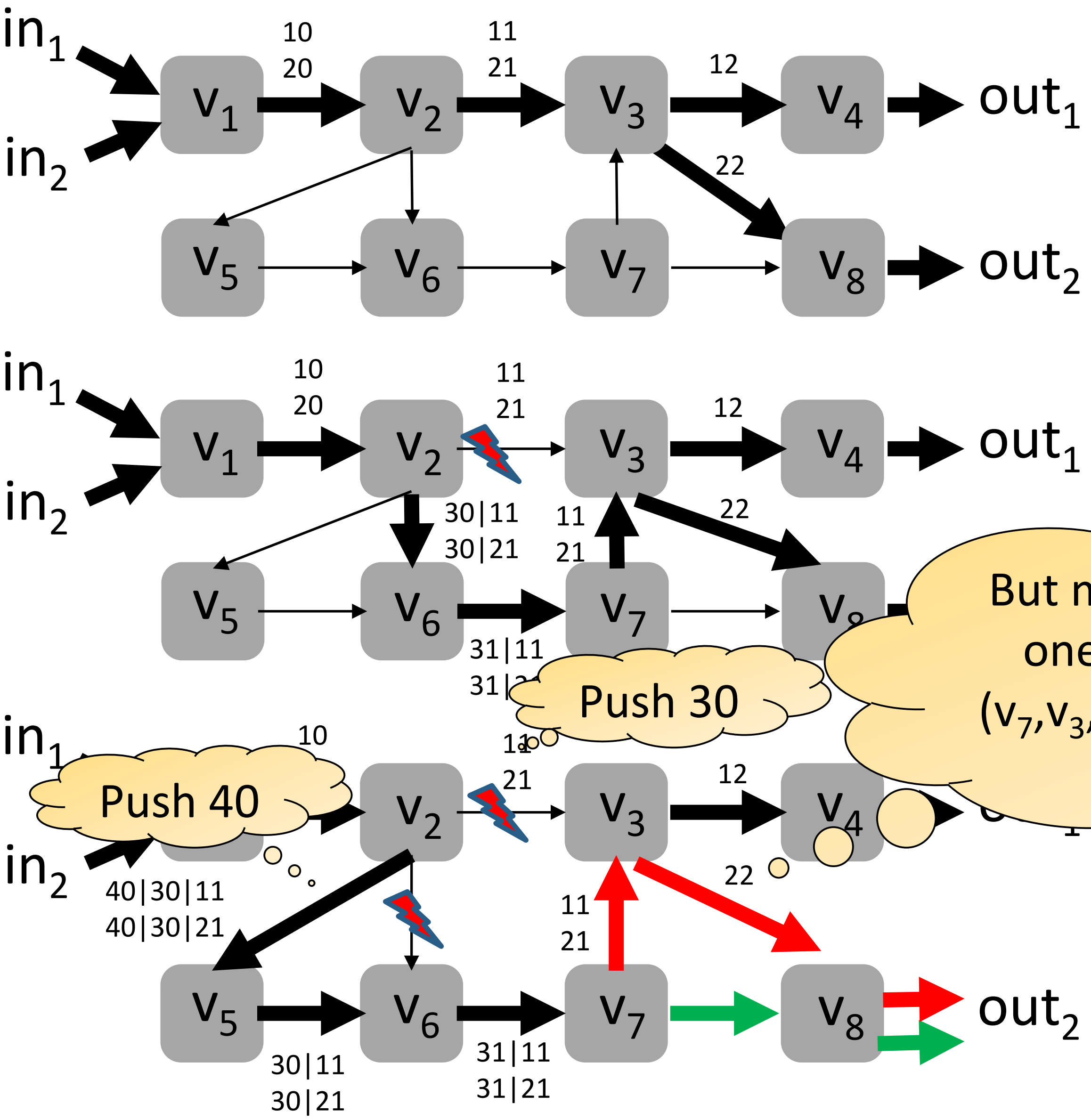
Original Routing

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)

MPLS Networks: 2 Failures



Original Routing

One failure: push 30:

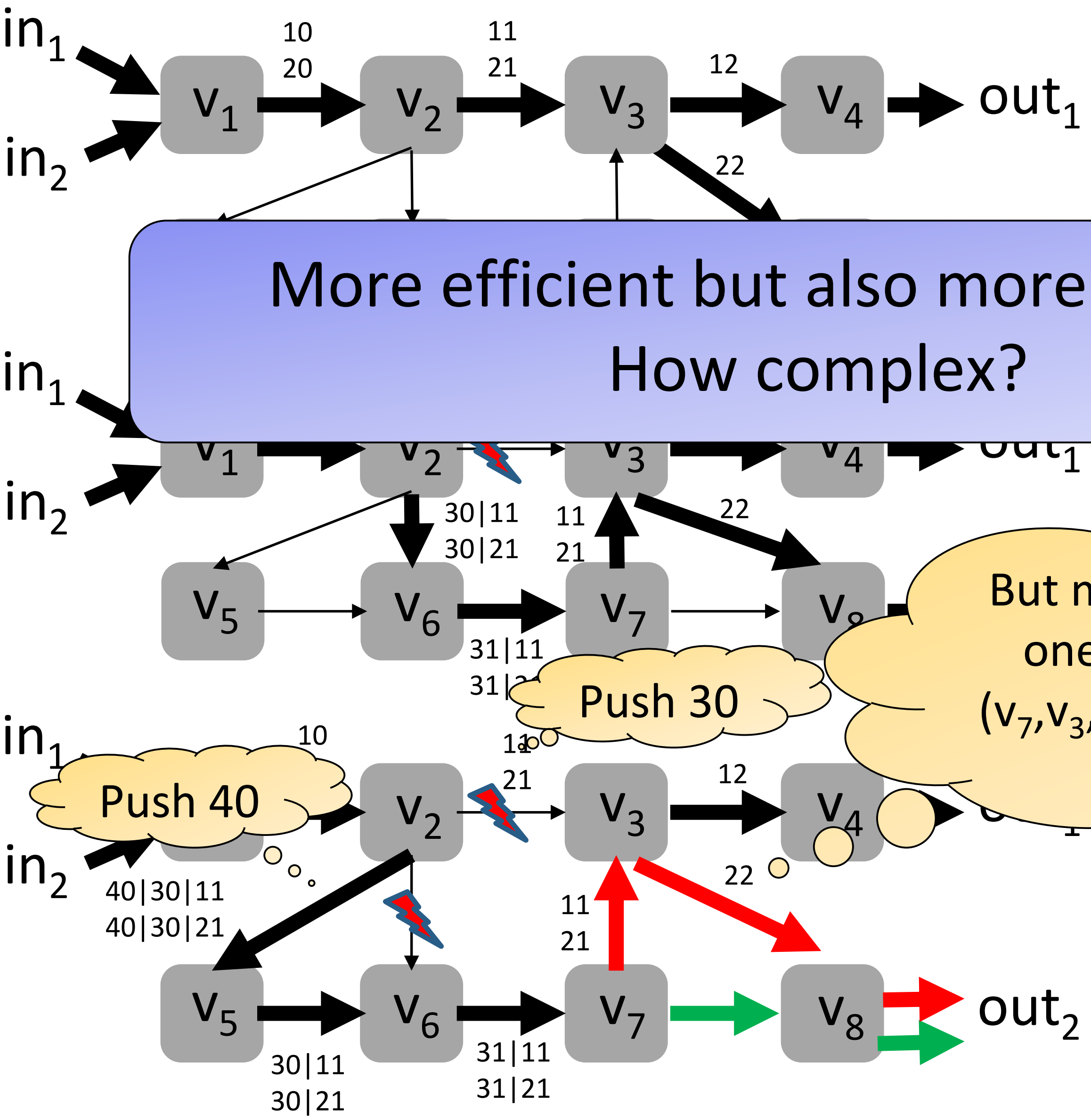
)

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

30: route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

MPLS Networks: 2 Failures



Original Routing

More efficient but also more complex!

How complex?

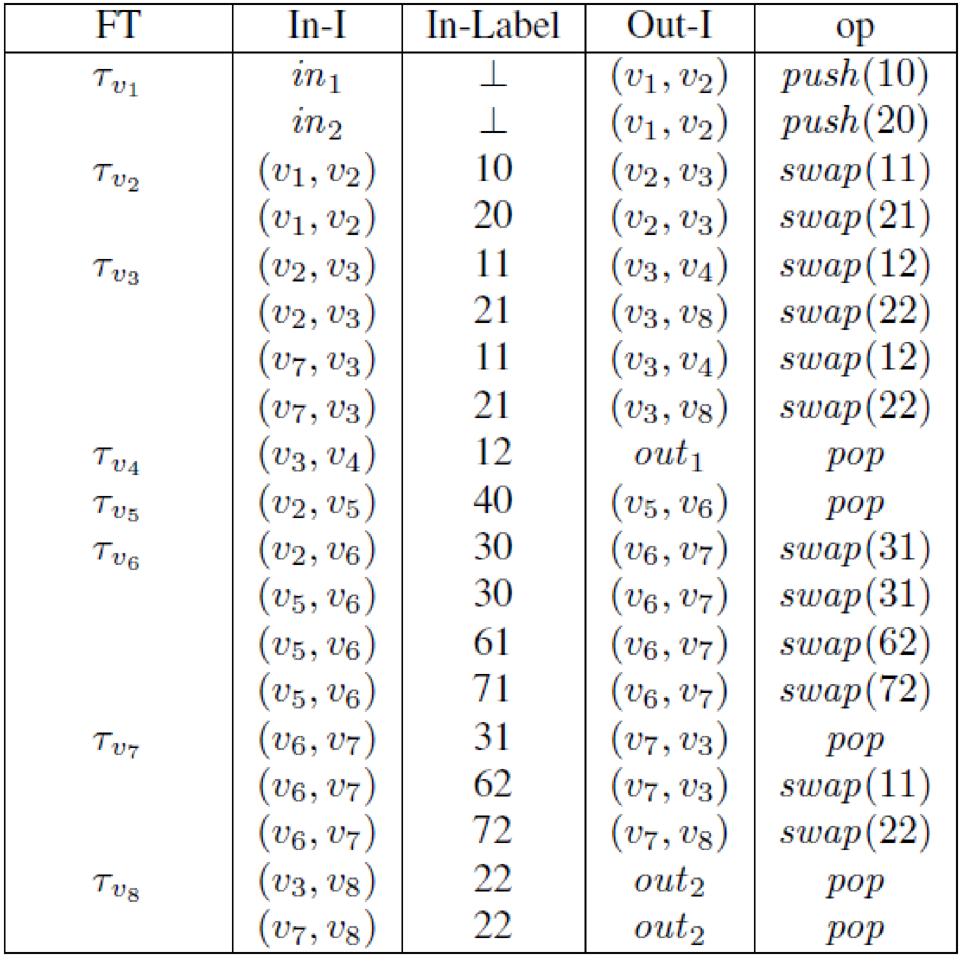
One failure: push 30:

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

n 30: route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

Forwarding Tables for Our Example



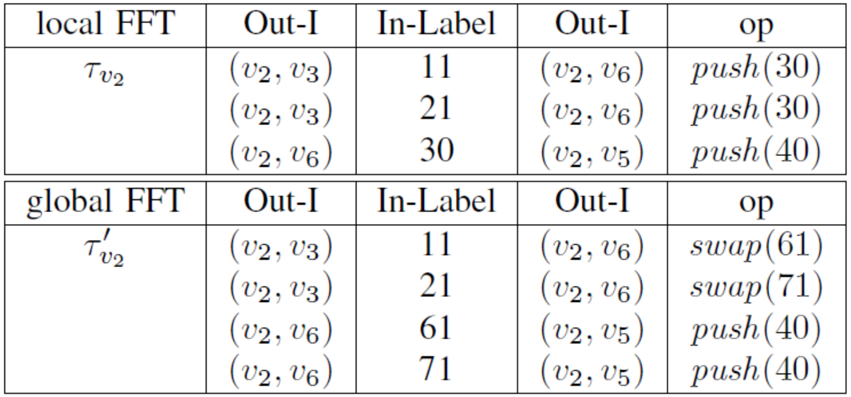
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(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

Flow Table

Protected link

Alternative link

Label

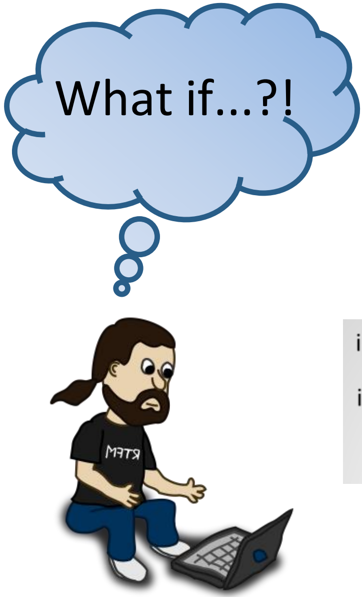


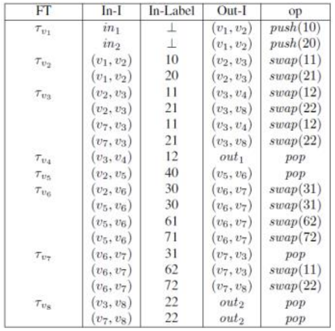
local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$

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)	$push(40)$

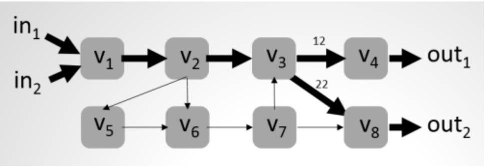
Failover Tables

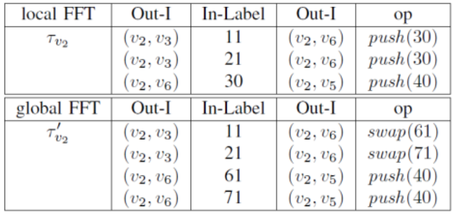
Polynomial-Time Verification: An Automata-Theoretic Approach



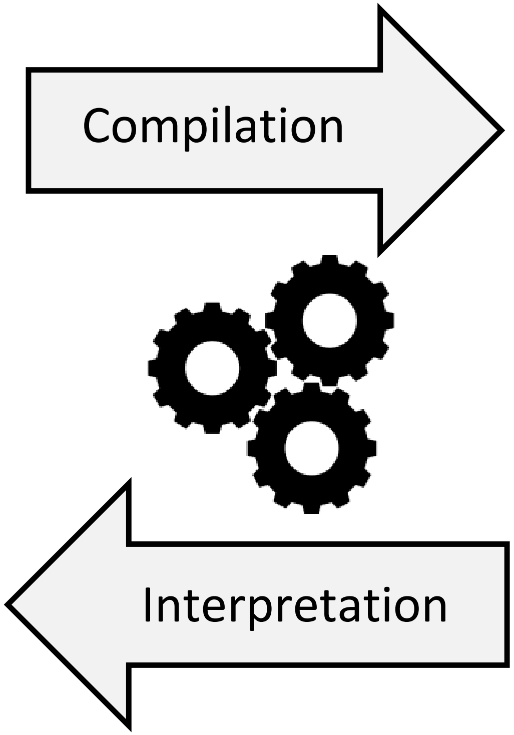


FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(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

$pX \Rrightarrow qXX$

$pX \Rrightarrow qYX$

$qY \Rrightarrow rYY$

$rY \Rrightarrow r$

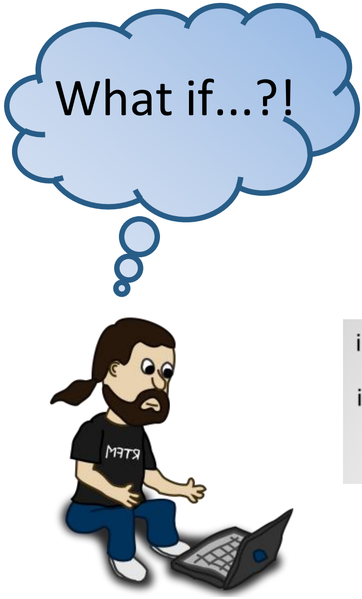
$rX \Rrightarrow pX$

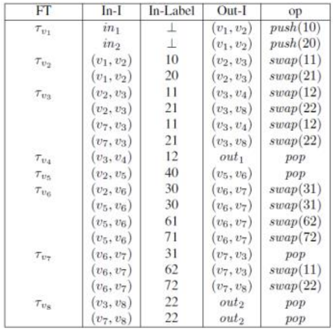
MPLS **configurations**,
Segment Routing etc.

Pushdown
Automaton and **Prefix
Rewriting Systems**
Theory

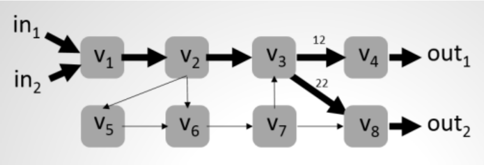
Polynomial-Ti
An Automat

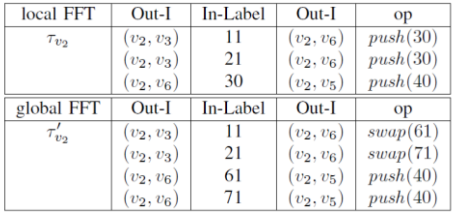
Use cases: Sysadmin issues queries to test certain properties, or do it on a regular basis automatically!



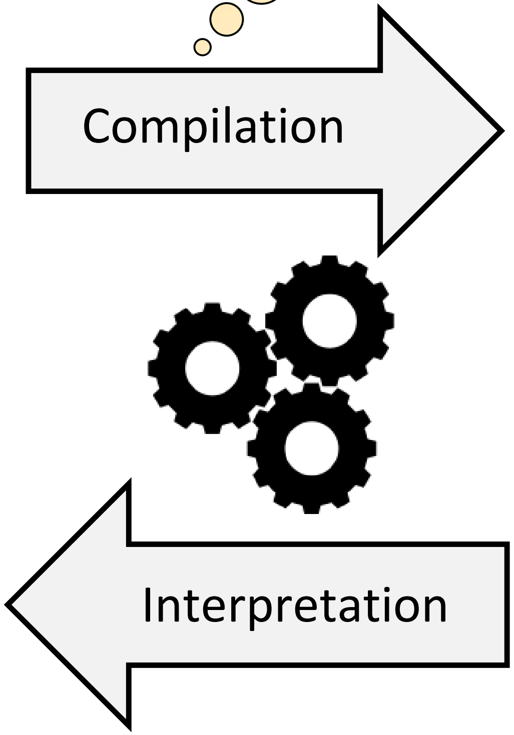


FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(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)	$push(40)$



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

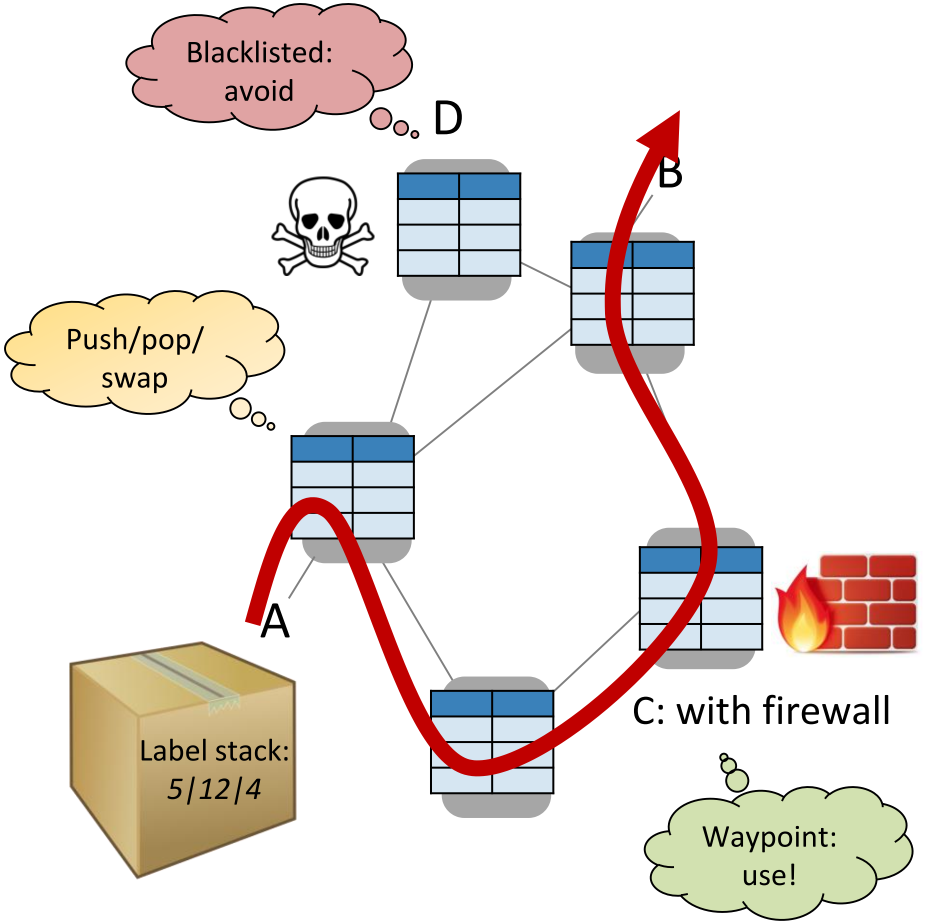
MPLS **configurations**, Segment Routing etc.

Pushdown Automaton and **Prefix Rewriting Systems** Theory

Questions with Answers in Polynomial Time

Interface Connectivity Problem

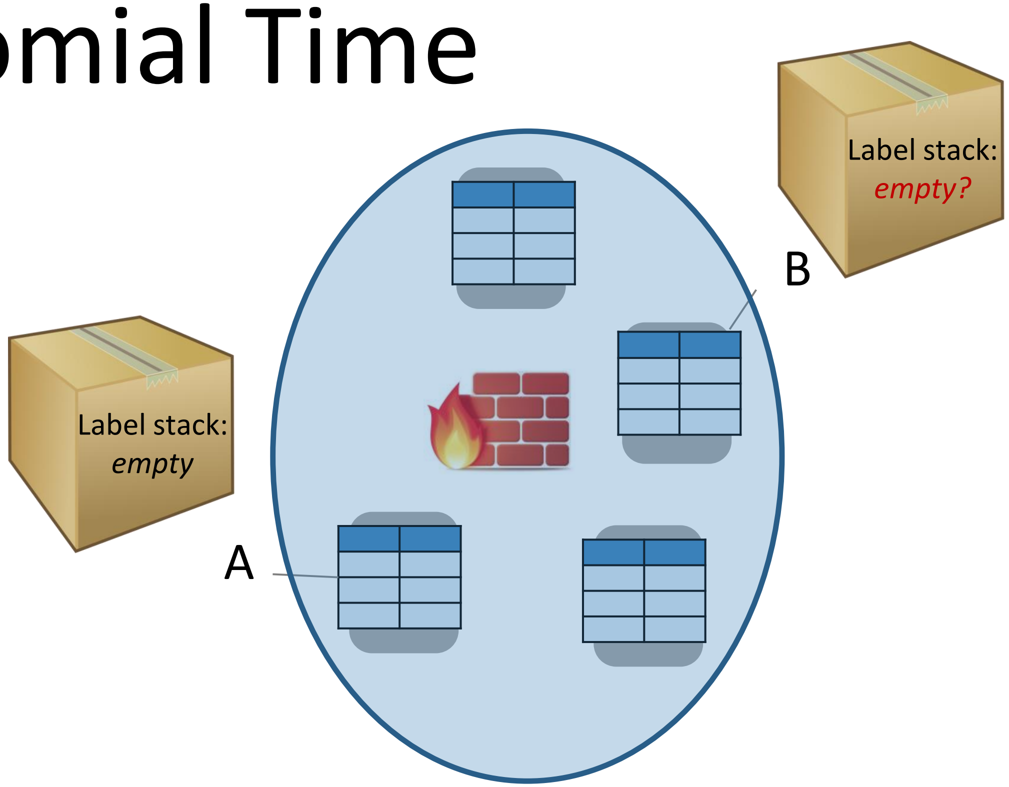
- Can a packet arriving at interface A with label-stack header h reach an interface B?
- Does the route avoid a given set of nodes?
- Will the packet always traverse a given waypoint?
- What subset of headers guarantees that a given interface is not reachable under at most k link failures?
- And everything for up to k failures!



Questions with Answers in Polynomial Time

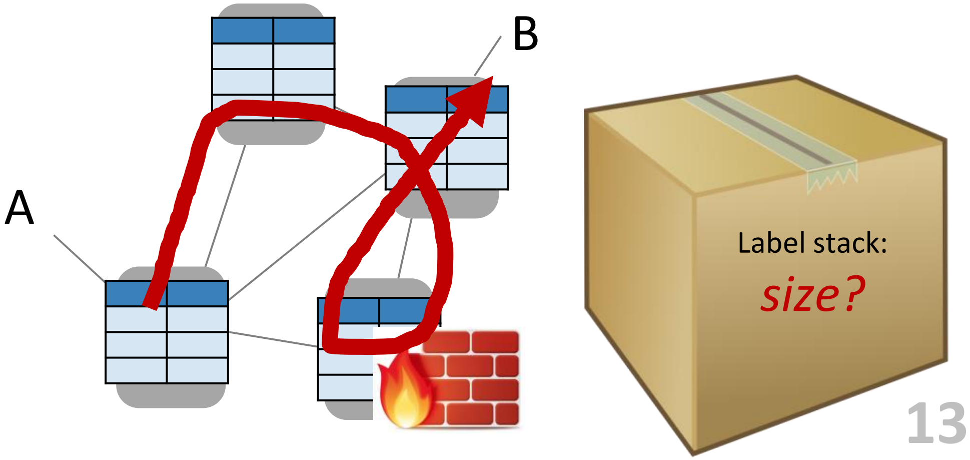
Transparency

- MPLS: transit networks!
- Will a packet with empty label-stack arriving at ingress interface A always leave at egress interface B also with the empty label-stack?
- Also under k failures?



Cyclic and repeated routing

- Will some server receive a given packet more than r-times during the routing?
- What is the max stack size during the routing?
- Under failures as well…



Our Approach

The clue: exploit the specific structure of MPLS rules

- OpenFlow rules: **arbitrary rewriting**

$$in\ x\ L^* \rightarrow out\ x\ L^*$$

VS

Header size not fixed!

- (Simplified) MPLS rules: **prefix rewriting**

FT: $in\ x\ L \rightarrow out\ x\ OP$, where $OP = \{swap, push, pop\}$

FFT: $out\ x\ L \rightarrow out\ x\ OP$, where $OP = \{swap, push, pop\}$

A Network Model

- A general network

Links

Outgoing interfaces

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$

Nodes

Incoming interfaces

Set of labels in packet header

A Network Model

- A general network

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$

Interface function

Interface function: maps outgoing interface to next hop node and incoming interface to previous hop node

$$\lambda_v : I_v^{in} \cup I_v^{out} \rightarrow V$$

That is: $(\lambda_v(in), v) \in E$ and $(v, \lambda_v(out)) \in E$

A Network Model

- A general network

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$

Routing function

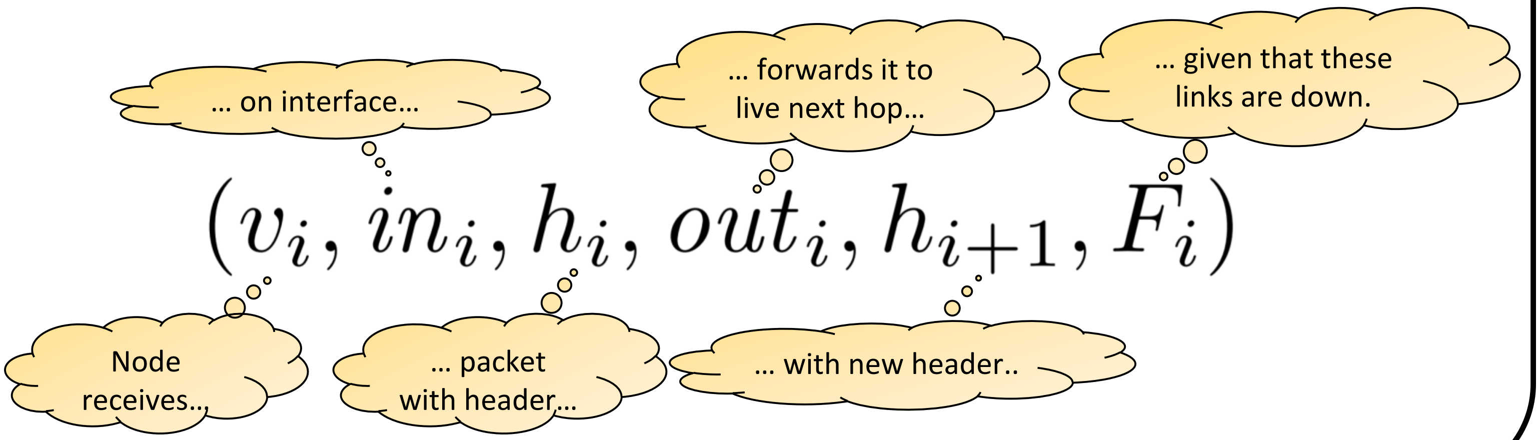
Routing function: for each set of failed links$F \subseteq E$, the routing function

$$\delta_v^F : I_v^{in} \times L^* \rightarrow 2^{(I^{out} \times L^*)}$$

defines, for all incoming interfaces and packet headers, outgoing interfaces together with modified headers.

Routing in Network

Packet routing sequence can be represented using tuples:



$$(v_i, in_i, h_i, out_i, h_{i+1}, F_i)$$

Node receives...

... on interface...

... packet with header...

... forwards it to live next hop...

... with new header..

... given that these links are down.

- Packet **routing** is then (in)finite sequence of tuples

$$(v_1, in_1, h_1, out_1, h_2, F_1),$$

$$(v_2, in_2, h_2, out_2, h_3, F_2),$$

$$\dots$$

MPLS Network Model

- MPLS supports three **operations** on header sequences:

$$Op = \{swap(\ell) \mid \ell \in L\} \cup \{push(\ell) \mid \ell \in L\} \cup \{pop\}$$

- The **local routing table** can then be defined as

$$\tau_v : I_v^{in} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

Interface + label

Maps to next hop and operation

- Local **link protection** function suggests backup interface

$$\pi_v : I_v^{out} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

protected

backup

typically: push

MPLS Pushdown Prefix Rewriting System

- Prefix rewriting system is set of **rewriting rules** $R \subseteq \Gamma^* \times \Gamma^*$

- We write $v \rightarrow w$ for $(v, w) \in R$ generates a **transition system** $G_R = (\Gamma^*, \rightarrow_R)$ such that $vt \rightarrow_R wt$ iff $t \in \Gamma^*$

Replace prefix

- Prefix rewriting system is called **pushdown system** if $|v| = 2$ and $1 \leq |w| \leq 3$ for all $(v, w) \in R$

Second symbol of v: top of stack label.

First symbol of v and w: control state of pushdown system.

$\|w\| = 1$	pop
$\|w\| = 2$	swap
$\|w\| = 3$	push

MPLS Pushdown Prefix Rewriting System

- **Control states**: (v, in) and (v, out, i)

Node and incoming link

How many times have we tried to reroute at this node already?

- **Labels**: stack symbols and $\perp$ at bottom

- Packet with header h arriving at interface in at v represented as **pushdown configuration**: $(v, in)h\perp$

- Packet to be forwarded at node v to outgoing interface out represented by **configuration**: $(v, out, i)h\perp$

Example Rules:
Regular Forwarding on Top-Most Label

Push:

Push label on stack

$$(v, in)\ell \rightarrow (v, out, 0)\ell'\ell \text{ if } \tau_v(in, \ell) = (out, push(\ell'))$$

Swap:

Swap top of stack

$$(v, in)\ell \rightarrow (v, out, 0)\ell' \text{ if } \tau_v(in, \ell) = (out, swap(\ell'))$$

Pop:

Pop top of stack

$$(v, in)\ell \rightarrow (v, out, 0) \text{ if } \tau_v(in, \ell) = (out, pop)$$

Example *Failover* Rules

Emumerate all rerouting options

Failover-Push:

$$(v, out, i)\ell \rightarrow (v, out', i+1)\ell'\ell \text{ for every } i,\ 0 \leq i < k,$$
$$\text{where } \pi_v(out, \ell) = (out', push(\ell'))$$

Failover-Swap:

$$(v, out, i)\ell \rightarrow (v, out', i+1)\ell' \text{ for every } i,\ 0 \leq i < k,$$
$$\text{where } \pi_v(out, \ell) = (out', swap(\ell')),$$

Failover-Pop:

$$(v, out, i)\ell \rightarrow (v, out', i+1) \text{ for every } i,\ 0 \leq i < k,$$
$$\text{where } \pi_v(out, \ell) = (out', pop).$$

Example rewriting sequence:

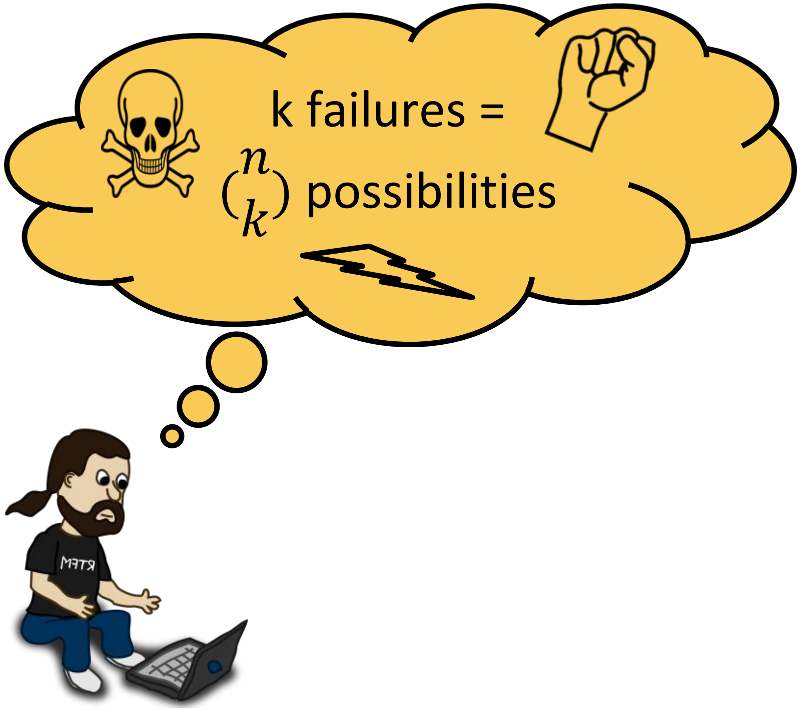
$$(v_1, in_1)h_1\bot \rightarrow (v_1, out, 0)h\bot \rightarrow (v_1, out', 1)h'\bot \rightarrow (v_1, out'', 2)h''\bot \rightarrow \ldots$$

Try default

Try first backup

Try second backup

Why Polynomial Time?!

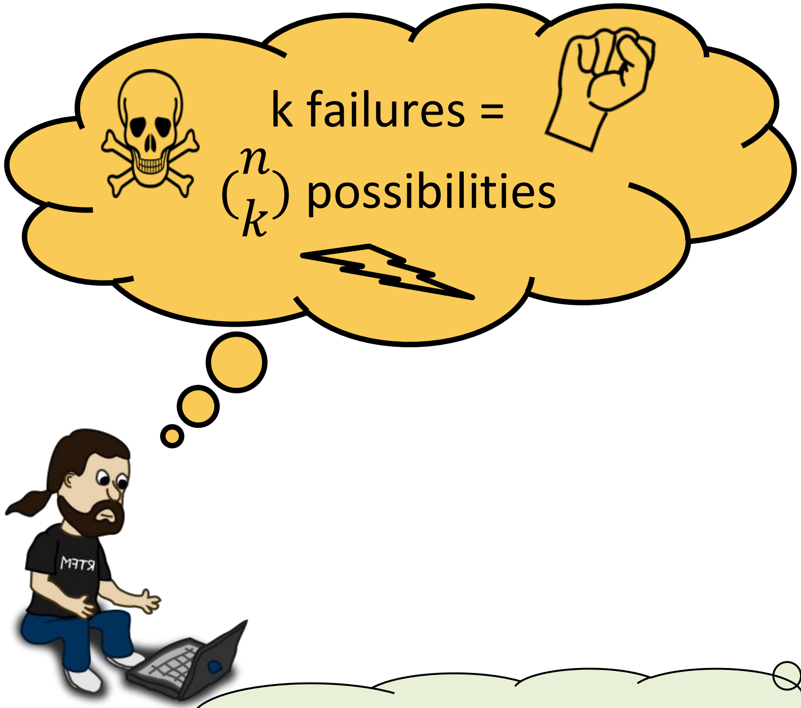


k failures = $\binom{n}{k}$ possibilities

- Arbitrary number k of failures: How can I avoid checking all $\binom{n}{k}$ many options?!

- Even if we reduce to **push-down automaton**: simple operations such as emptiness testing or intersection on Push-Down Automata (PDA) is computationally non-trivial and sometimes even **undecidable**!

Why Polynomial Time?!

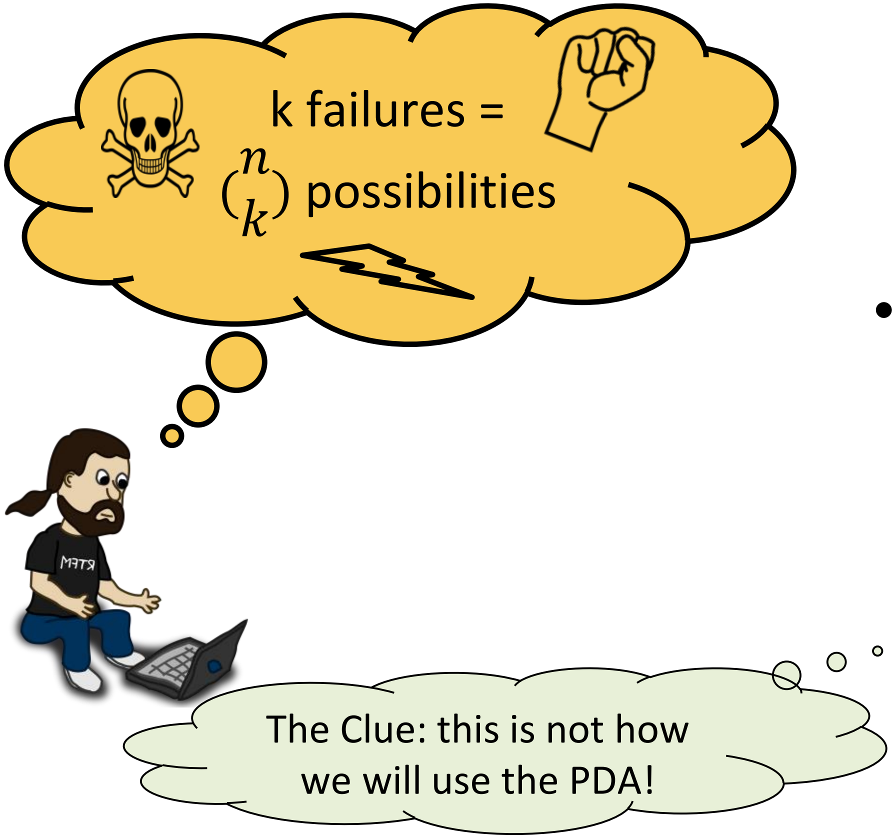


The Clue: this is not how we will use the PDA!

- Arbitrary number k of failures: How can I avoid checking all $\binom{n}{k}$ many options?!

- Even if we reduce to **push-down automaton**: simple operations such as emptiness testing or intersection on Push-Down Automata (PDA) is computationally non-trivial and sometimes even **undecidable**!

Why Polynomial Time?!



k failures = $\binom{n}{k}$ possibilities

The Clue: this is not how we will use the PDA!

- Arbitrary number k of failures: How can I avoid checking all $\binom{n}{k}$ many options?!
- Even if we reduce to **push-down automaton**: simple operations such as emptiness testing or intersection on Push-Down Automata (PDA) is computationally non-trivial and sometimes even **undecidable**!

The words in our language are sequences of pushdown stack symbols, not the labels of transitions.

Time for Automata Theory!

- Classic result by **Büchi** 1964: the set of all reachable configurations of a pushdown automaton a is regular set

- Hence, we can operate only on Nondeterministic Finite Automata (NFAs) when reasoning about the pushdown automata

- The resulting **regular operations** are all polynomial time

- Important result of **model checking**

Julius Richard Büchi

1924-1984

Swiss logician

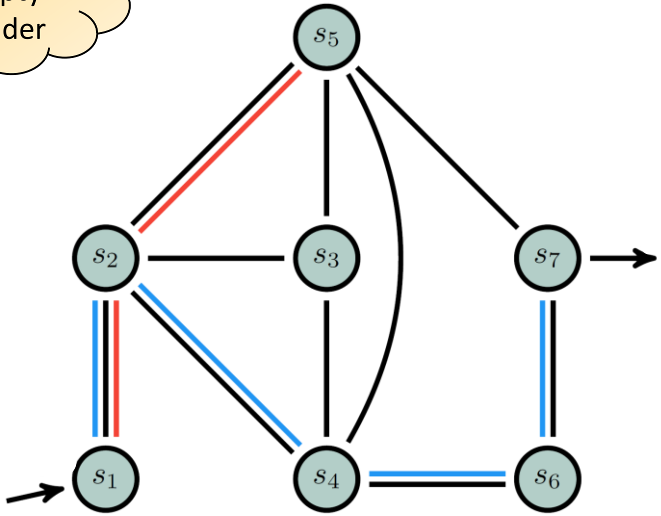
Preliminary Query Language: Example

Question: Beginning with an empty header [], can we get from s1 to s7 in any number of steps, and end with an empty header []?

Take multiple steps

Query: []s1 >> s7[]

Empty header



Output: Yes and witness trace (excerpt)

```
YES.
```

!

```
--- START ---
build_0
 <_e>
simstart (path_counter=0)
 <_e>
s1_i1 (path_counter=0)
 <_e>
s1_i1 (path_counter=0)
 <_e>
s1_s2_0 (path_counter=0)
 <_10 _e>
s1_s2_0 (path_counter=1)
 <_10 _e>


s7_i1_0 (path_counter=2)
 <_e>
simend (path_counter=0)
 <_e>
destroy_0
 <_e>
destroy_1
 <_e>
complete
 <_e>
  [ target reached ]
```

!

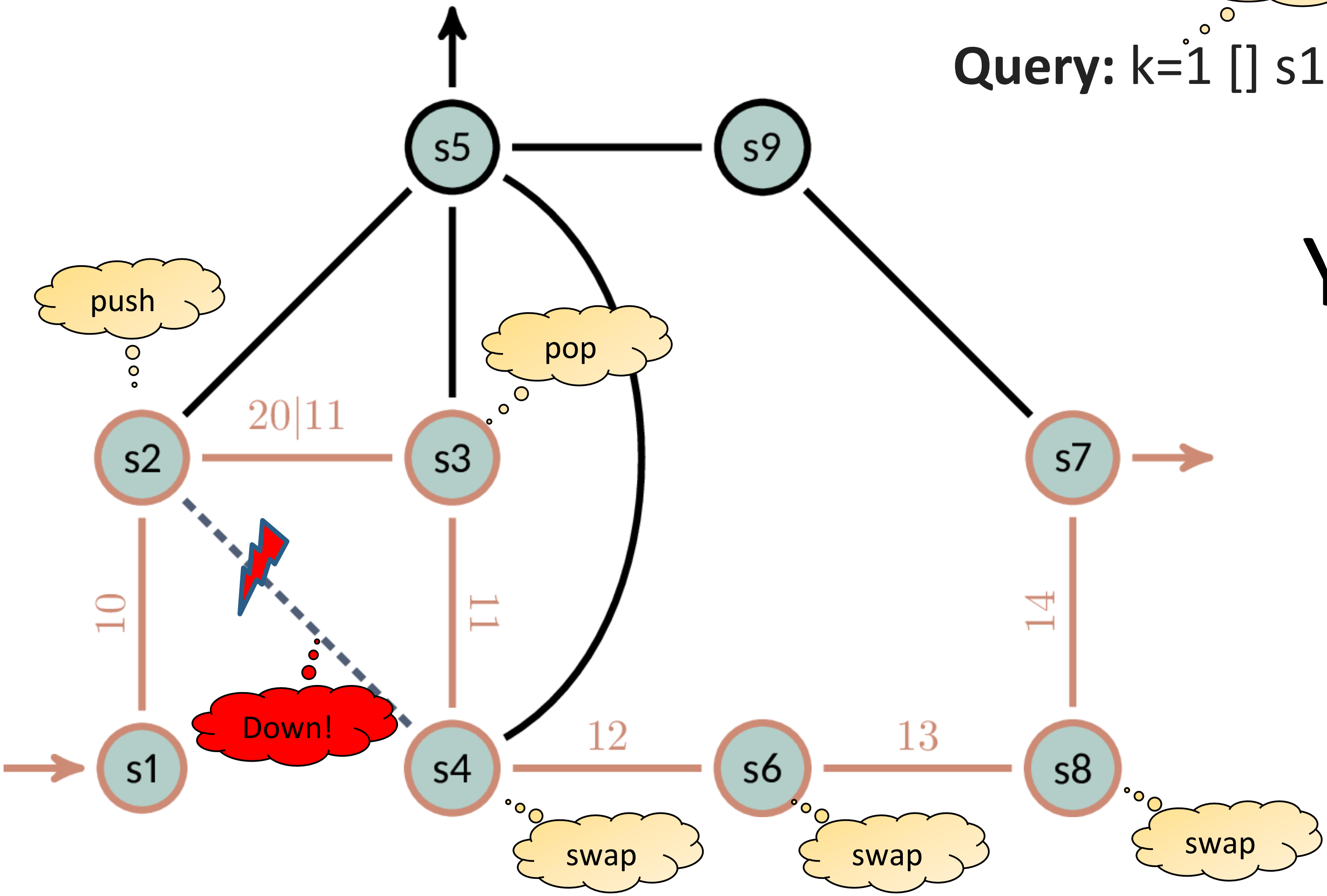
Example 2: Traversal Testing

Traversal test: Can traffic starting with [] go through s3, under up to k=1 failures?

1 failure

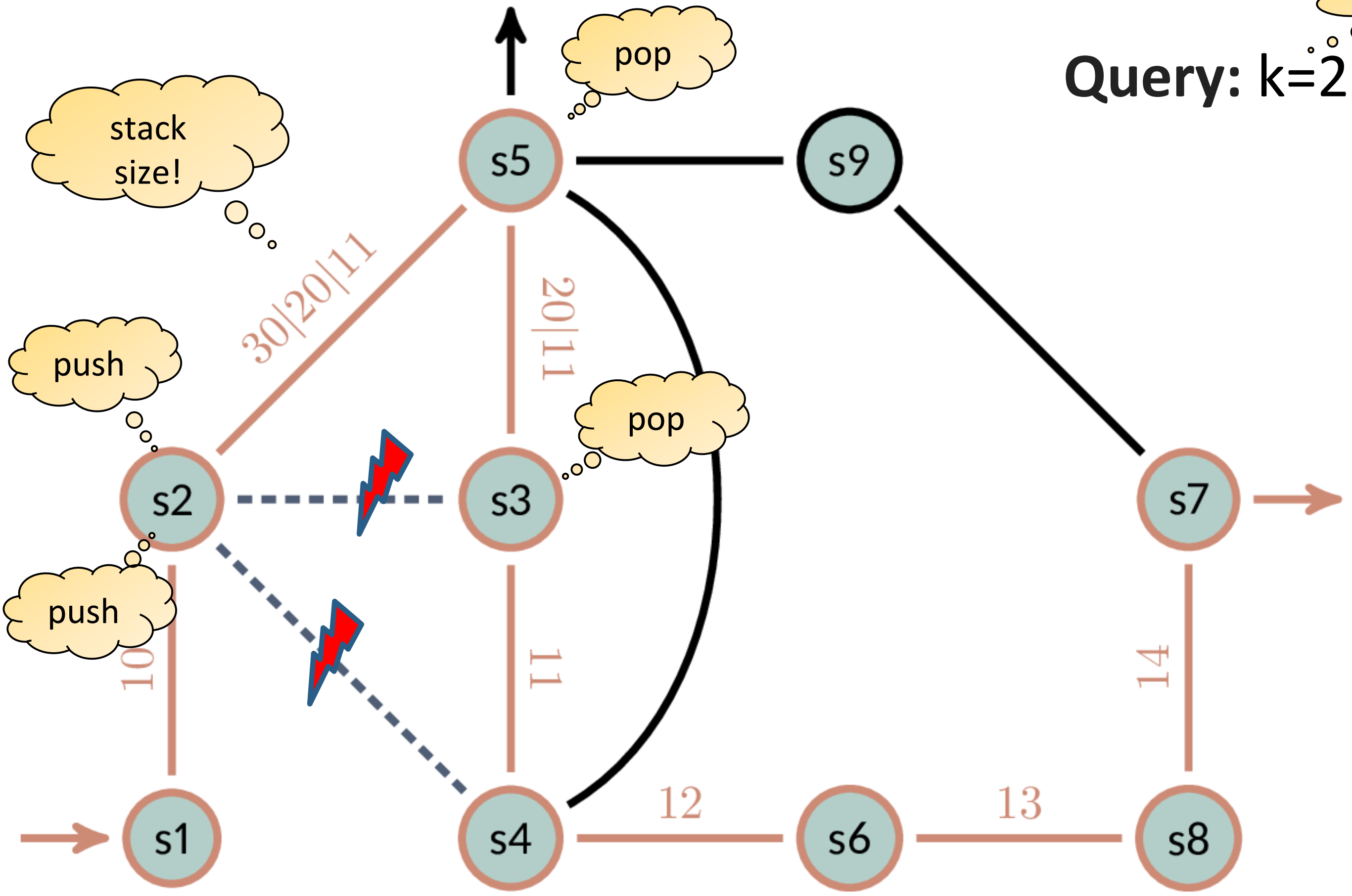
Query: k=1 [] s1 >> s3 >> s7 []

YES!



Example 3: Traversal with 2 Failures

Traversal test with k=2: Can traffic go through s5, under up to k=2 failures?



2 failures

Query: k=2 [] s1 >> s5 >> s7 []

YES!

Example 4: Transparency Violation

Transparency with k=3: Can transparency be violated under up to k=3 failures?

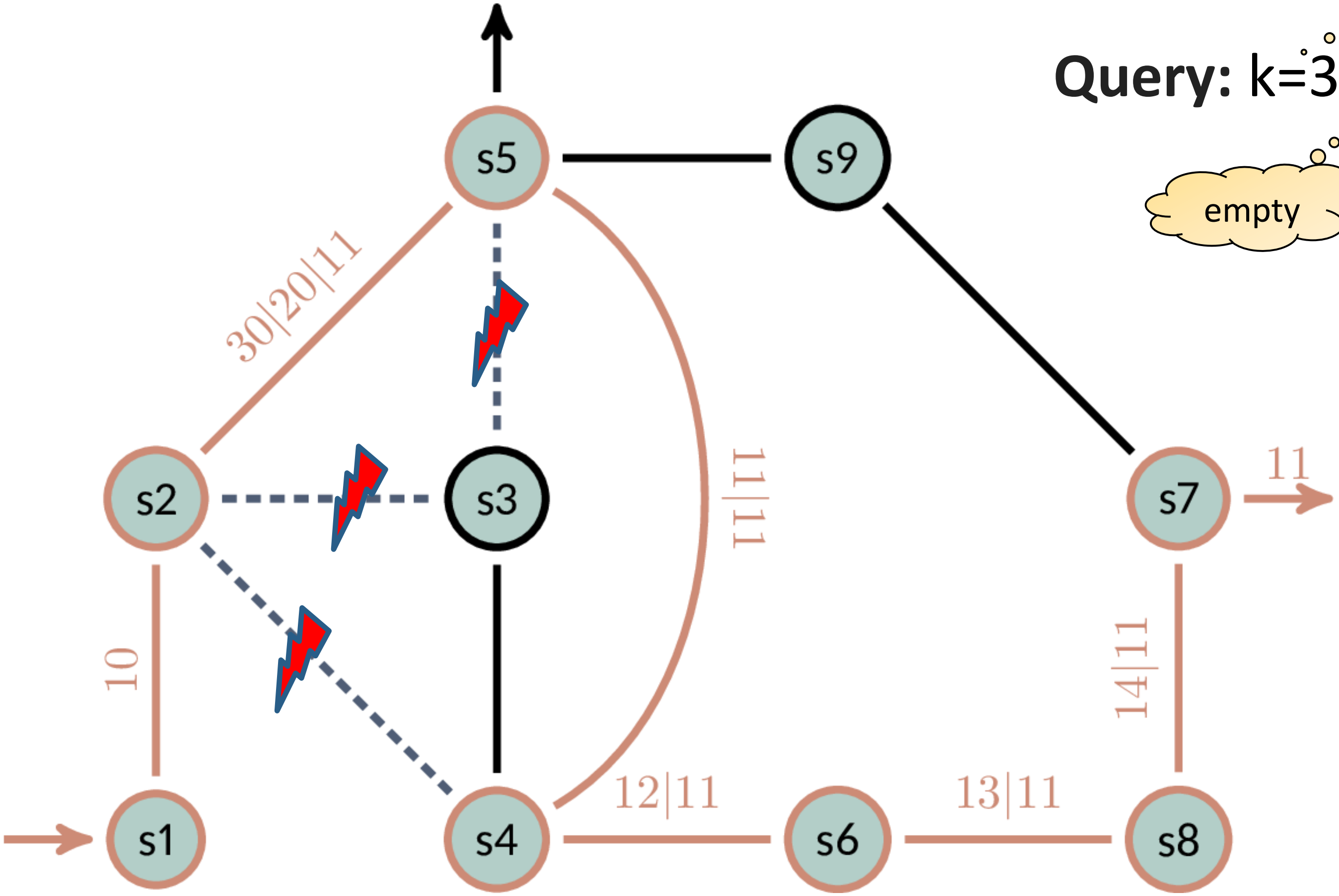
3 failures

Query: k=3 [] s1 >> s7 [+]

empty

non-empty

YES!



Root cause is a misconfiguration in s5, causing it to swap to 11 instead of popping when doing the failover on s5-s4.

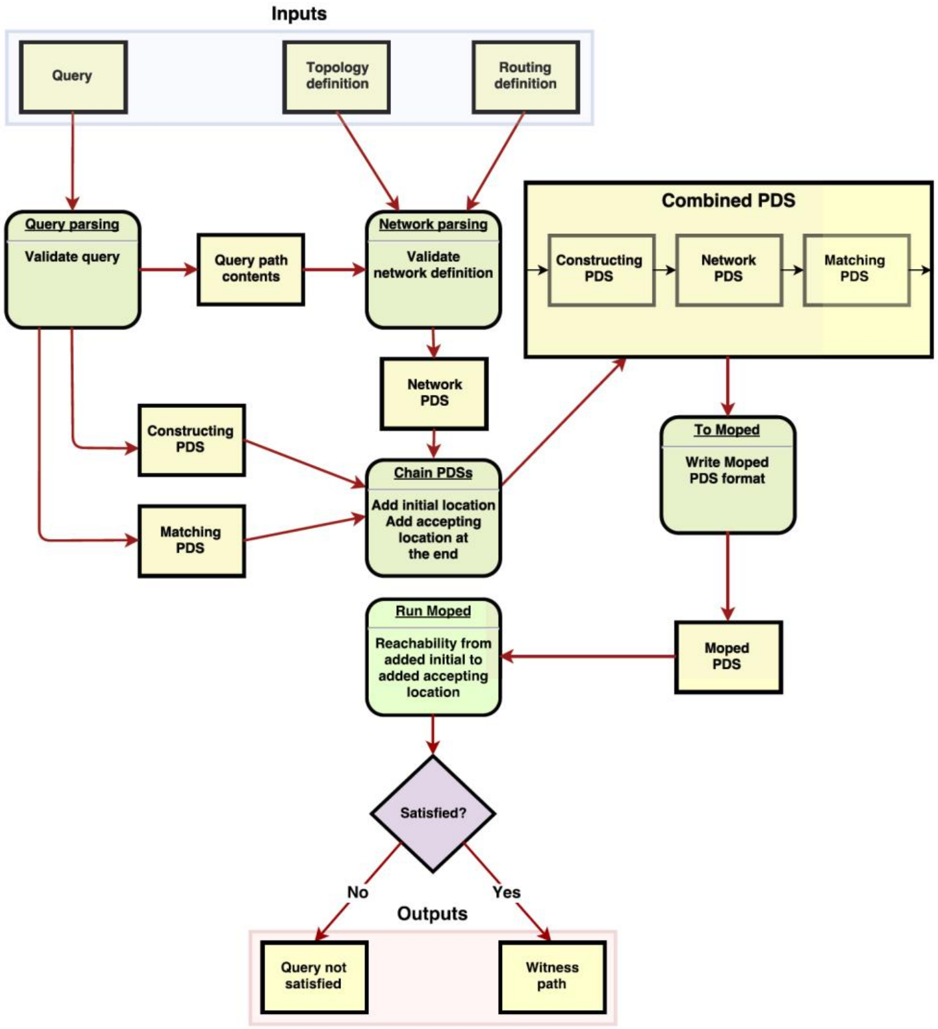
Preliminary Tool

Part 1: Parses query and constructs Push-Down System (PDS)

- In Python 3

Part 2: Reachability analysis of constructed PDS

- Using Moped tool



query processing flow

Preliminary Evaluation

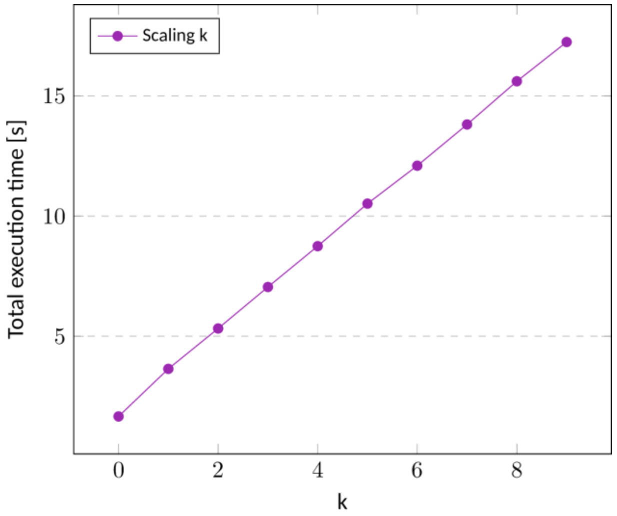
For small queries fast: 1000s of links, within seconds

Links	Switches	Network size	Build	Verify	Total	Query size	PDS transitions
104	36	140	0.35	0.327	0.677	30	10658
224	72	296	0.531	0.365	0.896	30	16890
464	144	608	0.939	0.43	1.369	30	29930
944	288	1232	1.742	0.654	2.396	30	56010
1904	576	2480	3.342	0.993	4.335	30	108170
3824	1152	4976	6.734	1.789	8.523	30	212490

100,000s

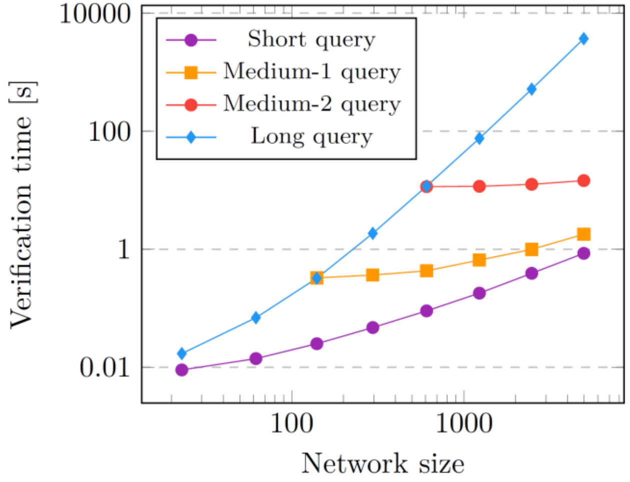
1000s

secs



# failures affects performance only linearly!

Bottleneck are large queries



Summary

- Polynomial-time verification of MPLS reachability and policy-related properties like waypointing
  - For arbitrary number of failures (up to linear in n)!
  - Supports arbitrary header sizes („infinite“)
  - Also allows to compute headers which do (not) fulfill a property
  - Allows to support a constant number of stateful nodes as well
  - Extends to Segment Routing networks based on MPLS (SR-MPLS)

- Leveraging theory from Prefix Rewriting Systems and Büchi‘s classic result

Future Work

- **Other networks and properties** which can be verified in polynomial time?
- Good tradeoff **expressiveness vs polynomial-time** verifiability?
- We‘re looking for **industrial case studies** and collaborations

Thank you! Questions?

Further Reading

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.

WNetKAT: A Weighted SDN Programming and Verification Language
Kim G. Larsen, Stefan Schmid, and Bingtian Xue.
20th International Conference on Principles of Distributed Systems (**OPODIS**), Madrid, Spain, December 2016.

TI-MFA: Keep Calm and Reroute Segments Fast
Klaus-Tycho Foerster, Mahmoud Parham, Marco Chiesa, and Stefan Schmid.
IEEE Global Internet Symposium (**GI**), Honolulu, Hawaii, USA, April 2018.

Local Fast Failover Routing With Low Stretch
Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
ACM SIGCOMM Computer Communication Review (**CCR**), 2018.