Emerging Communication Networks: A Case for Automation – and Formal Methods?

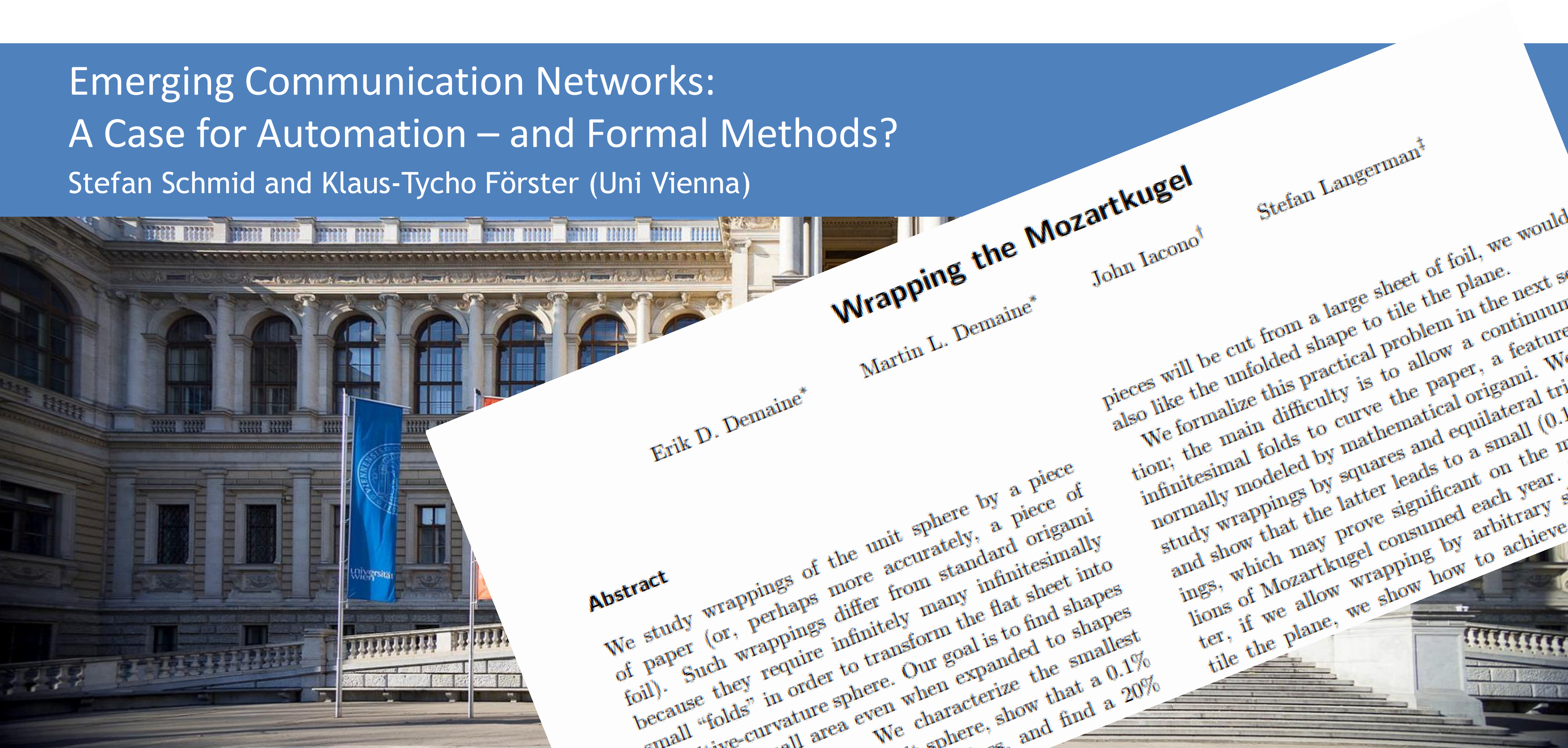
Stefan Schmid and Klaus-Tycho Förster (Uni Vienna)

Emerging Communication Networks:
A Case for Automation – and Formal Methods?

Stefan Schmid and Klaus-Tycho Förster (Uni Vienna)

Emerging Communication Networks:
A Case for Automation – and Formal Methods?

Stefan Schmid and Klaus-Tycho Förster (Uni Vienna)



Emerging Communication Networks: A Case for Automation – and Formal Methods?

Stefan Schmid and Klaus-Tycho Förster (Uni Vienna)

NORDUnet
Nordic Gateway for Research & Education

Flexibilities: Great Time for Networking Research!

Passau, Germany

Inn, Donau, Ilz

Flexibilities: Great Time for Networking Research!

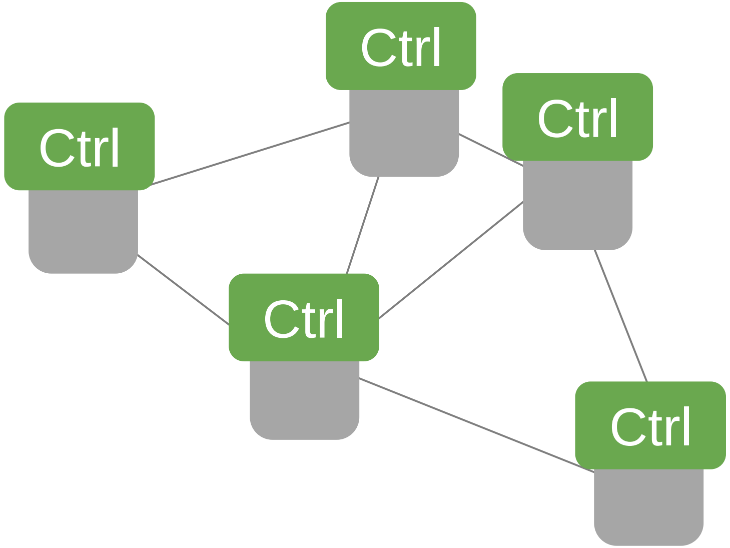
Passau, Germany

Inn, Donau, Ilz

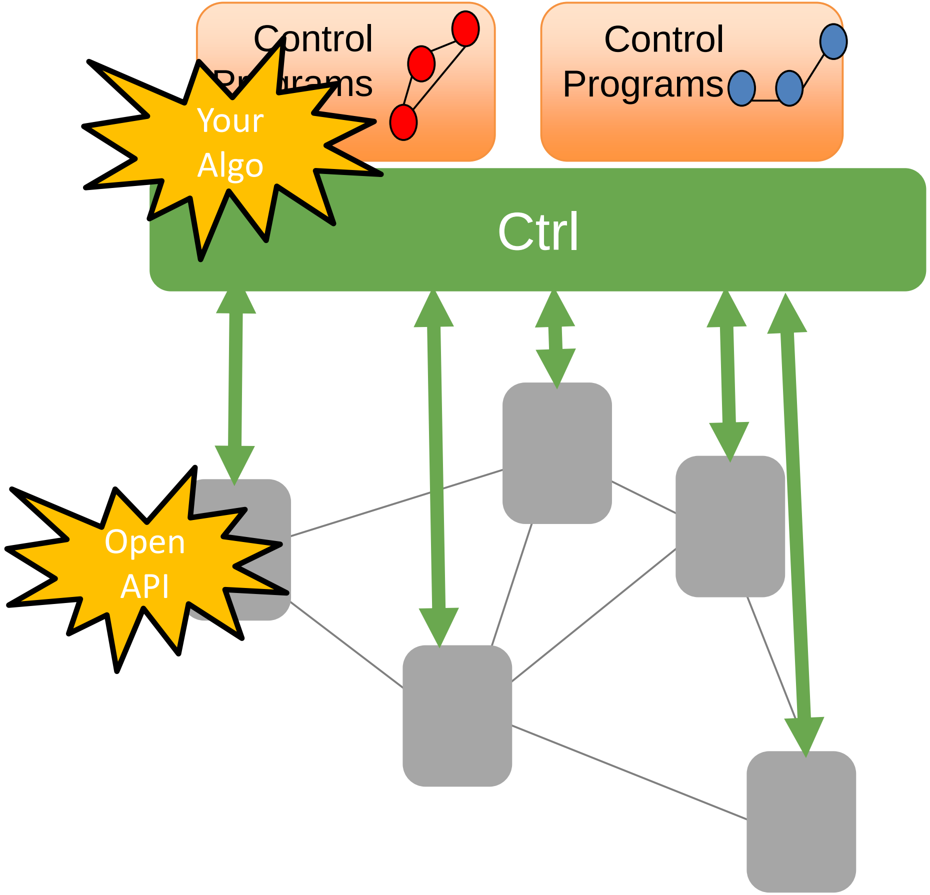
Flexibilities: Great Time for Networking Research!

u, Germany
Inn, Donau, Ilz

Remember? SDN
(„The Linux of Networking“)

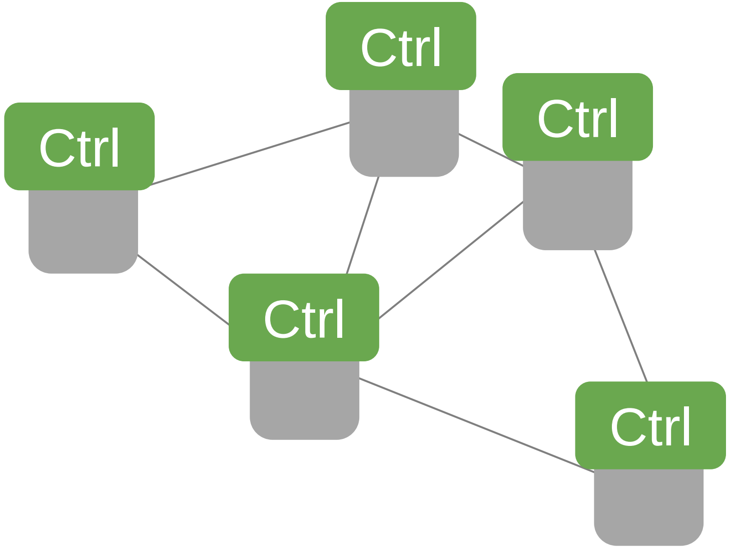


Traditionally: Proprietary

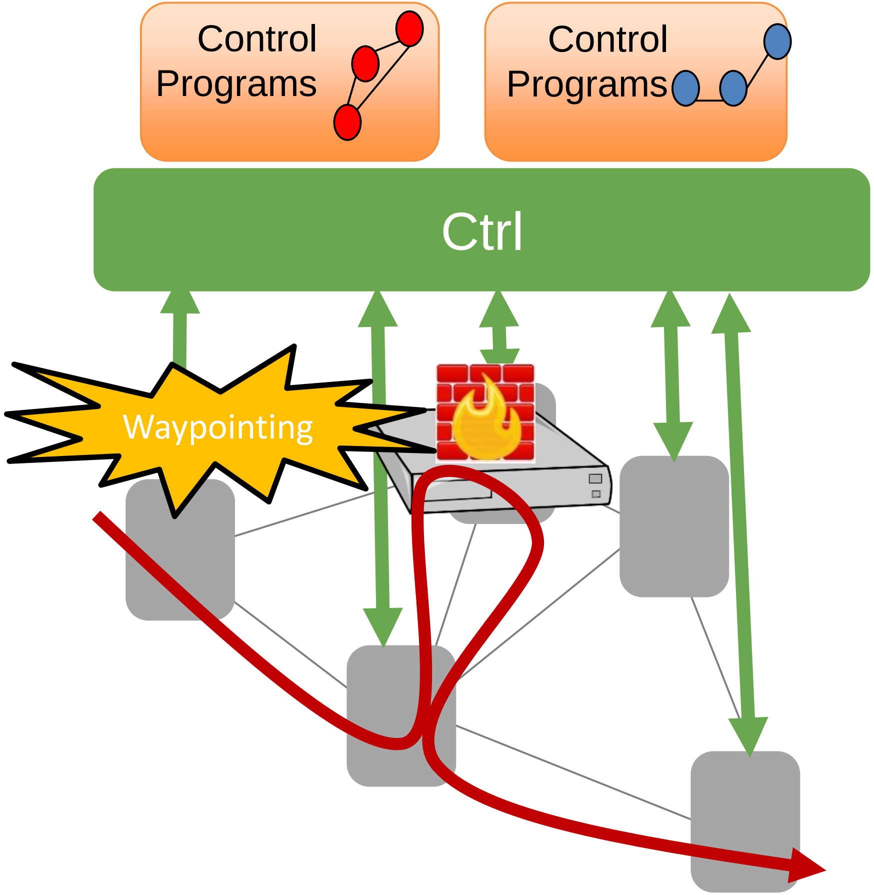


SDN: Programmable

Virtualization
(Flexible Placement and New Services)

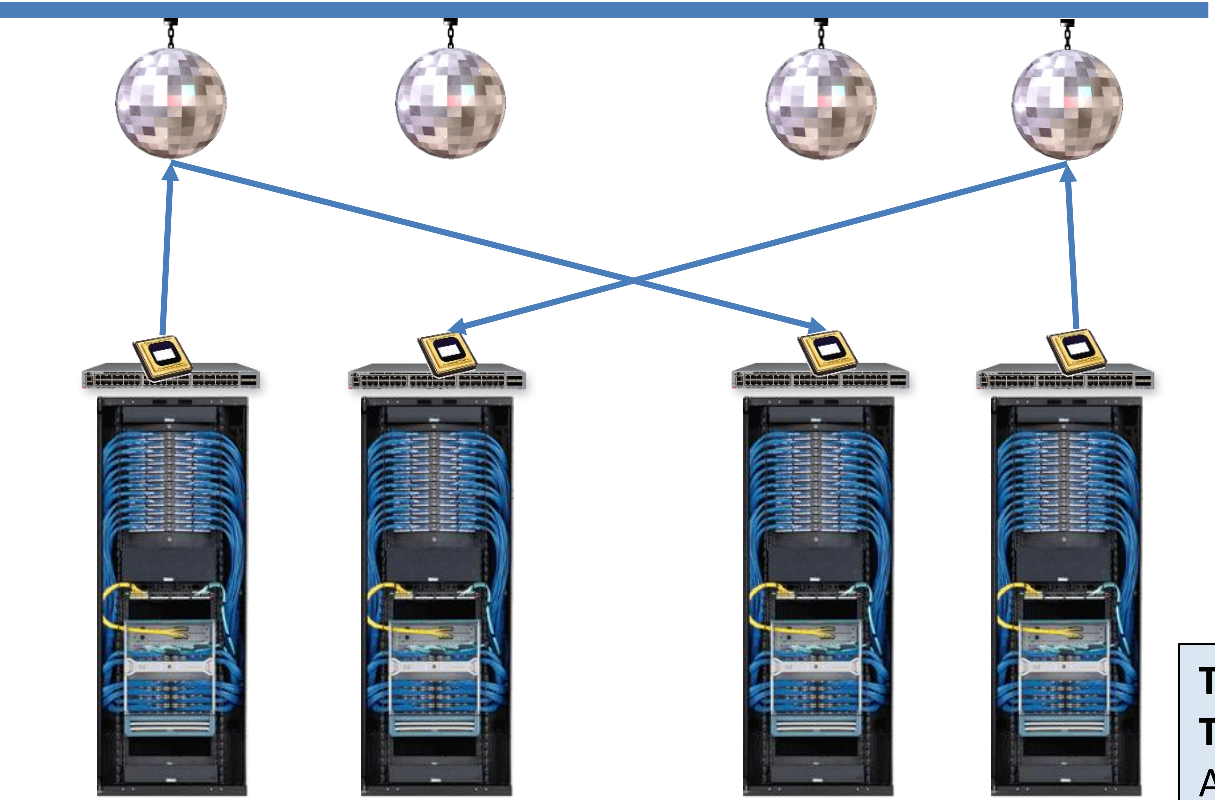


Traditionally: Proprietary



SDN: Programmable

Remember? Topology Programming

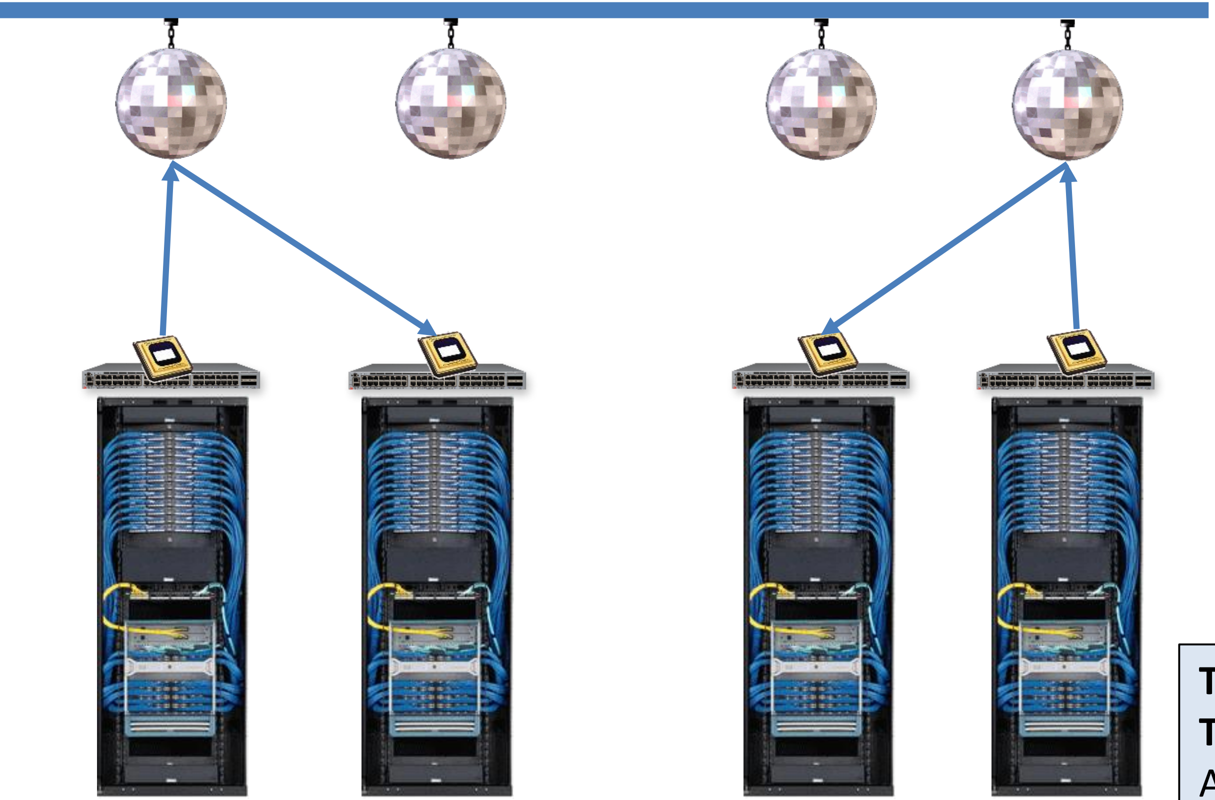


- **Reconfigure** networks towards needs

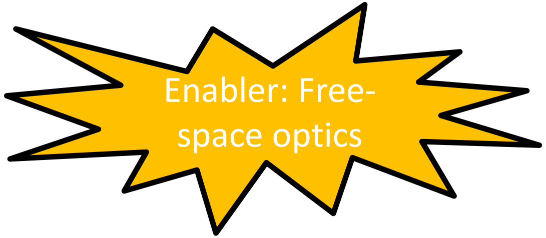
Enabler: Free-space optics

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks.
Avin et al. ACM SIGCOMM CCR, 2018.

Remember? Topology Programming



- **Reconfigure** networks towards needs



Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks.
Avin et al. ACM SIGCOMM CCR, 2018.

Opportunity

- Additional **dimensions for optimization**: can be exploited to improve performance, utilization, …
- New network **services** (e.g., service chaining)

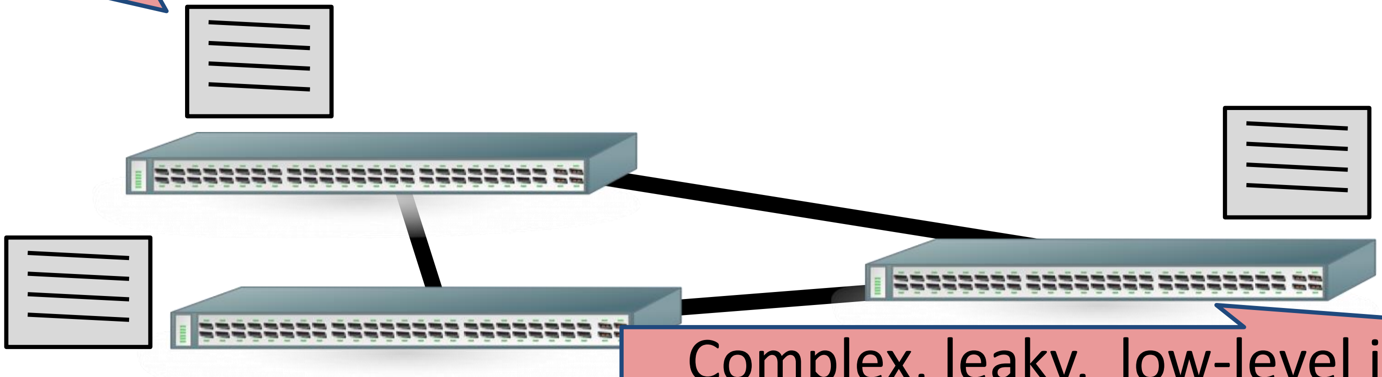
Challenge

- But: optimizations become **harder** and are somtimes not yet well-understood (e.g., embedding, topology programming)

Another Challenge: Complexity

Manual, device-centric network configuration
(CLI, LANmanager)

Un-evolved Best Practices
(tcpdump, traceroute - from the 1990s)



Complex, leaky, low-level interfaces
(VLANs, Spanning Tree, Routing)

Operating networks today: **manual** and error-prone task.

Complexity is Problematic

Datacenter, enterprise, carrier networks have become mission-critical infrastructure!
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 webservices

Go Daddy.com

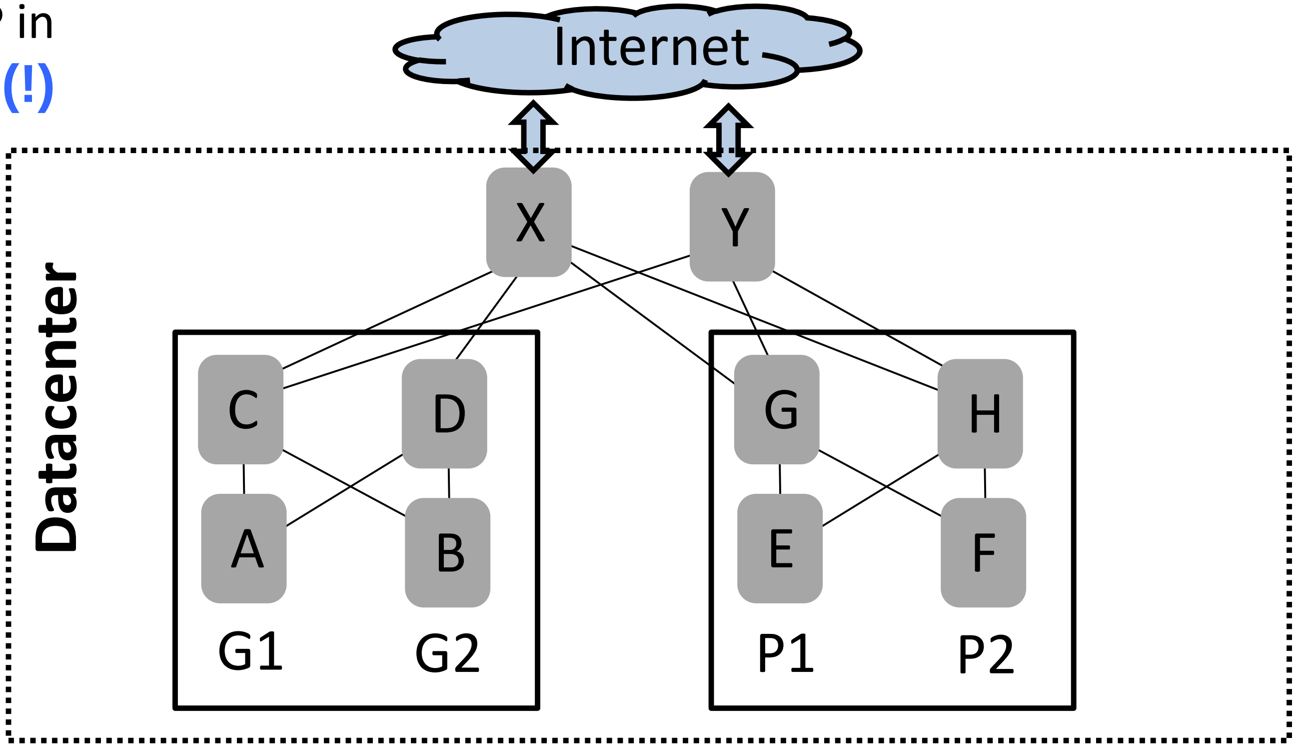
Service outage was due to a series of internal network events that corrupted router data tables

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

United Airlines

Reasoning About Failures is Particularly Hard

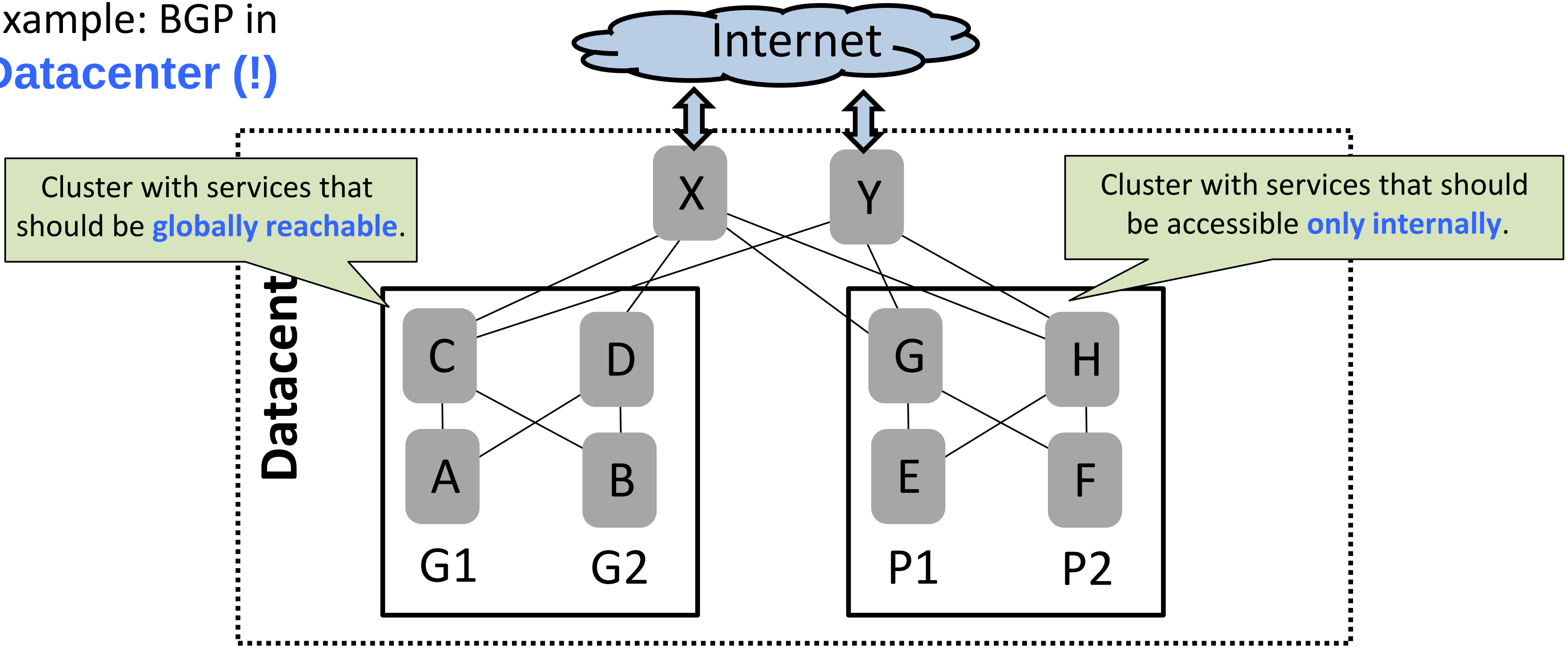
Example: BGP in
Datacenter (!)



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

Reasoning About Failures is Particularly Hard

Example: BGP in
Datacenter (!)

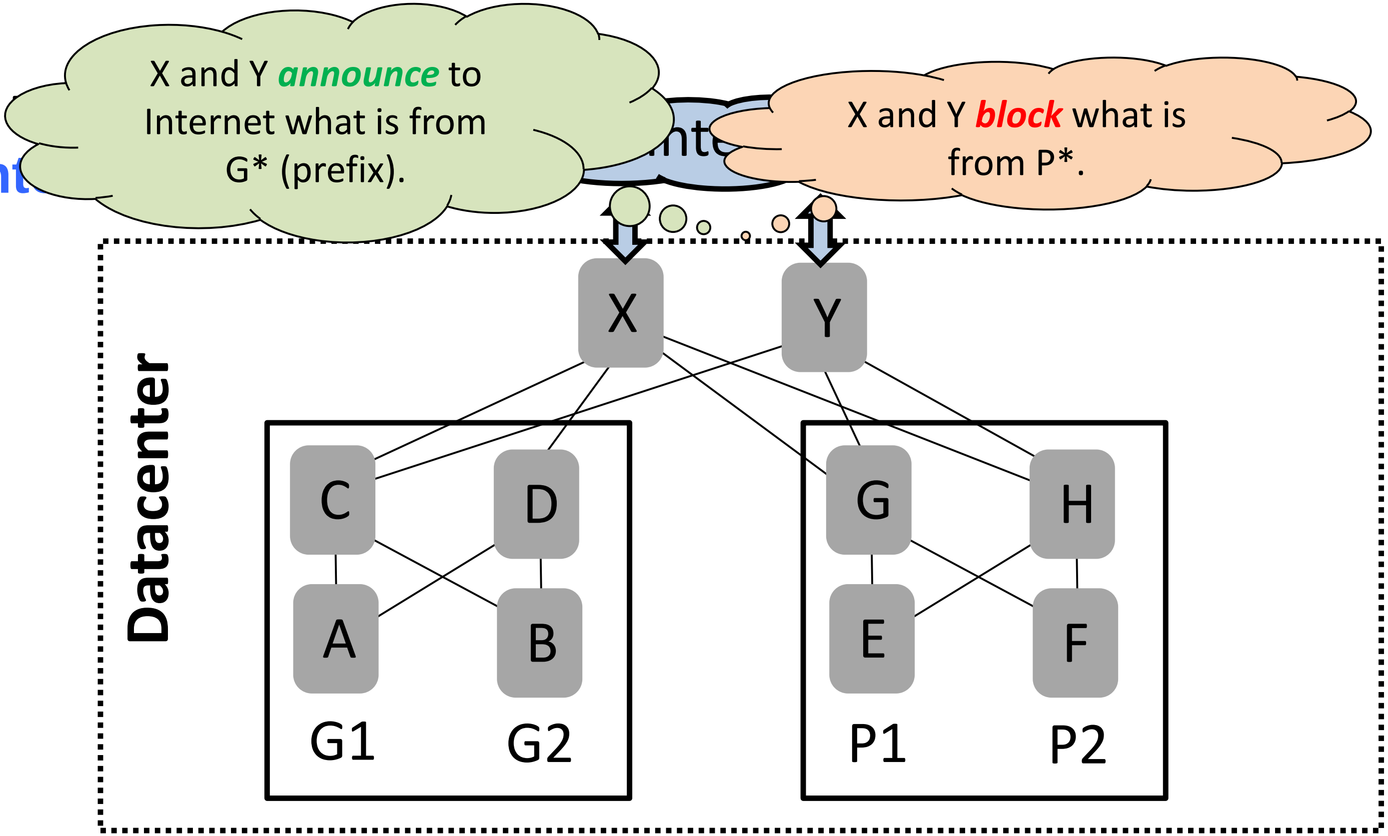


Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Reasoning About Failures is Particularly Hard

Example:

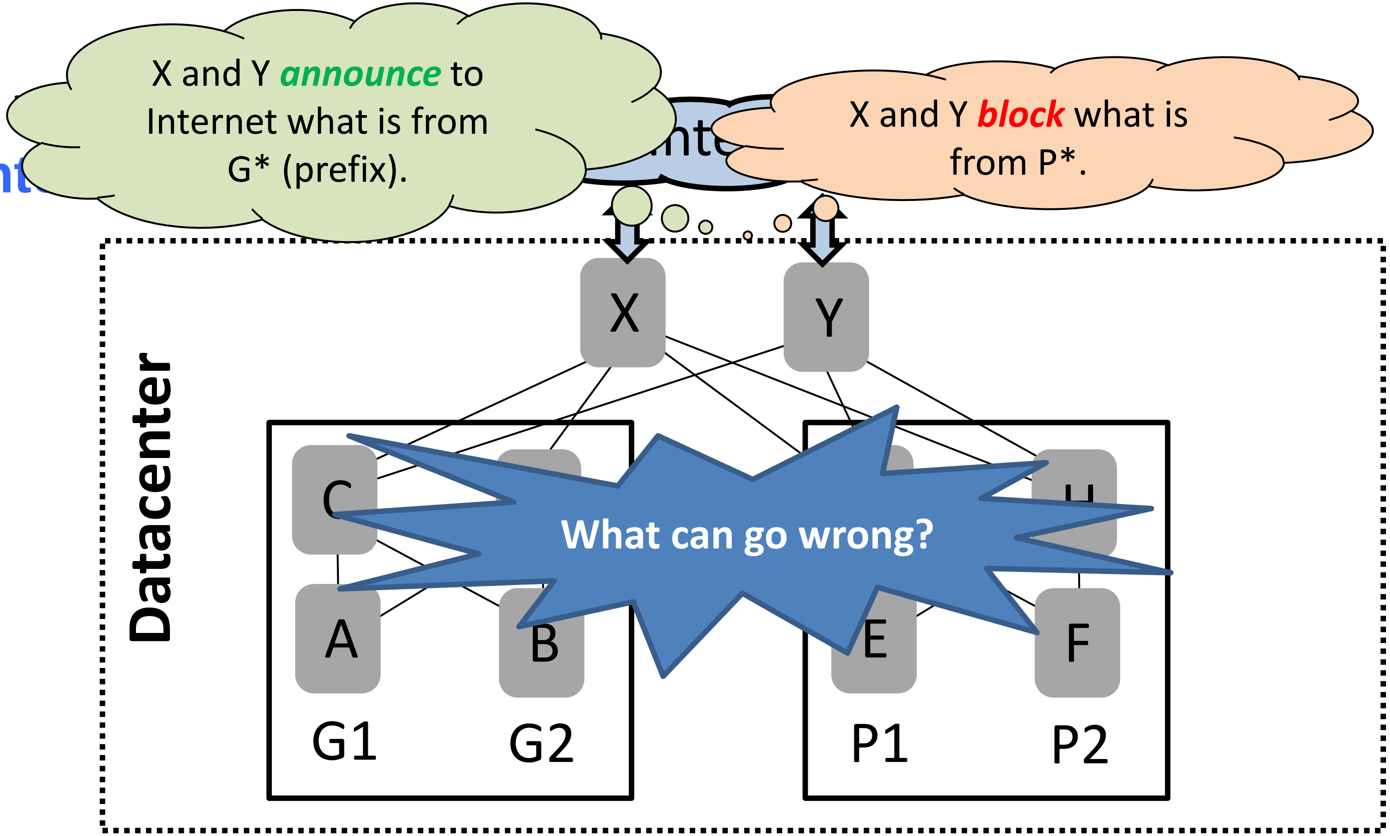
Datacent



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

Reasoning About Failures is Particularly Hard

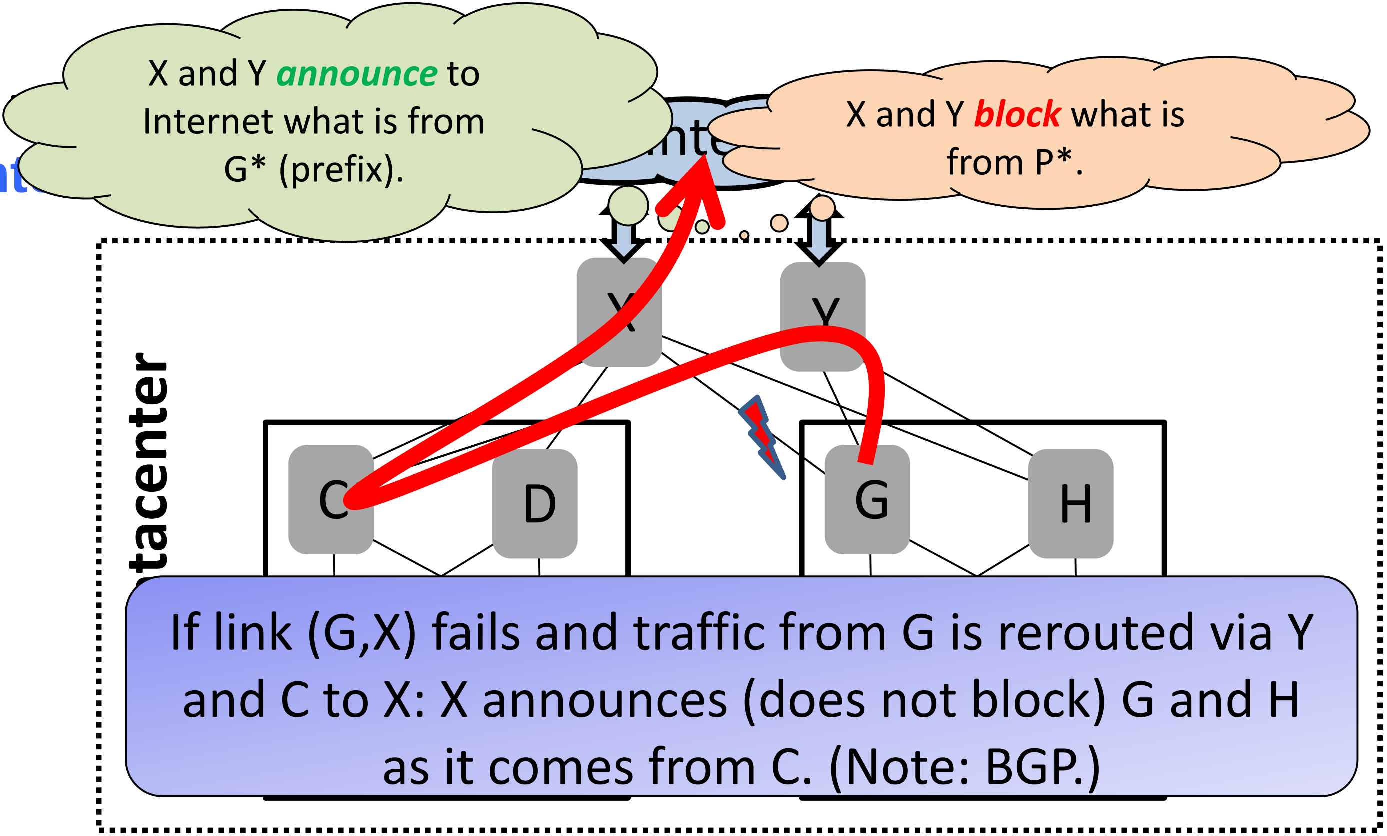
Example:
Datacent



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

Reasoning About Failures is Particularly Hard

Example:
Datacent



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

Let’s give up control: self-* networks!

Self-repairing, self-optimizing, “self-driving”, …

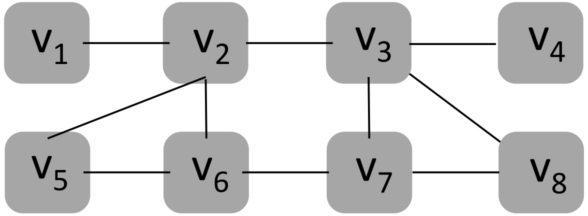
It’s about automation!

Roadmap

- 1st Use Case for Automation: What-if Analysis
- 2nd Use Case for Automation: Consistent Rerouting

Example: Self-Repairing MPLS Networks

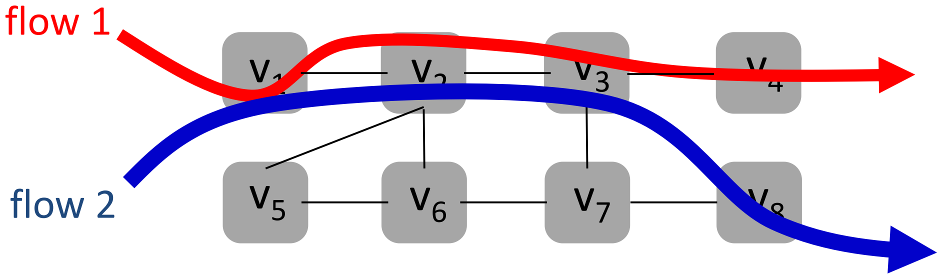
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

Example: Self-Repairing MPLS Networks

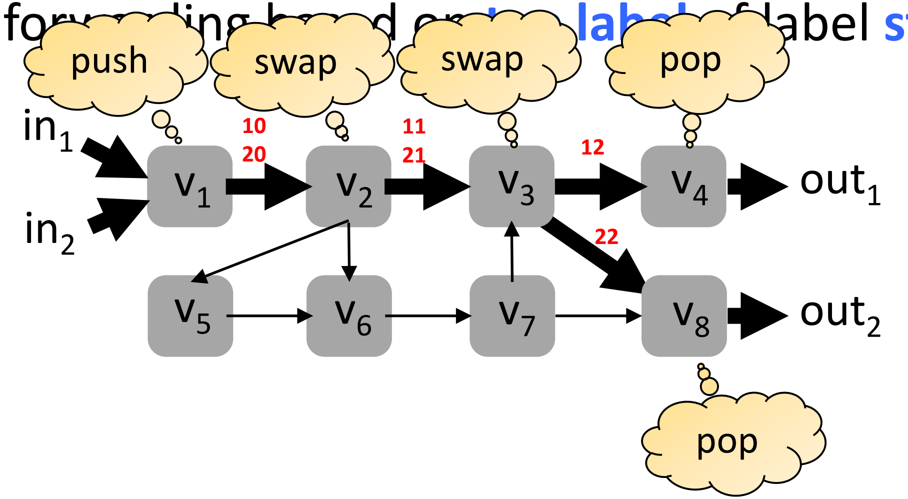
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

Example: Self-Repairing MPLS Networks

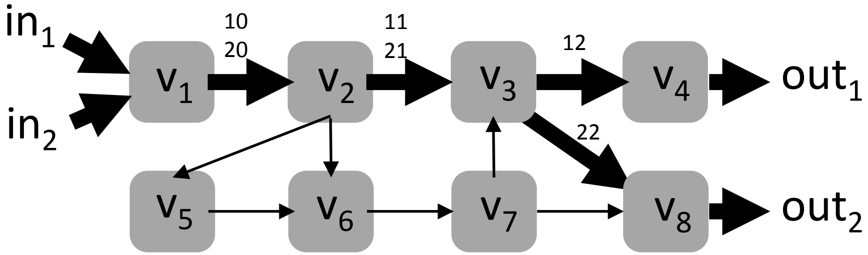
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

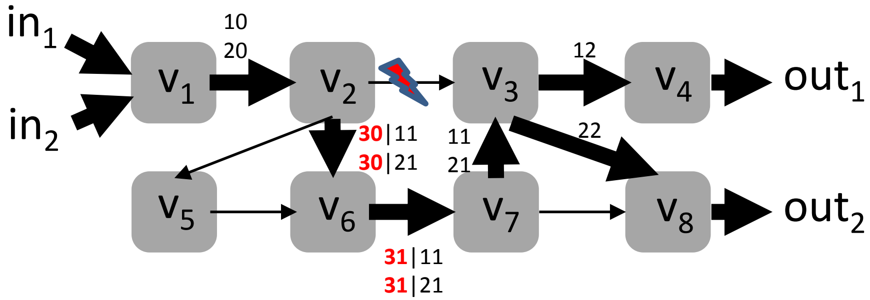
Fast Reroute Around *1 Failure*

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



Default routing of two flows

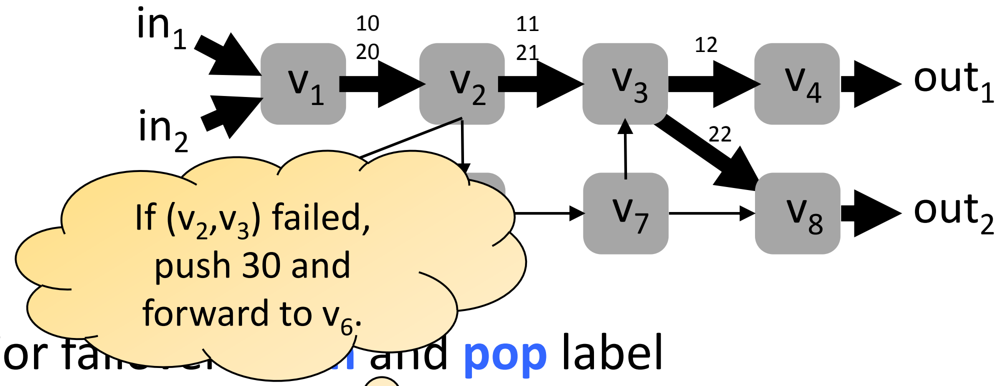
- For failover: **push** and **pop** label



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

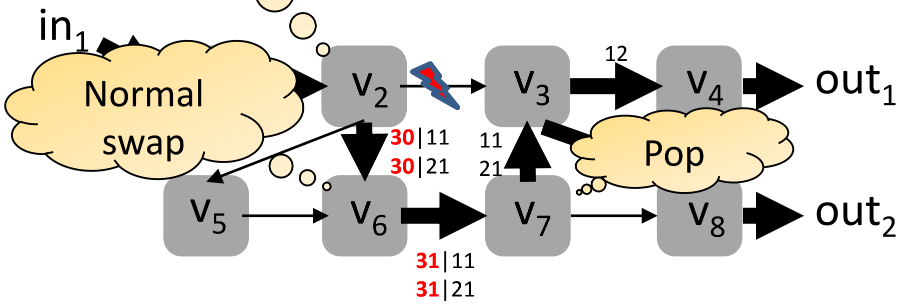
Fast Reroute Around *1 Failure*

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



Default routing of two flows

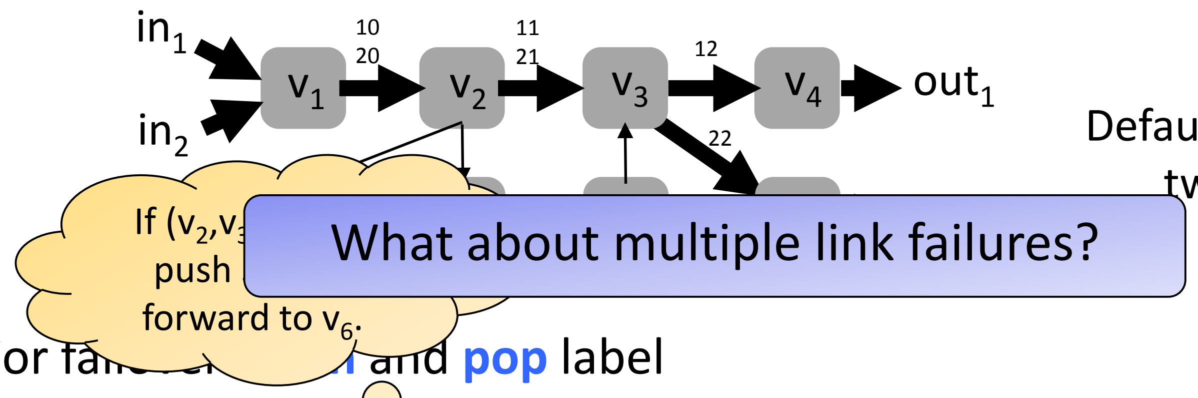
- For fa… and **pop** label



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

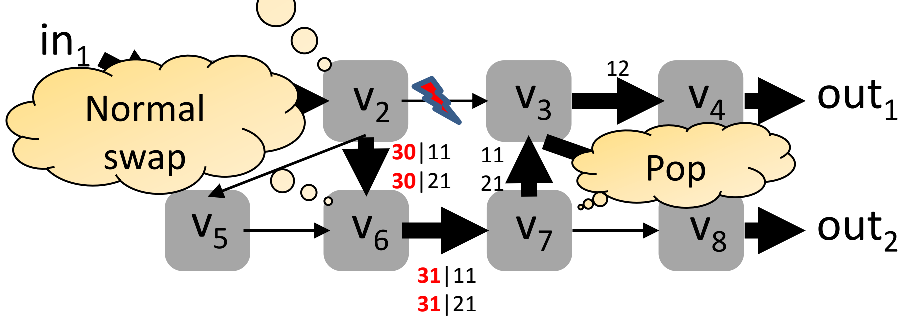
Fast Reroute Around *1 Failure*

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



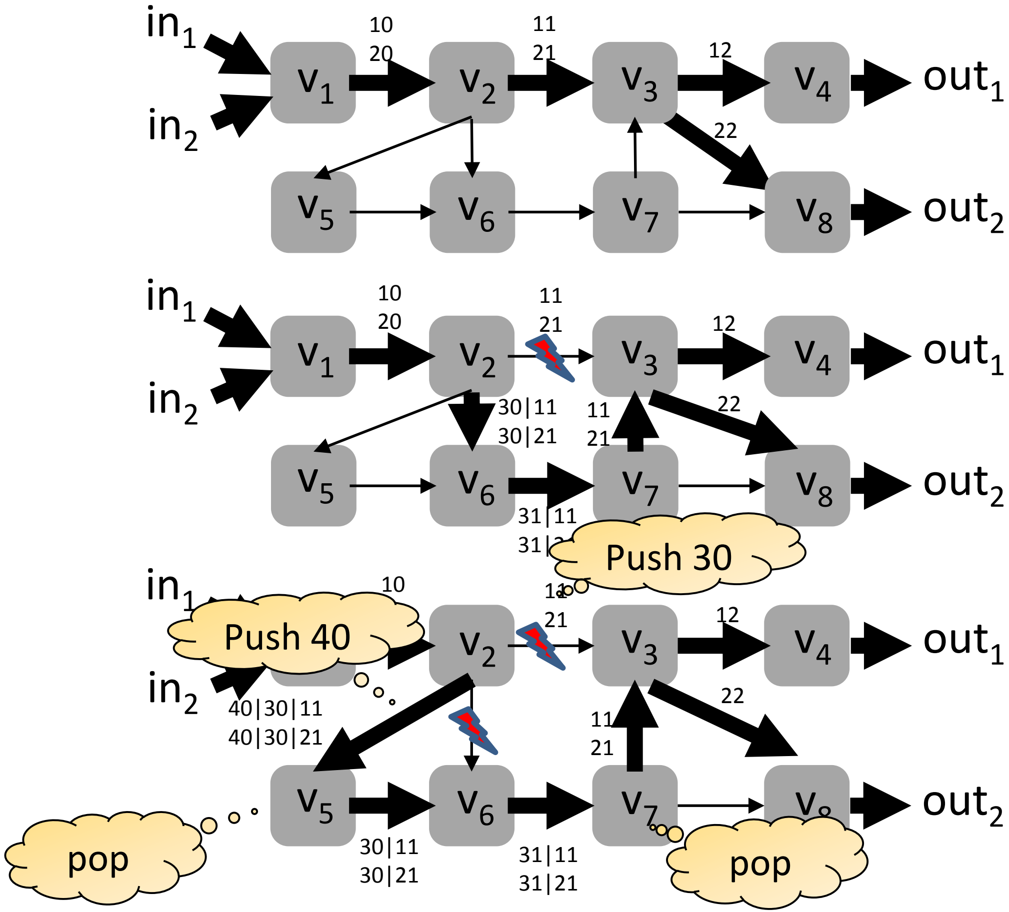
Default routing of two flows

- For fa… and **pop** label



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

2 Failures: Push *Recursively*



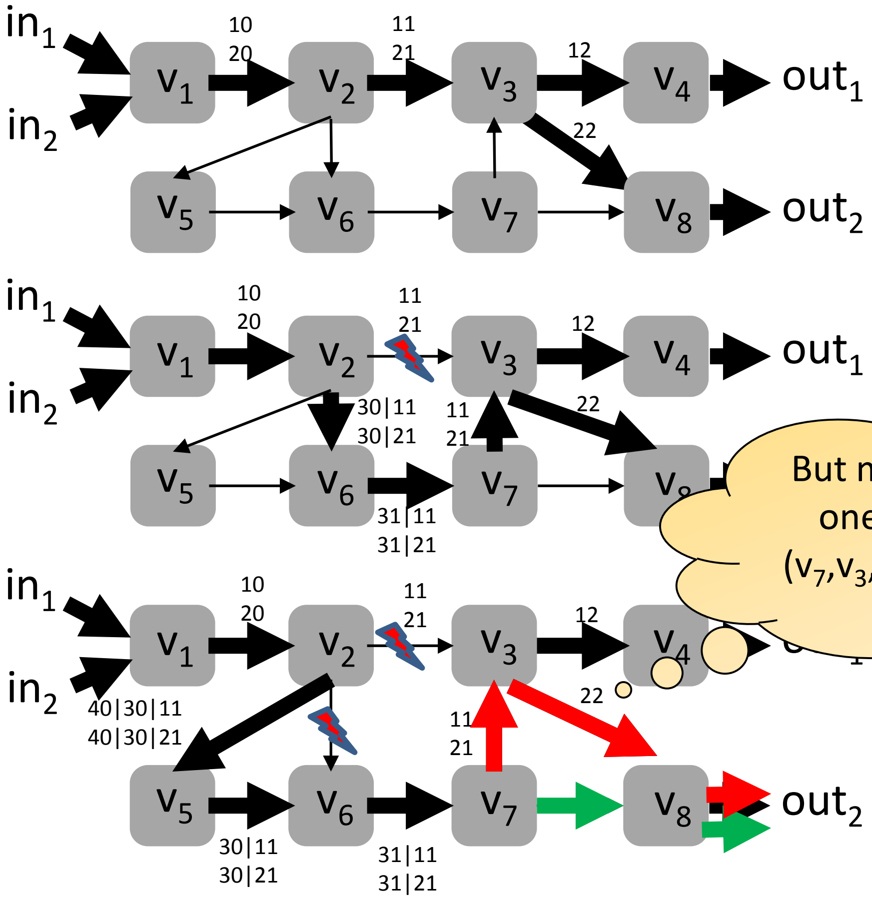
Original Routing

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

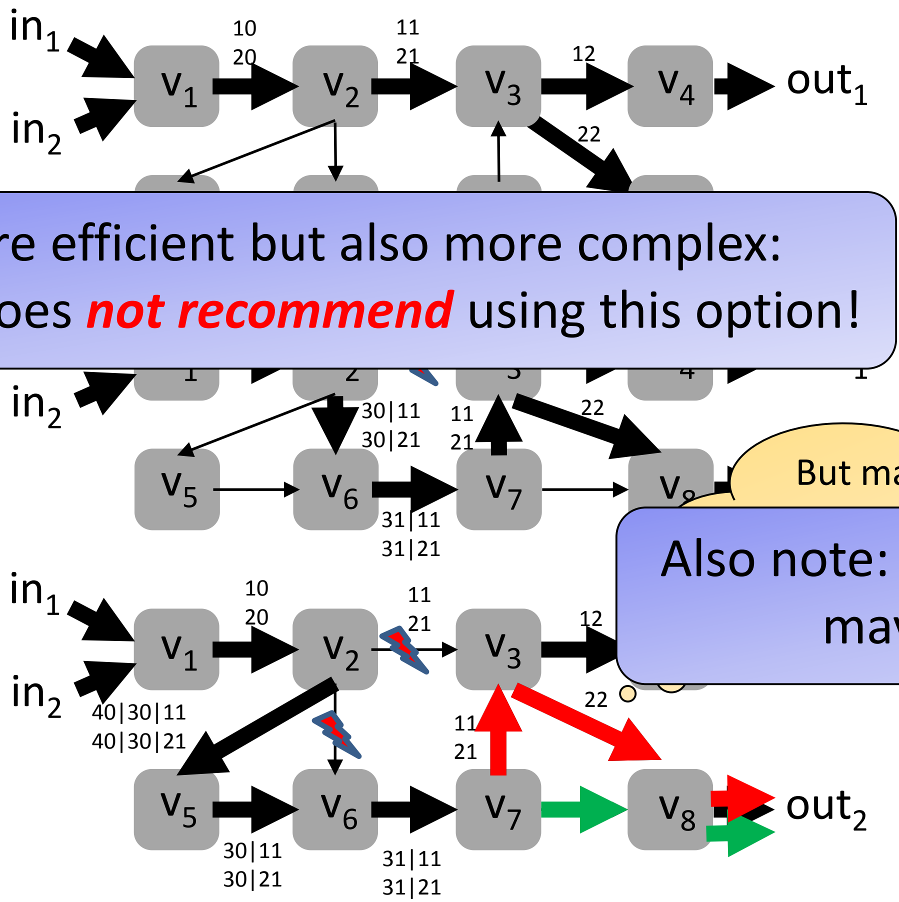
One failure: push 30: …)

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

… 30: route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

2 Failures: Push *Recursively*



Original Routing

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

One failure: push 30:

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

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)	[illegible]
	in_2	$\perp$	(v_1, v_2)	[illegible]
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	[illegible]
	(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}	[illegible]	40	[illegible]	pop
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	$swap(62)$
	(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

Version which does not mask links individually!

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)	$push(40)$

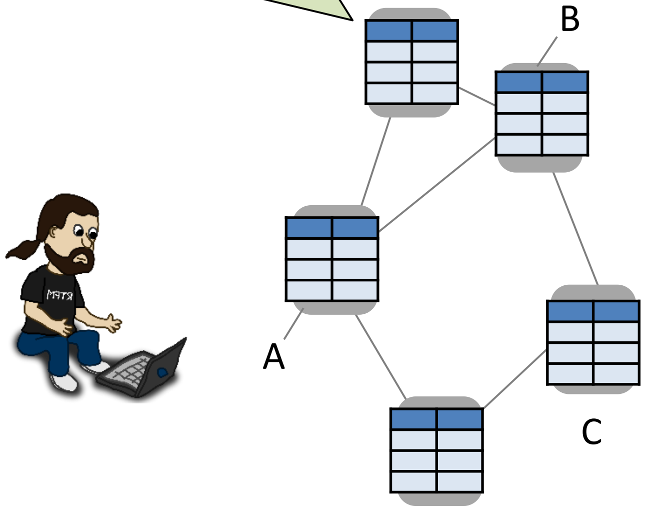
global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

Failover Tables

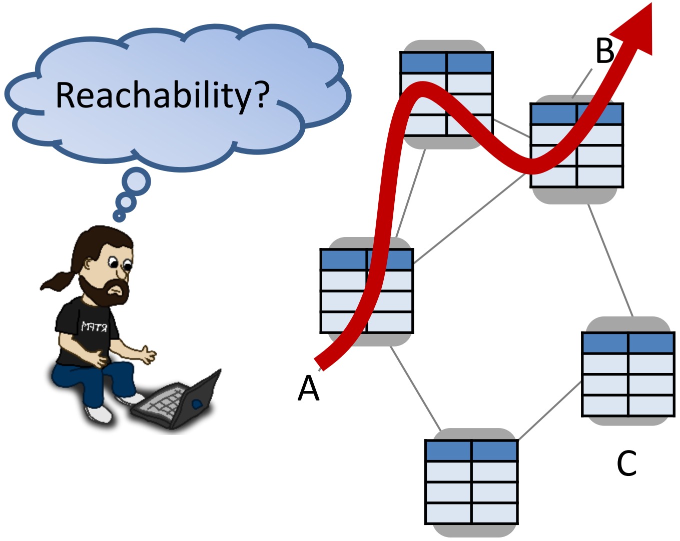
MPLS Tunnels in Today‘s ISP Networks

Responsibilities of a Sysadmin

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



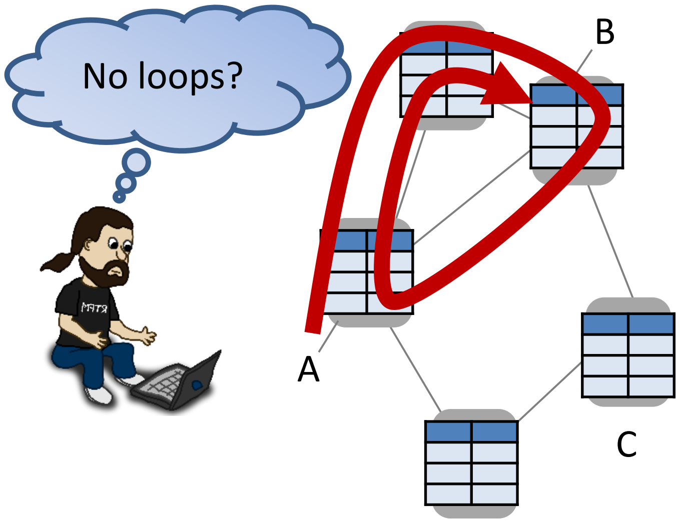
Responsibilities of a Sysadmin



Sysadmin responsible for:

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

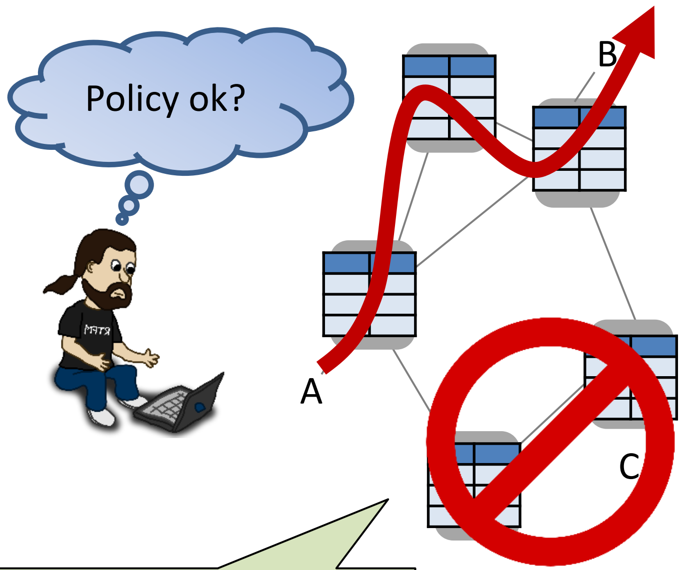
Responsibilities of a Sysadmin



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

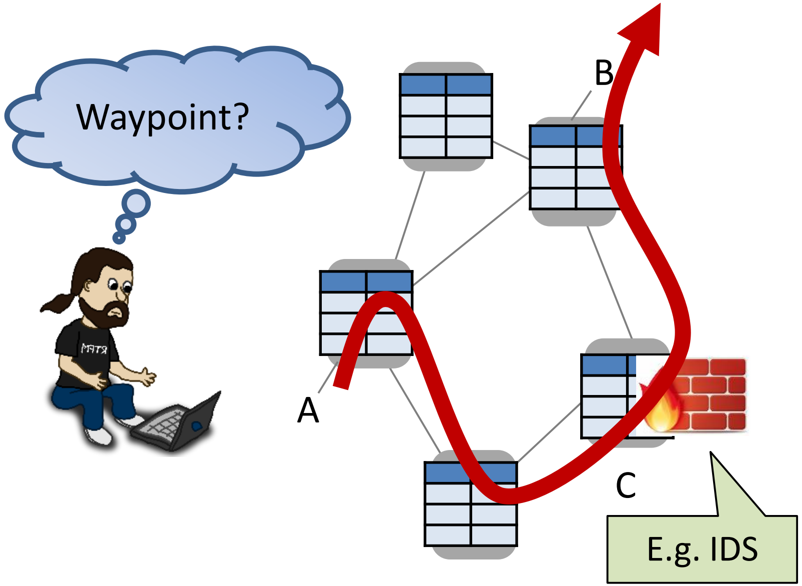


E.g. ***NORDUnet***: no traffic via Iceland (expensive!).

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?

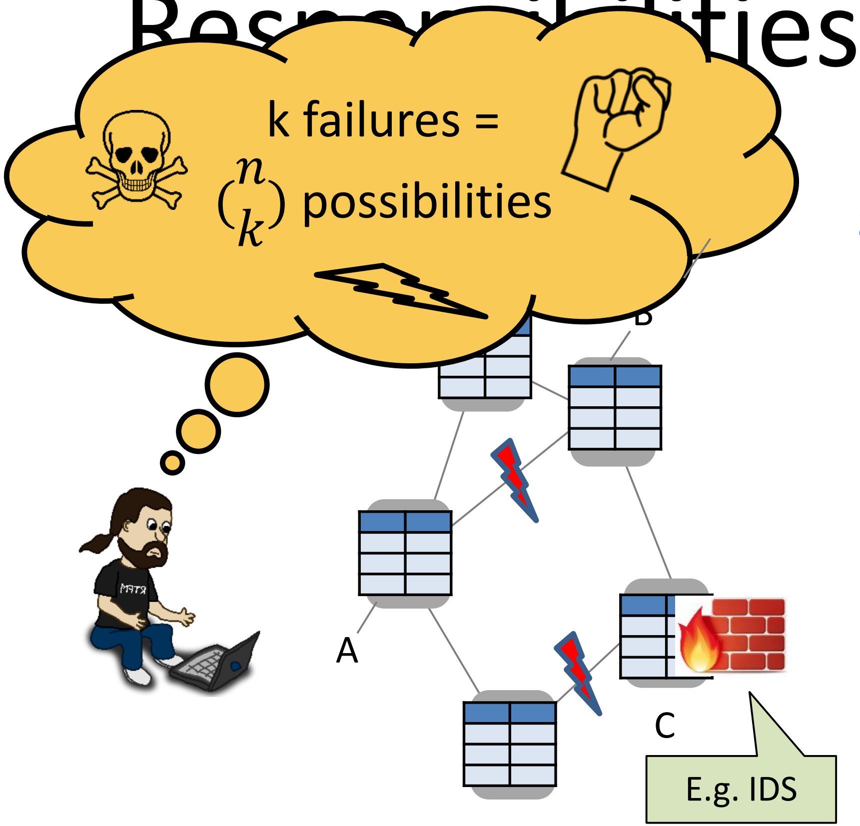
Responsibilities of a Sysadmin



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?!

Can we automate such tests
or even self-repair?

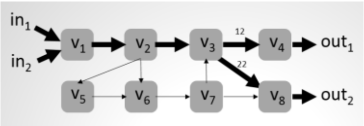
Can we automate such tests or even self-repair?

Yes! Automated **What-if Analysis Tool** for MPLS and SR in ***polynomial time***.

(INFOCOM 2018, CoNEXT 2018)

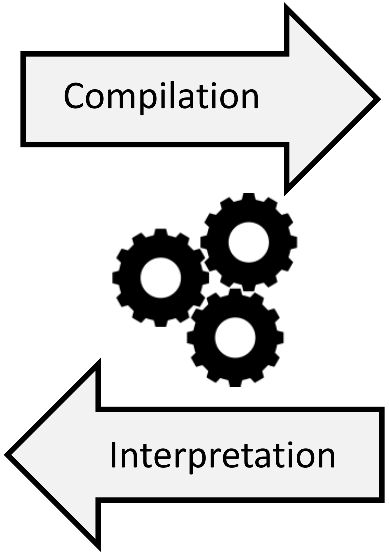
Leveraging Automata-Theoretic Approach

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\bot$	(v_1, v_2)	$push(10)$
	in_2	$\bot$	(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)	$push(40)$

MPLS **configurations**, Segment Routing etc.



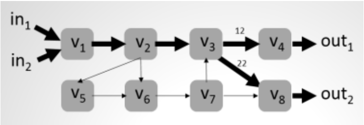
$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

Pushdown Automaton and **Prefix Rewriting Systems** Theory

Leveraging Autom… oach

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

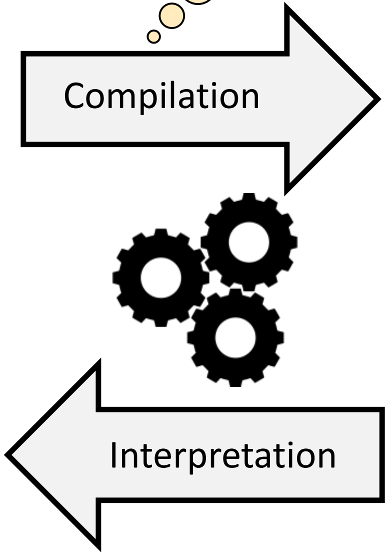
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$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

MPLS **configurations**, Segment Routing etc.



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

Pushdown Automaton and **Prefix Rewriting Systems** Theory

Mini-Tutorial: A Network Model

- Network: a 7-tuple

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

Mini-Tutorial: A Network Model

- Network: a 7-tuple

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

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 $(v, \lambda_v(out)) \in E$

Mini-Tutorial: A Network Model

- Network: a 7-tuple

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

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 **sequence of tuples**:

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

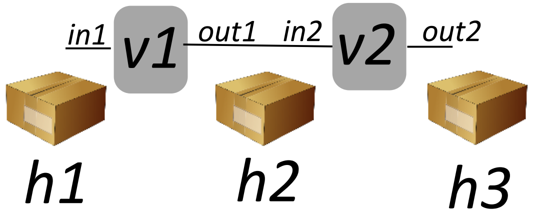
- v_i: Node receives...
- in_i: ... on interface...
- h_i: ... packet with header...
- out_i: ... forwards it to live next hop...
- h_{i+1}: ... with new header..
- F_i: ... given that these links are down.

- Example: **routing** (in)finite sequence of tuples

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

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

$$\ldots$$



Example Rules:
Regular Forwarding on Top-Most Label

Push:

Push label on stack

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

Swap:

Swap top of stack

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

Pop:

Pop top of stack

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

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:

$$(v_1, in_1)h_1\bot \rightarrow (v_1, out, 0)h\bot \rightarrow (v_1, out', 1)h'\bot \rightarrow (v_1, out'', 2)h''\bot \rightarrow \ldots \rightarrow (v_1, out_1, i)h_2\bot$$

Try default

Try first backup

Try second backup

A Complex and Big Formal Language!
Why Polynomial Time?!

- 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**!

A Complex and Big Formal Language! Why Polynomial Time?!

- 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**!

This is ***not*** how we will use the PDA!

A Complex and Big Formal Language! Why Polynomial Time?!

- 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!
(*Or:* Swiss to the Rescue!)

- 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

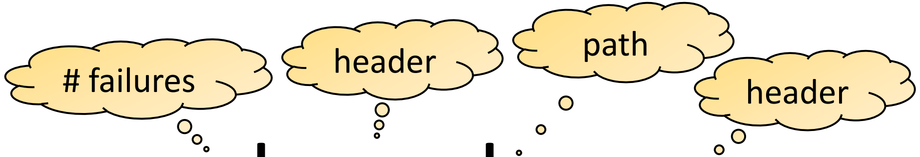
Tool and Query Language

<u>Part 1:</u> Parses query and constructs Push-Down System (PDS)

- In Python 3

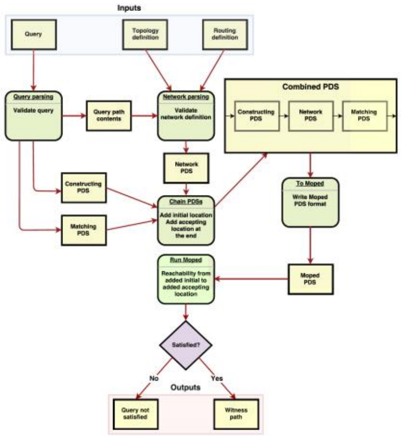
<u>Part 2:</u> Reachability analysis of constructed PDS

- Using ***Moped*** tool



k <a> b <c>

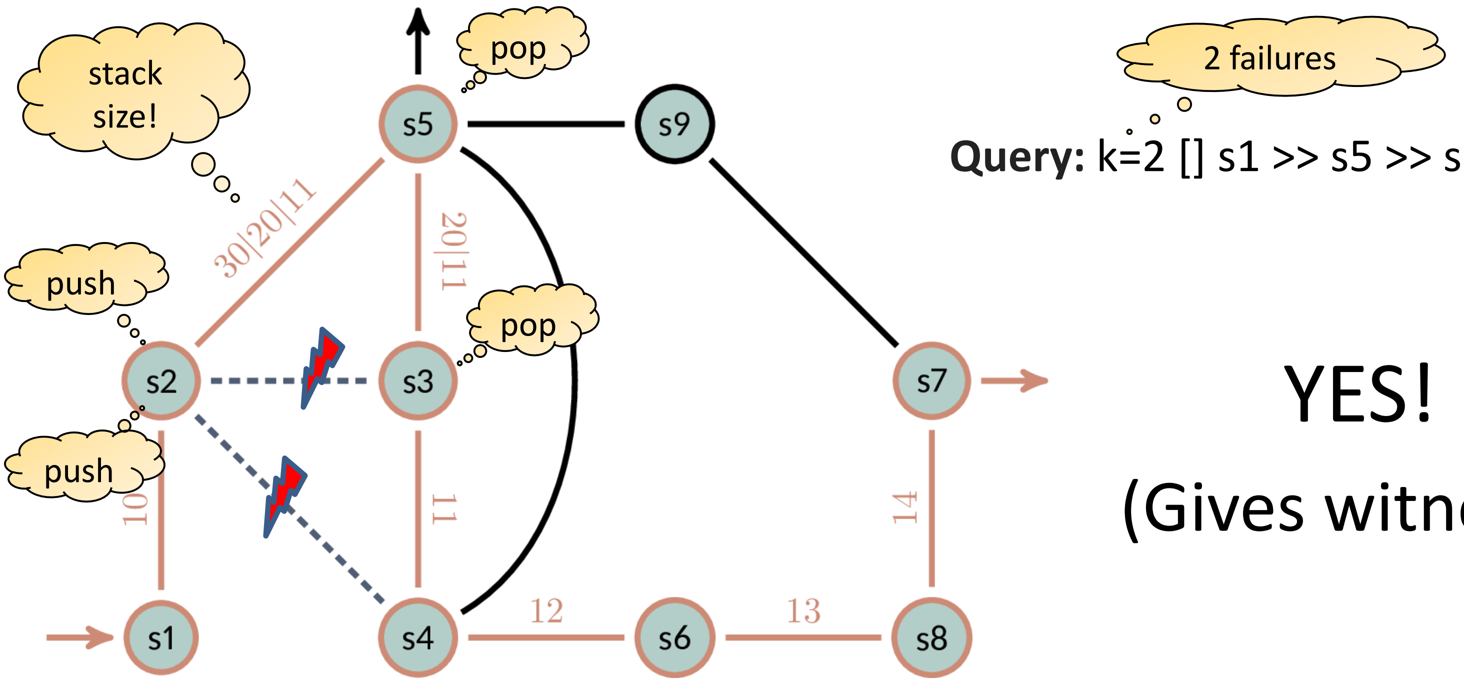
Regular query language



query processing flow

Example: Traversal Testing With 2 Failures

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



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

YES!

(Gives witness!)

Case Study: NORDUnet (thanks, Henrik ☺)

```
show route forwarding-table family mpls extensive
| display .ml
show isis adjacency detail | display .ml
```

- Small but complex network
  - 24 MPLS routers
- Between 20-90min, ca. ~GB memory

Related Work

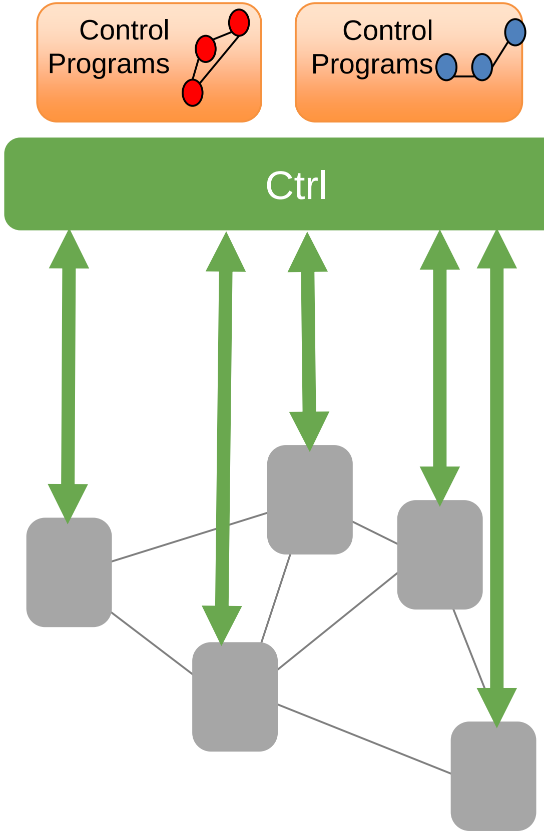
	P-Rex	NetKAT	HSA	VeriFlow	Anteater
Protocol Support	SR/MPLS	OF	Agn.	OF	Agn.
Approach	Autom.	Alg.	Geom.	Tries	SAT
Complexity	Polynom.	PSPACE	Polynom.	NP	NP
Static	✓	✓	✓	χ	✓
Reachability	✓	✓	✓	✓	✓
Loop Queries	✓	✓	✓	✓	✓
What-if	✓	N/A	✓	N/A	χ
Unlim. Header	✓	N/A	χ	χ	N/A
Performance	✓	✓ [1]	✓	✓	✓
Waypointing	✓	✓	✓	✓	χ
Language	Py., C	OCaml	Py., C	Py.	C++, Ruby

Roadmap

- 1st Use Case for Automation: What-if Analysis ✓
- 2nd Use Case for Automation: Consistent Rerouting

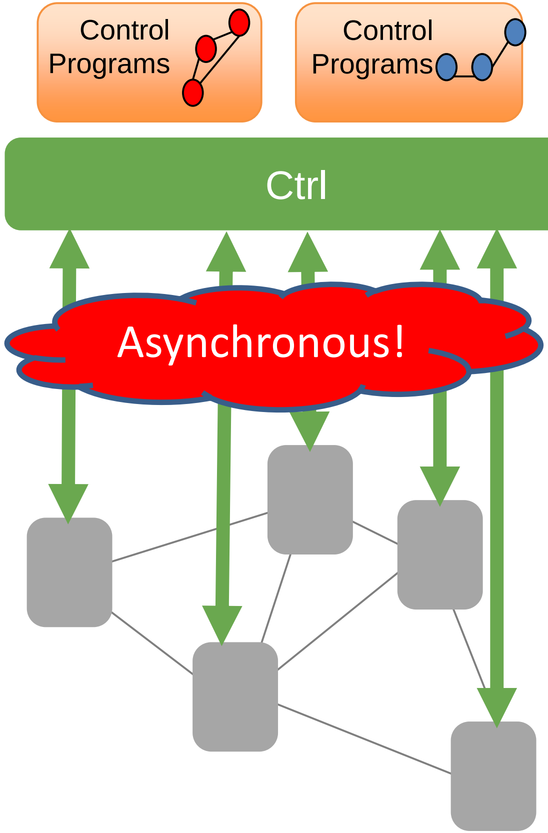
Consistent Rerouting

Challenge: Decoupling



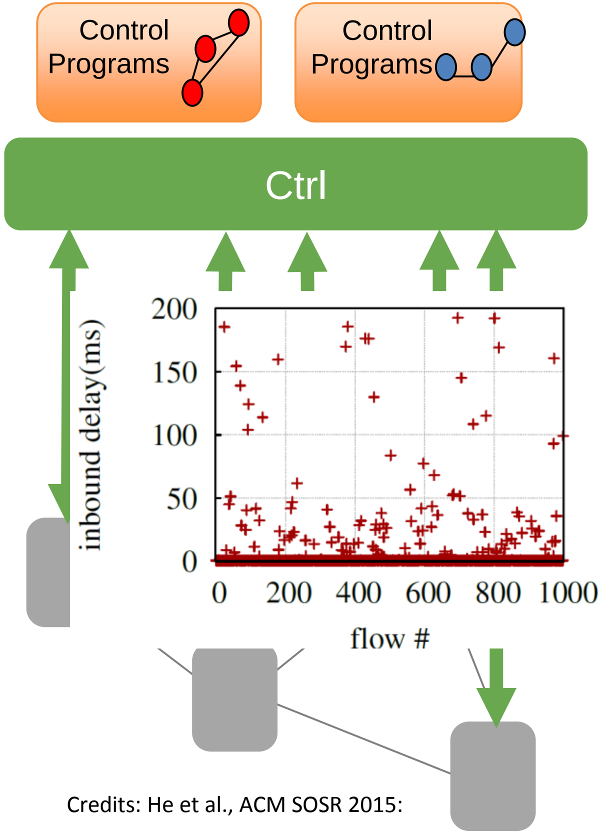
Consistent Rerouting

Challenge: Decoupling



Consistent Rerouting

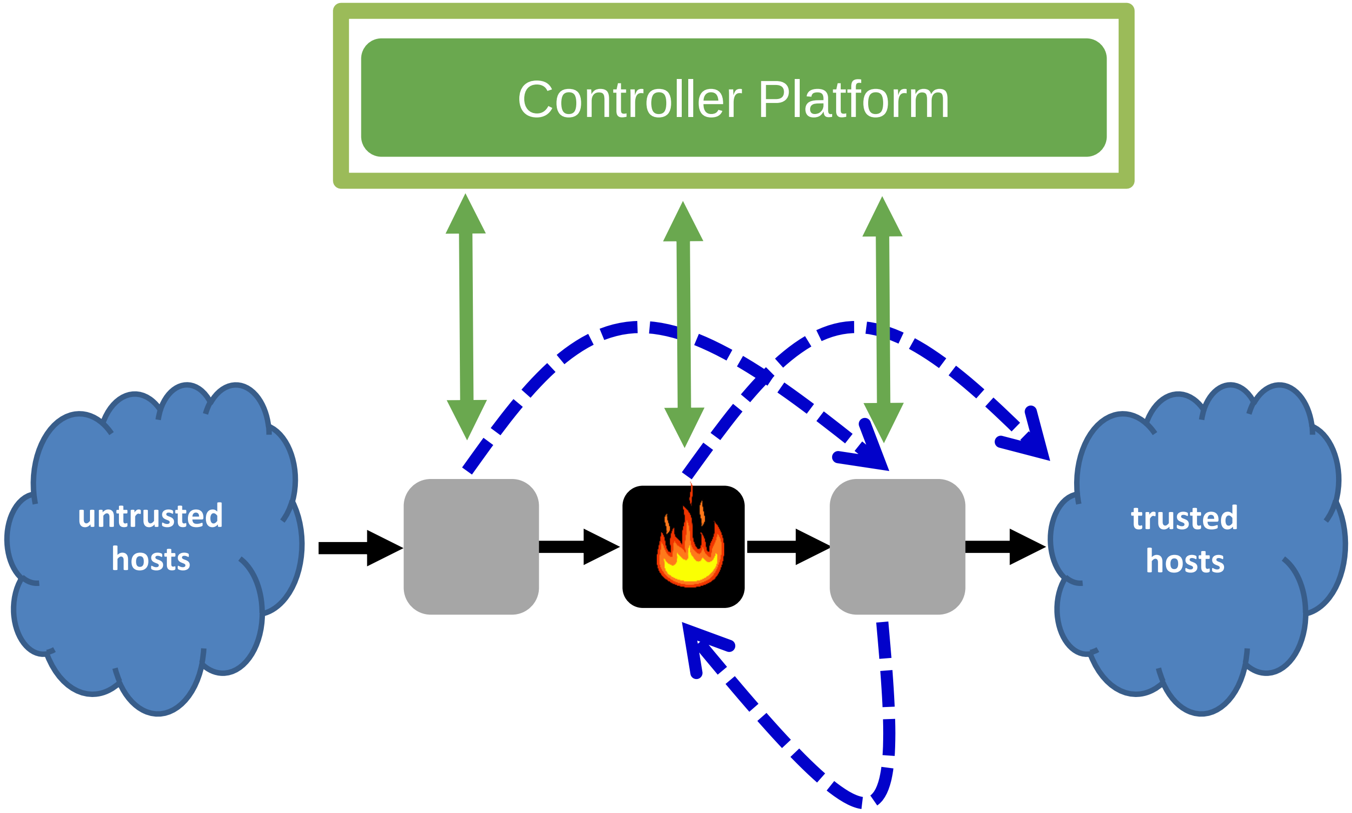
Challenge: Decoupling



Credits: He et al., ACM SOSR 2015: without network latency

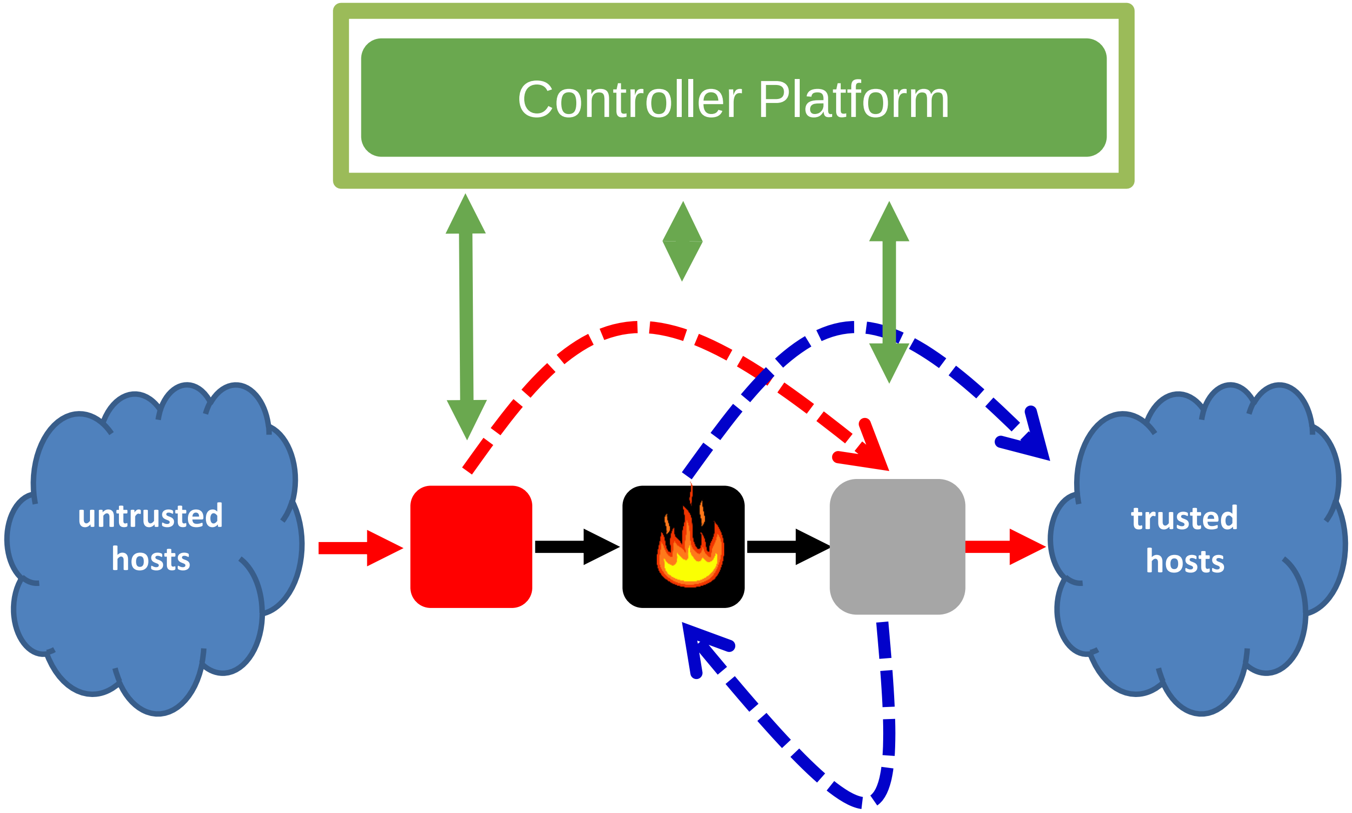
Despite centralization: SDN stays a distributed system!

What could go wrong…?



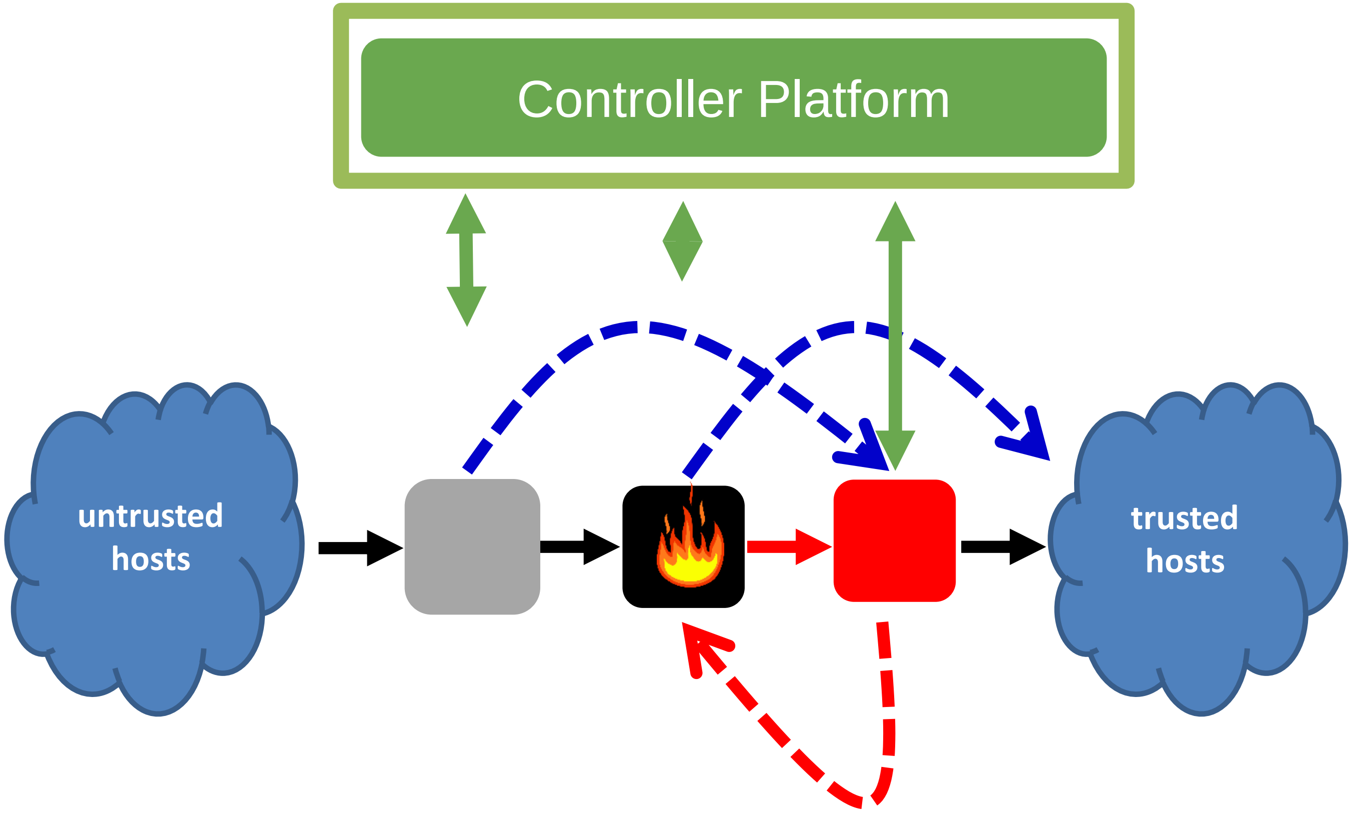
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Problem 1: Bypassed Waypoint



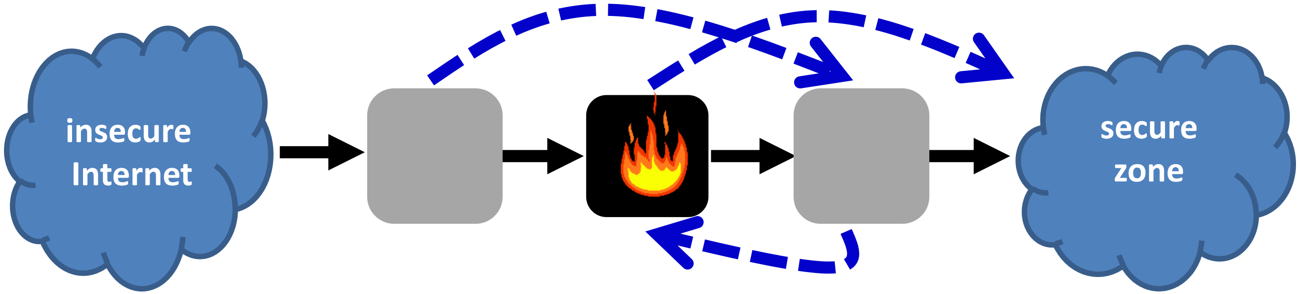
Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

Problem 2: Transient Loop

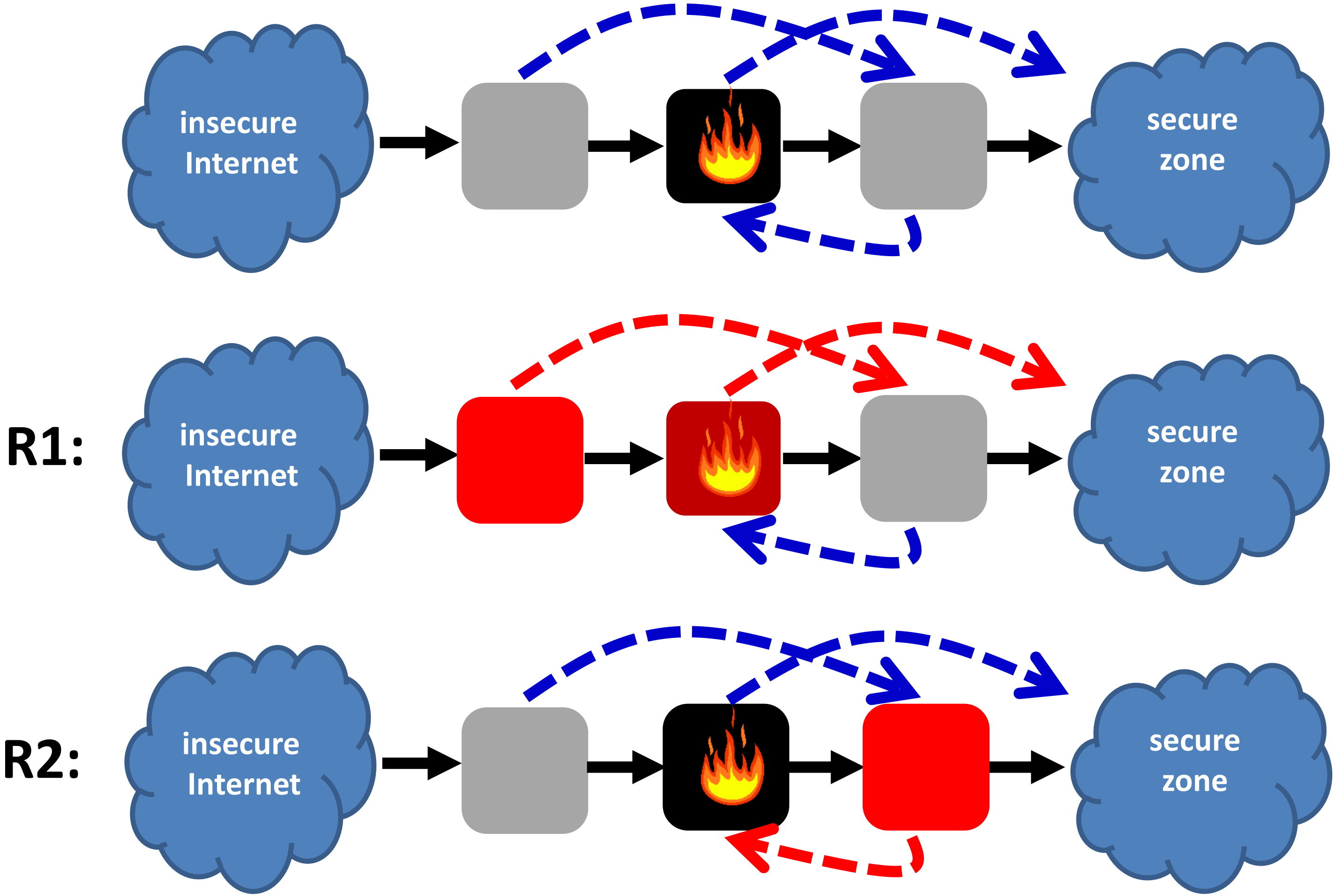


Invariant: Traffic from untrusted hosts to trusted hosts via firewall!

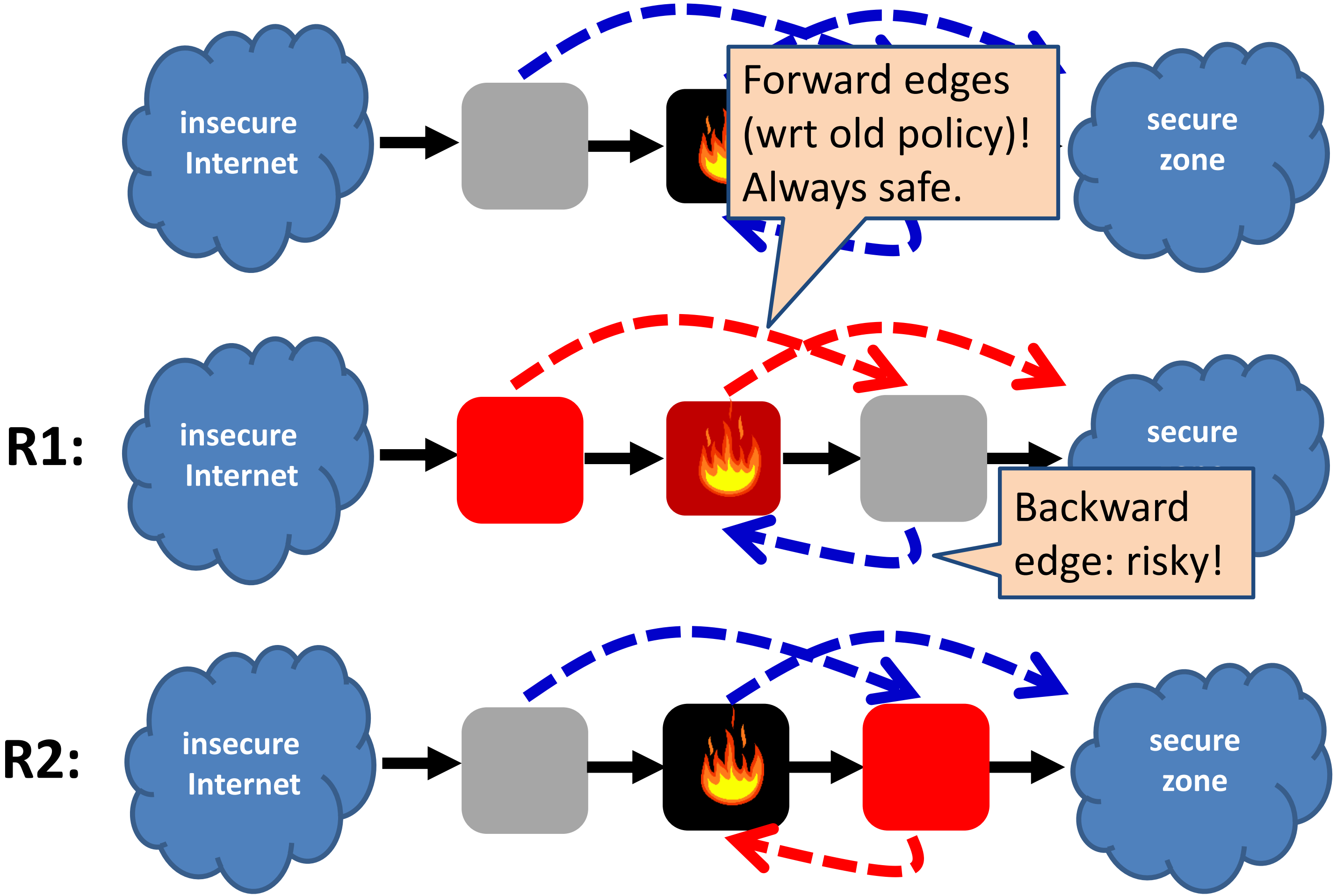
Loop-Free Update Schedule



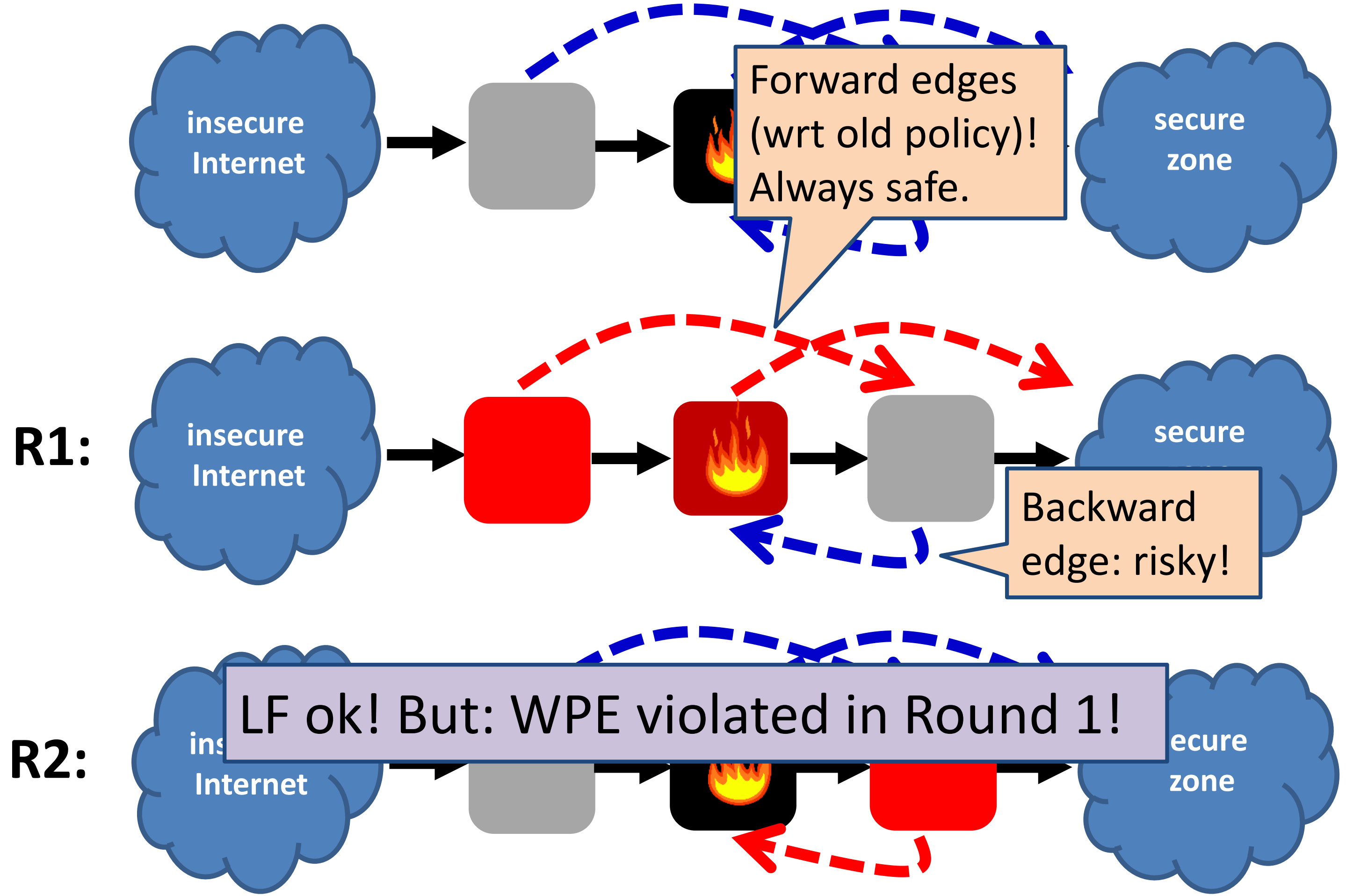
Loop-Free Update Schedule



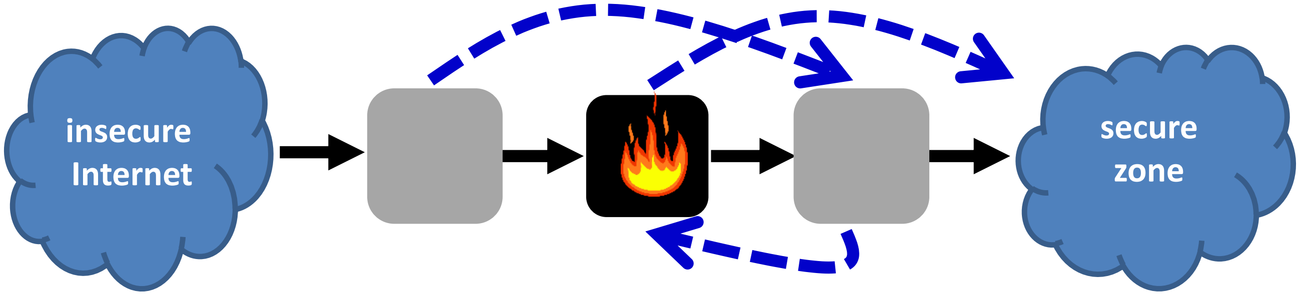
Loop-Free Update Schedule



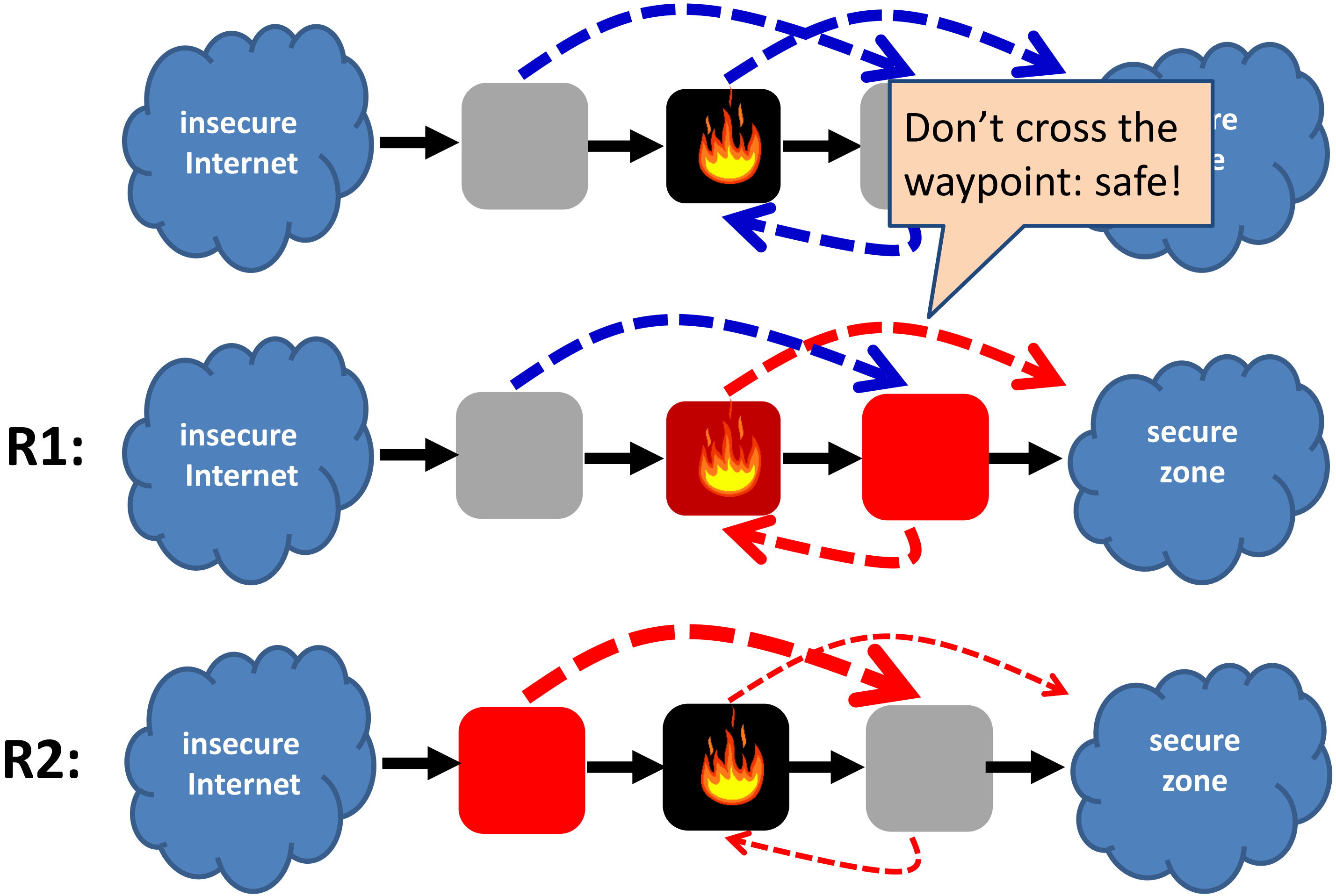
Loop-Free Update Schedule



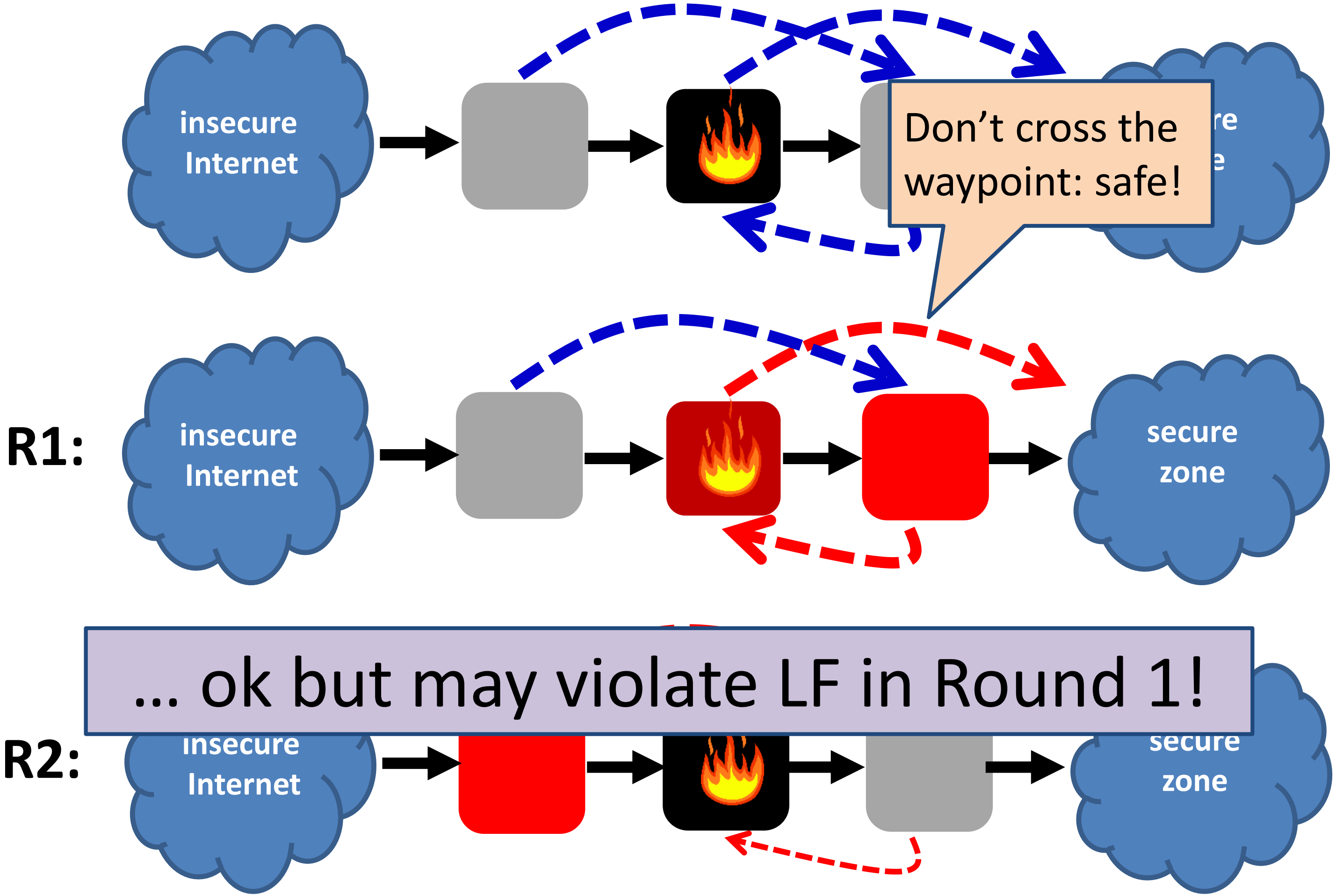
Waypoint Respecting Schedule



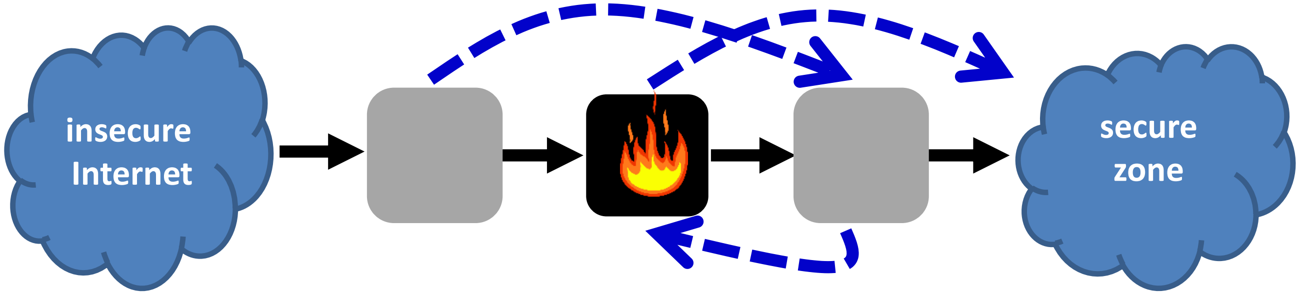
Waypoint Respecting Schedule



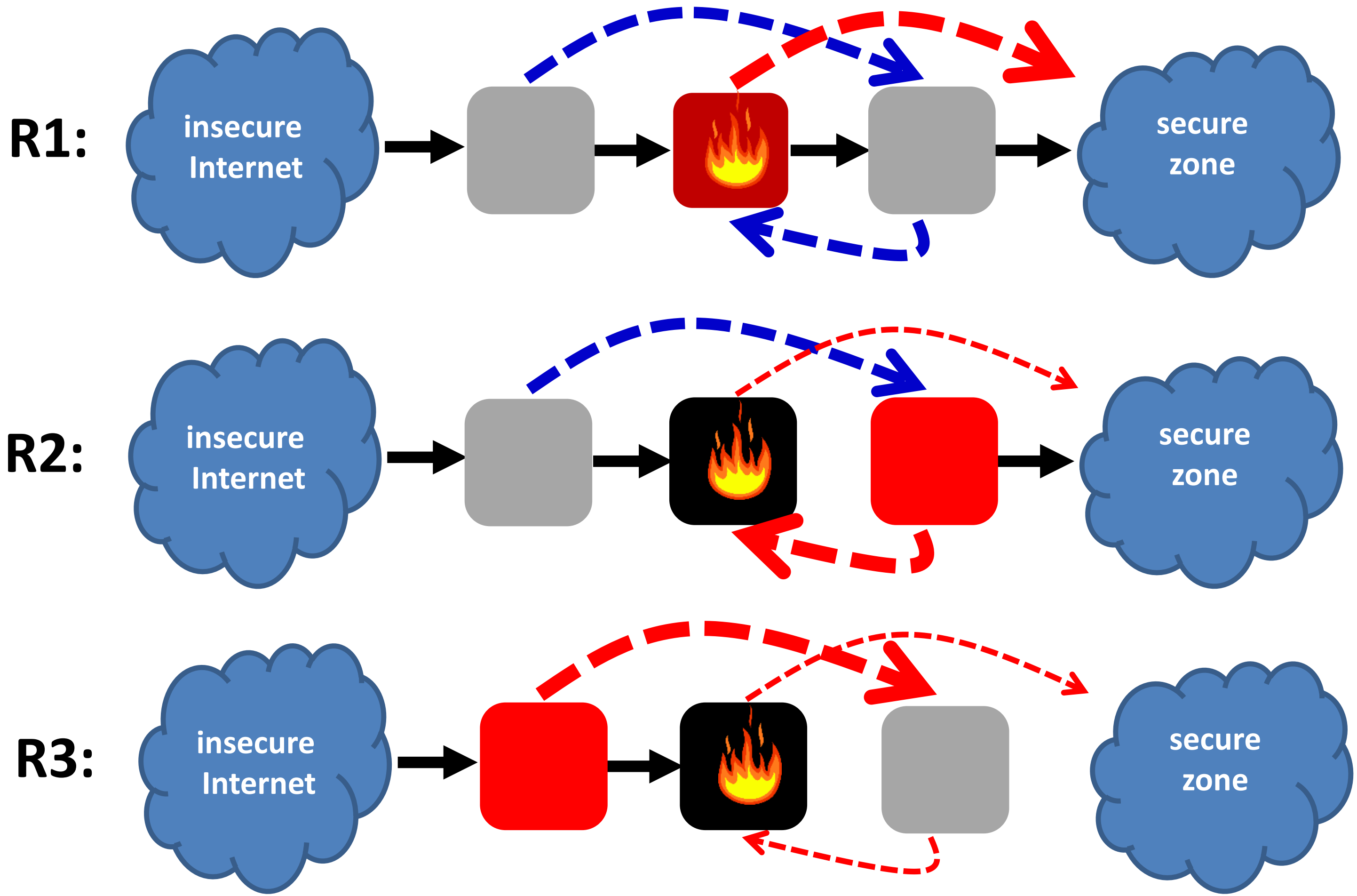
Waypoint Respecting Schedule



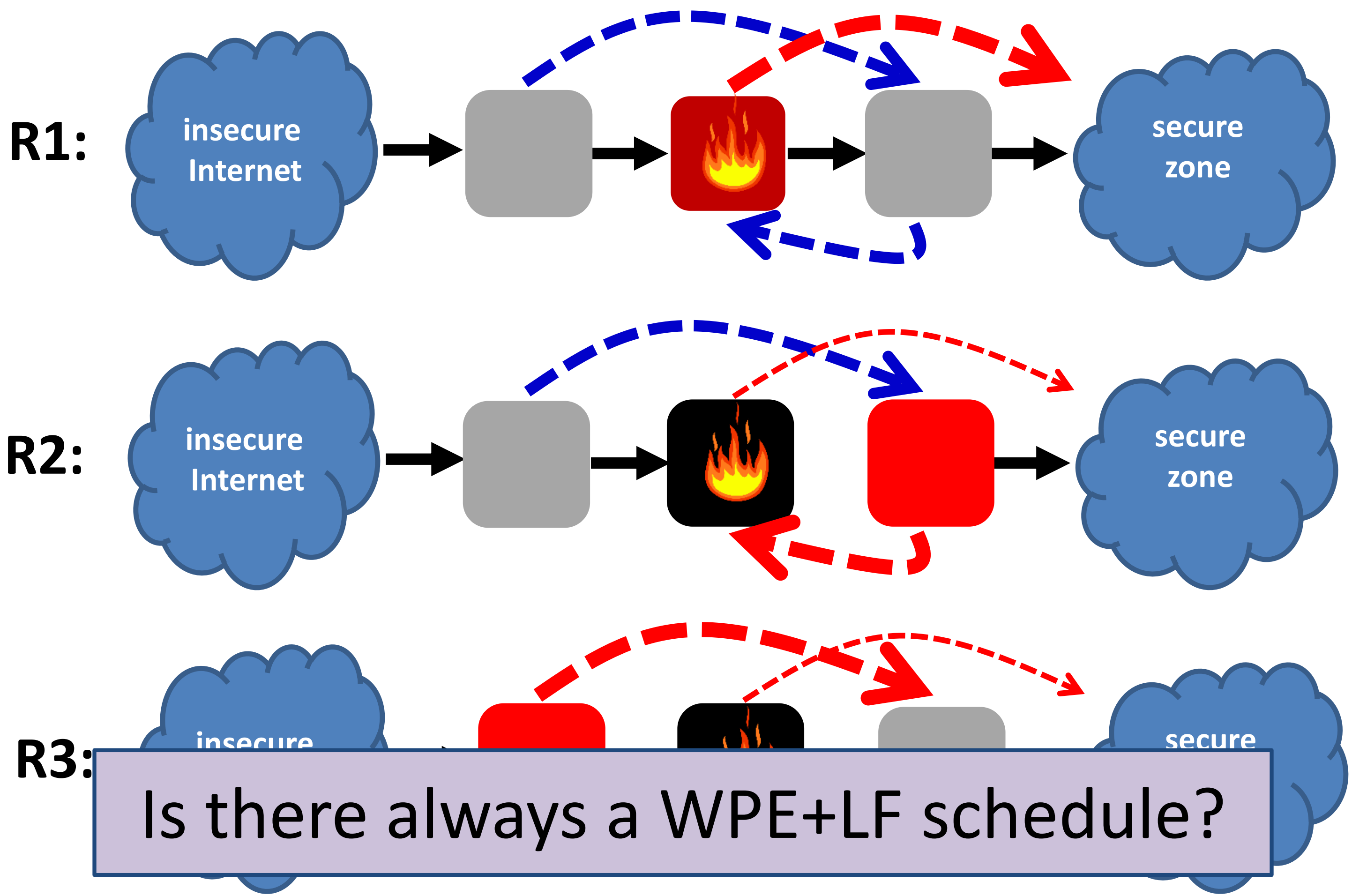
Can we have both LF and WPE?



Yes: but it takes 3 rounds!

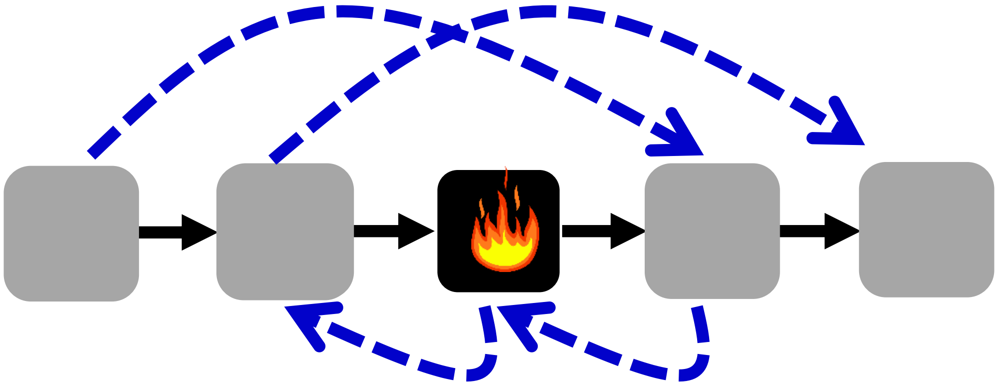


Yes: but it takes 3 rounds!

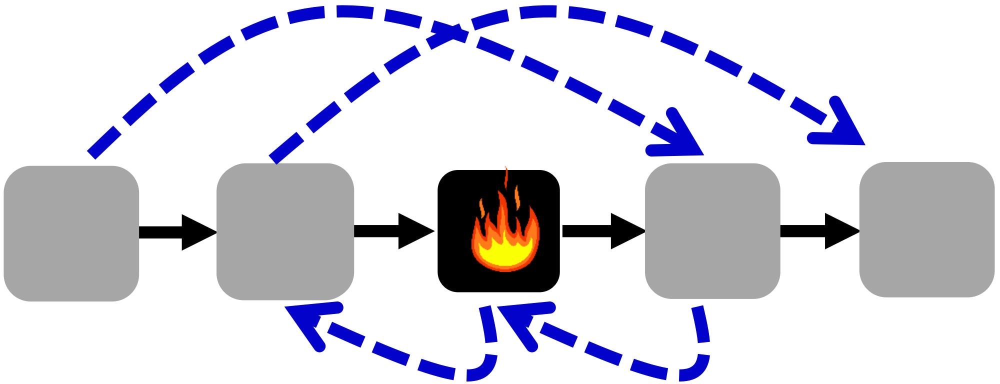


Is there always a WPE+LF schedule?

What about this one?



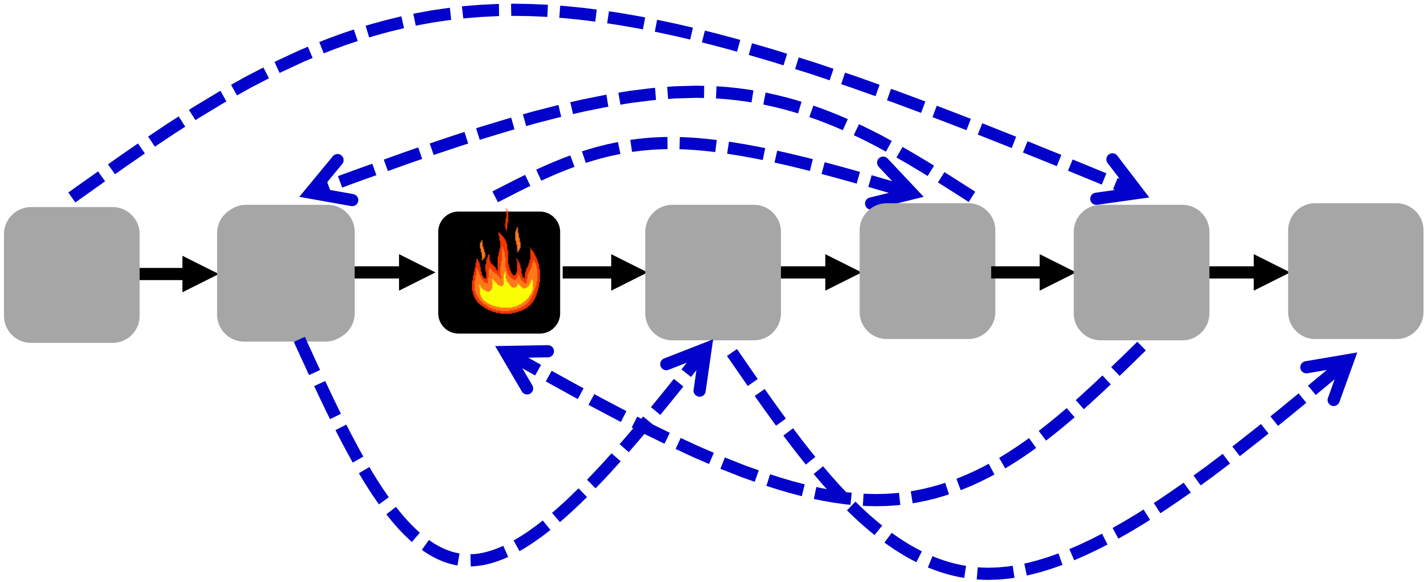
LF and WPE may conflict!



- ❑ Cannot update any **forward edge** in R1: WP
- ❑ Cannot update any **backward edge** in R1: LF

No schedule exists!

What about this one?



Complexity overview for LF & WPE

- LF & WPE may conflict
  - Deciding: NP-complete

- LF: Always possible in *n* rounds (*relaxed* version: O(log *n*) rounds)
  - Fastest schedule: NP-complete
  - Approximations? ***Unknown***

- LF: Maximizing simultaneous updates: NP-complete
  - Can be approximated well (Feedback Arc Set / Max. Acyc. Subg.)
  - But: Can turn O(1) instances into Ω(*n*) schedules

Complexity overview for LF & WPE

- LF & WPE may conflict
  - Deciding: NP-complete

- LF: Always possible in *n* rounds (*relaxed* version: O(log *n*) rounds)
  - Fastest schedule: NP-complete
  - Approximations? ***Unknown***

- LF: Maximizing simultaneous updates: NP-complete
  - Can be approximated well (Feedback Arc Set / Max. Acyc. Subg.)
  - But: Can turn O(1) instances into Ω(*n*) schedules

Some synthesis results already exist for LF & WPE
(McClurg et al. PLDI'15, Zhou et al. NSDI'15)

Complexity overview for LF & WPE

- LF & WPE may conflict
  - Deciding: NP-complete

- LF: Always possible in *n* rounds (*relaxed* version: O(log *n*) rounds)
  - Fastest schedule: NP-complete
  - Approximations? ***Unknown***

- LF: Maximizing simultaneous updates: NP-complete
  - Can be approximated well (Feedback Arc Set / Max. Acyc. Subg.)
  - But: Can turn O(1) instances into Ω(*n*) schedules

None for **congestion** (bandwith/capacities)

Some synthesis results already exist for LF & WPE
(McClurg et al. PLDI'15, Zhou et al. NSDI'15)

Complexity overview for LF & WPE

- LF & WPE may conflict
  - Deciding: NP-complete

- LF: Always possible in *n* rounds (*relaxed* version: O(log *n*) rounds)
  - Fastest schedule: NP-complete
  - Approximations? ***Unknown***

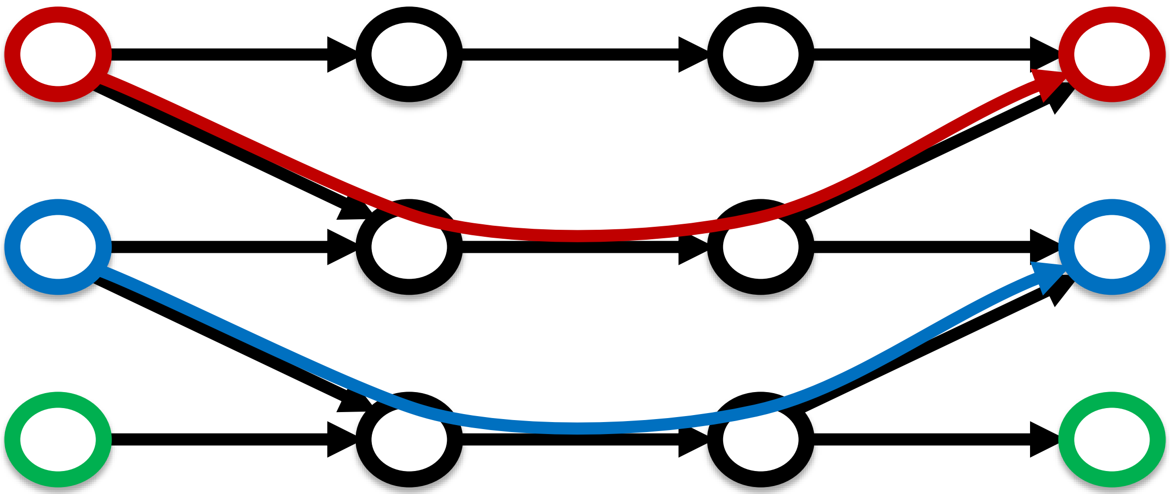
- LF: Maximizing simultaneous updates: NP-complete
  - Can be approximated well (Feedback Arc Set / Max. Acyc. Subg.)
  - But: Can turn O(1) instances into Ω(*n*) schedules

None for **congestion** (bandwith/capacities)

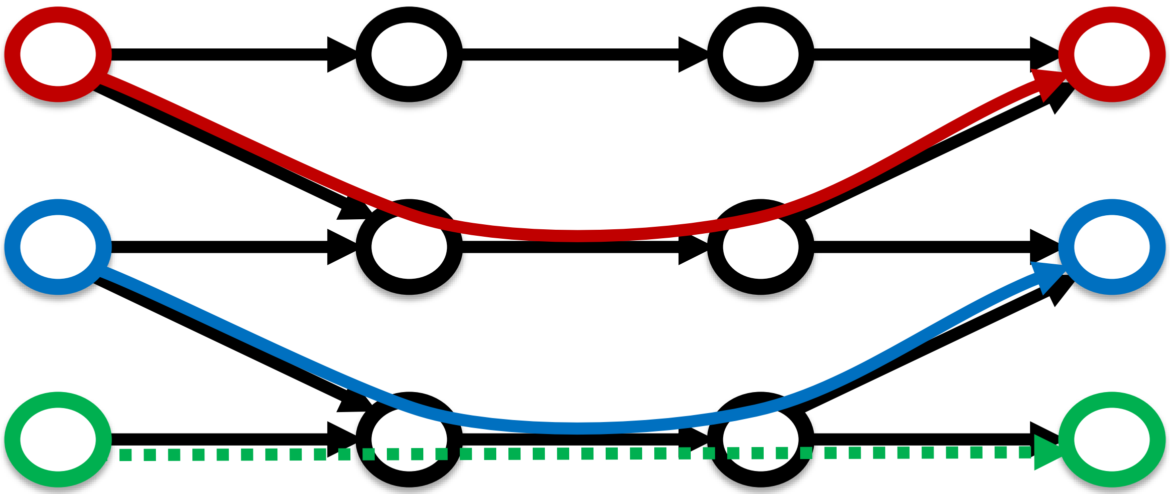
Some synthesis results already exist for LF & WPE
(McClurg et al. PLDI'15, Zhou et al. NSDI'15)

What about congestion?

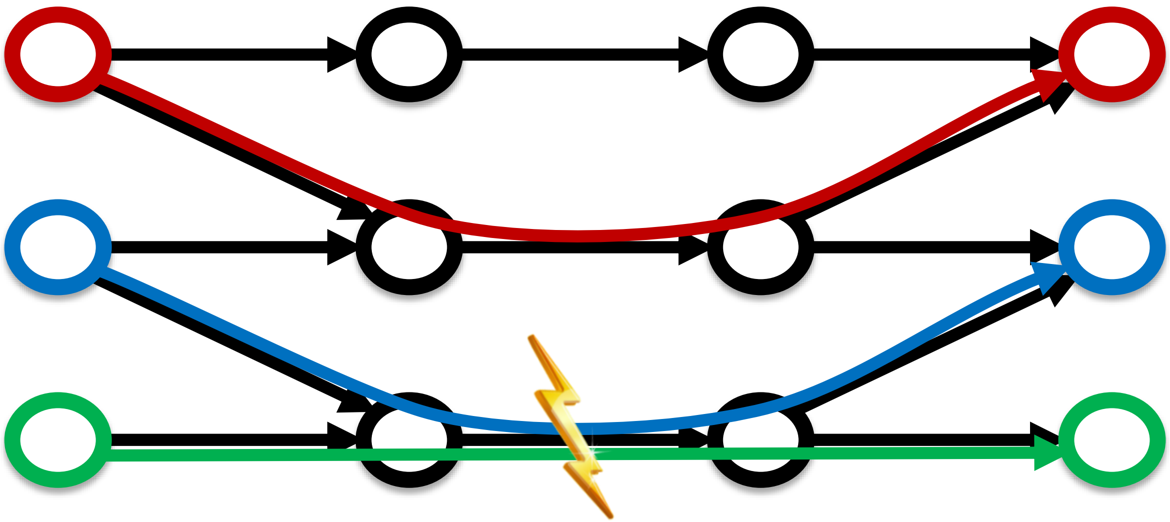
A Small Sample Network



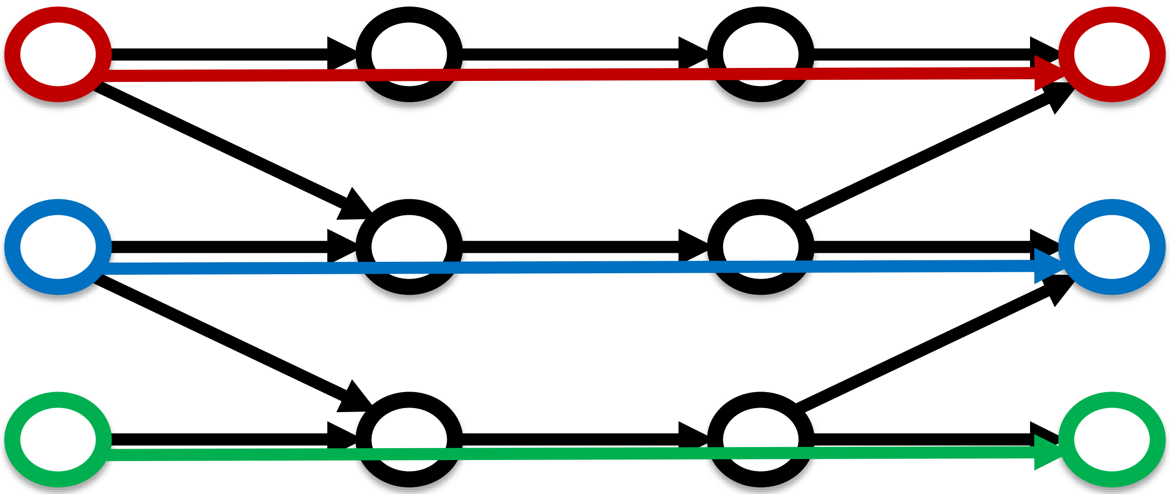
Green wants to send as well



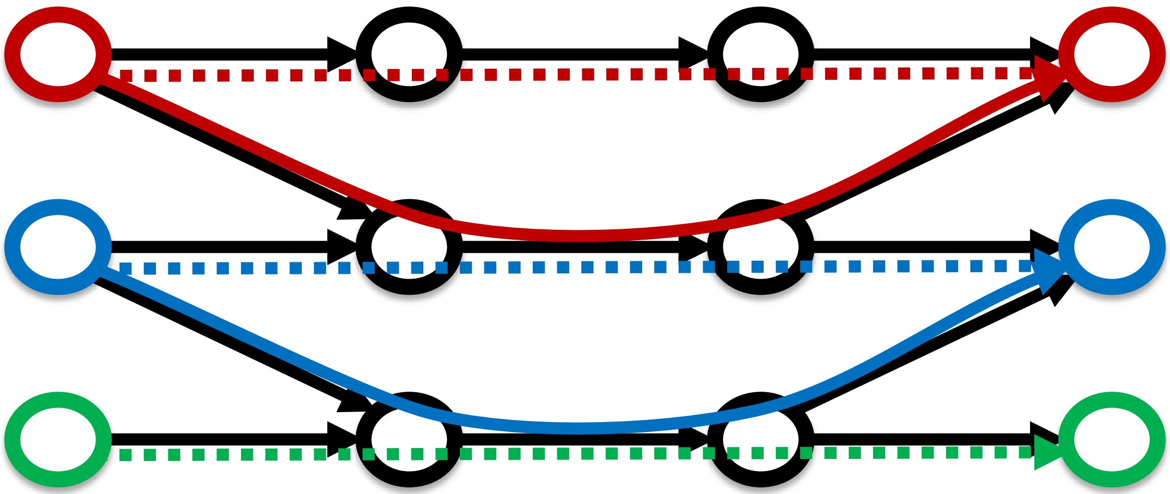
Congestion!



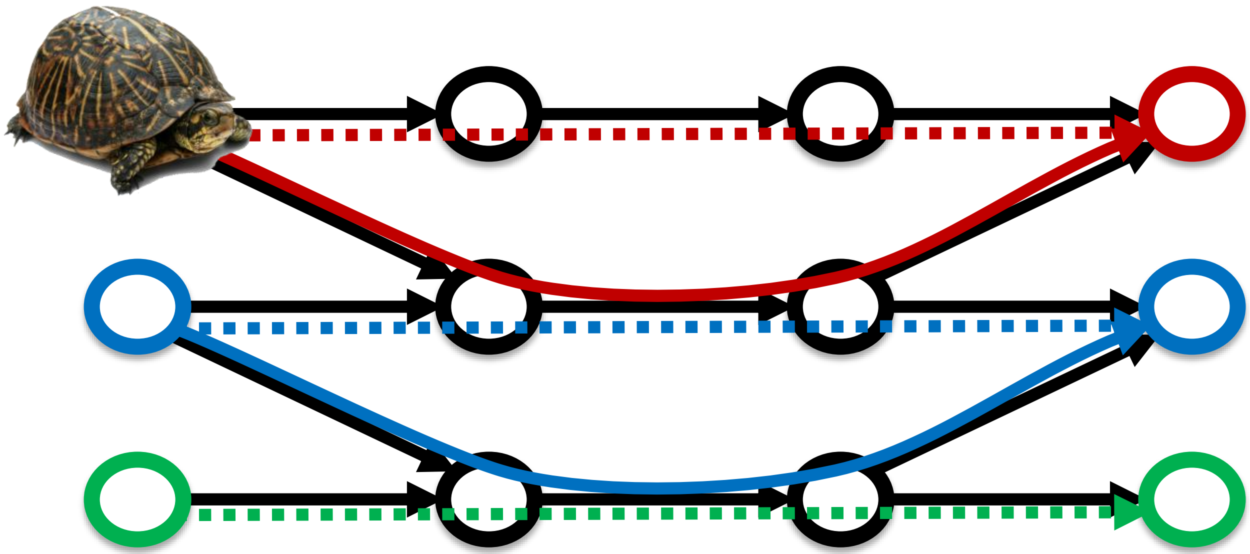
This would work



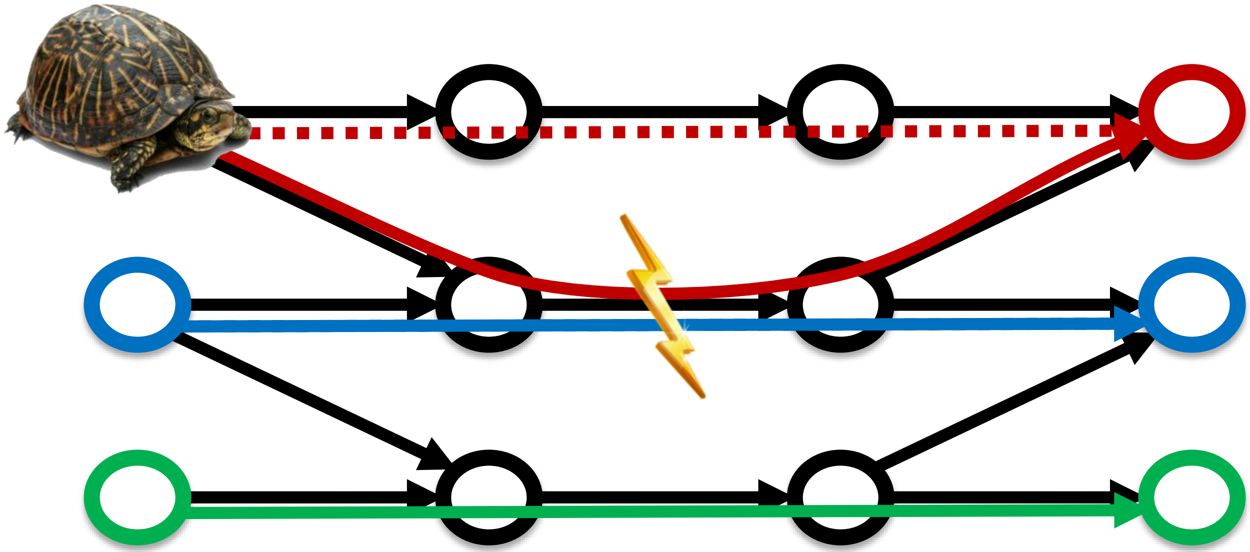
So lets go back



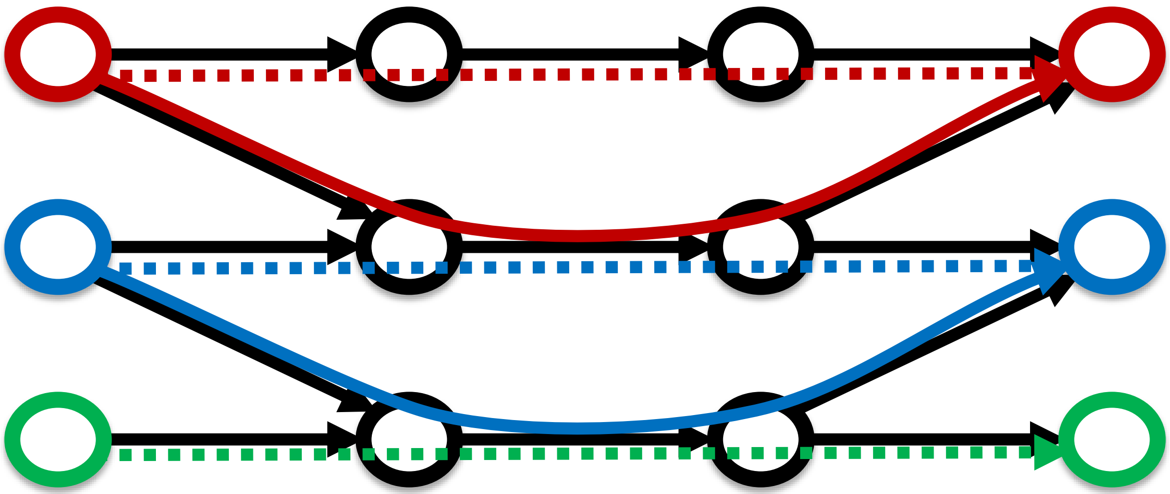
But Red is a bit Slow..



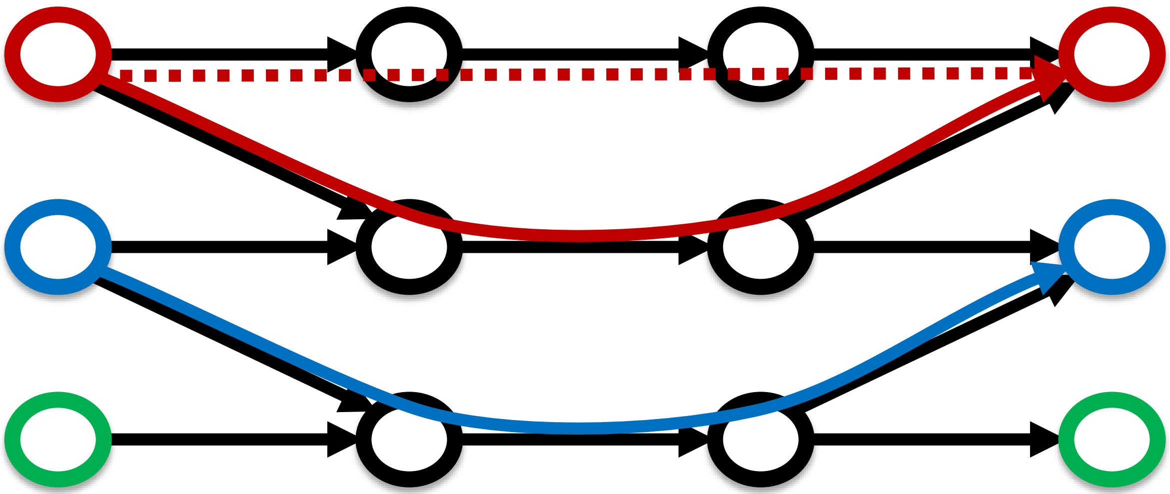
Congestion Again!



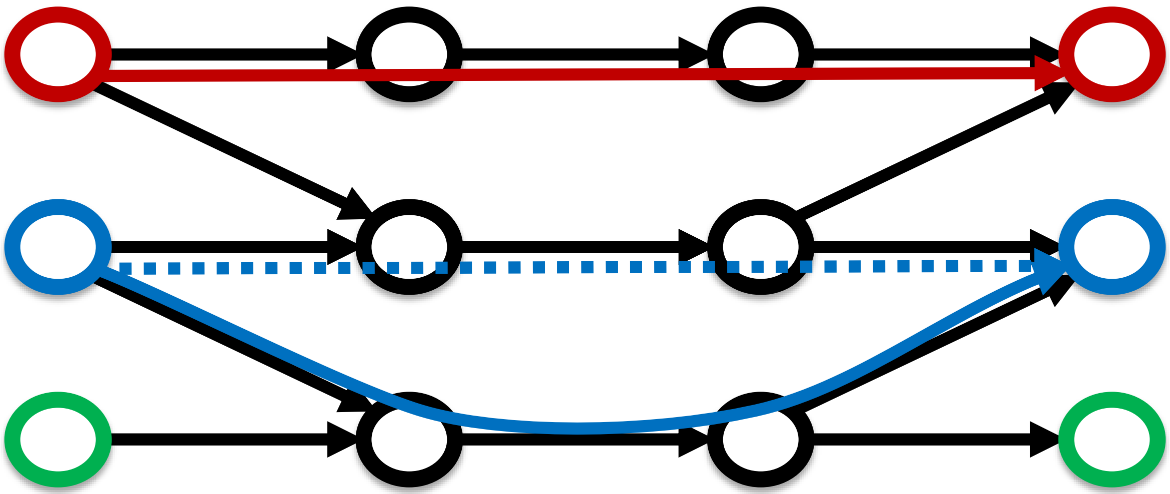
So lets go Back …



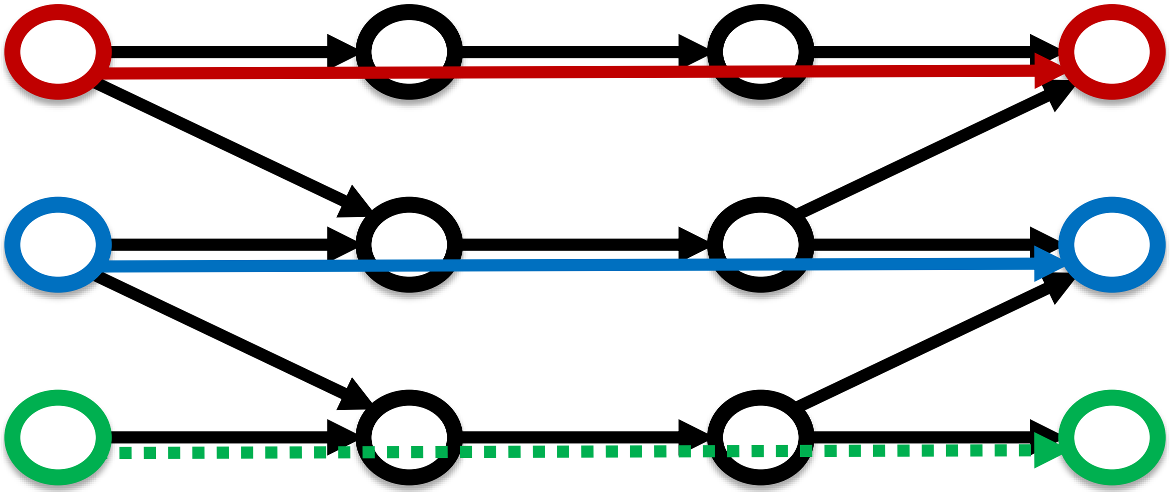
First, Red switches



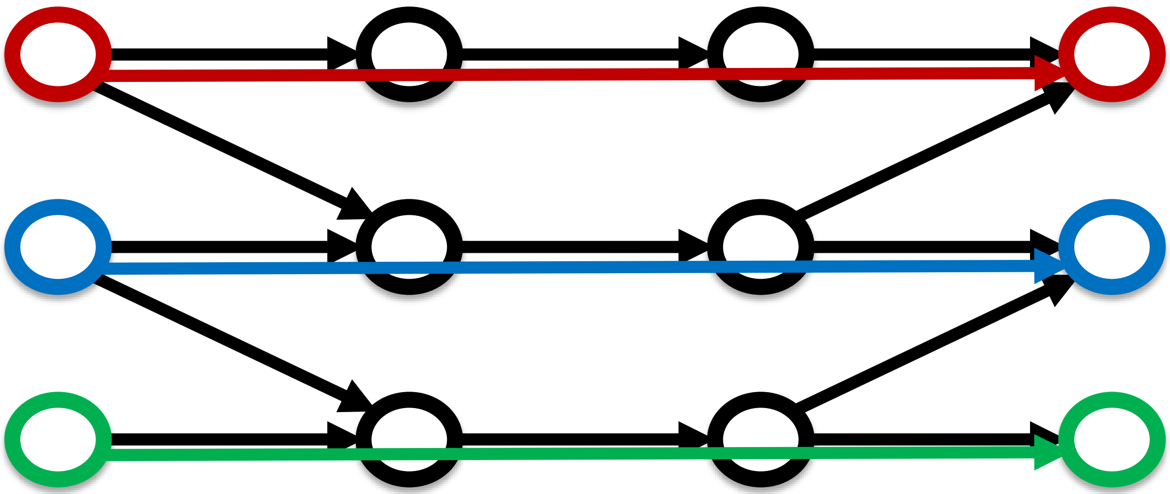
Then, Blue …



And then, Green …



Done



Consistent Migration of Flows

Introduced in *SWAN* (Hong et al., SIGCOMM 2013)

Idea: Flows can be on the old or **new** route

For all edges: $\sum_{\forall F} \max(\mathbf{old}, \mathbf{new}) \leq capacity$

Already for 2 flows

For unsplittable flows:

- Feasibility hardness: **NP-hard**
- Feasibility algorithm: **EXPTIME**

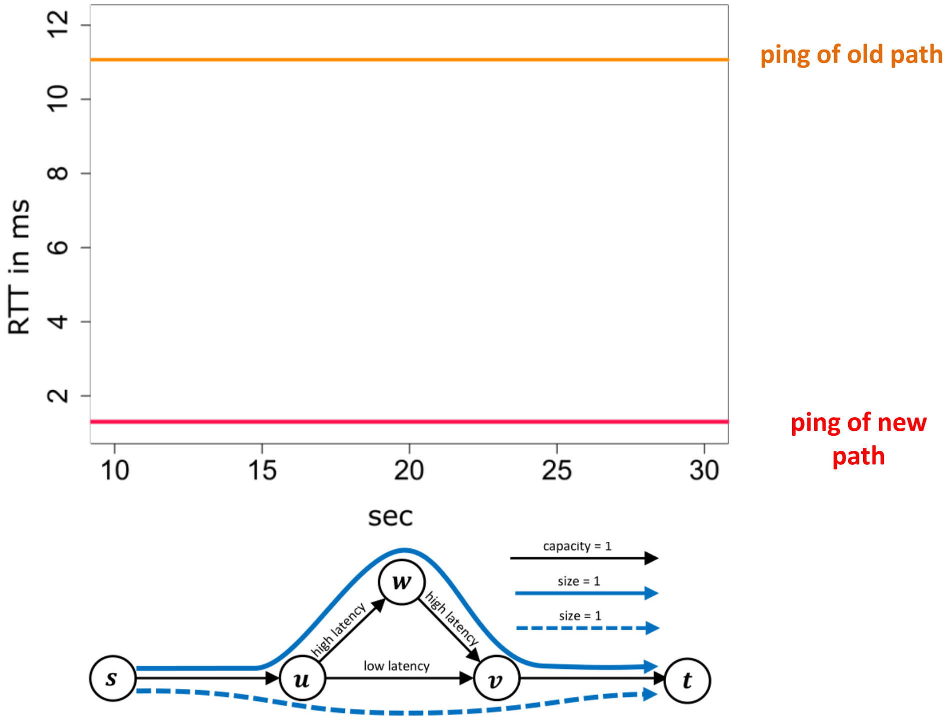
Open question:
Max schedule length ***polynomial***?

For splittable flows:

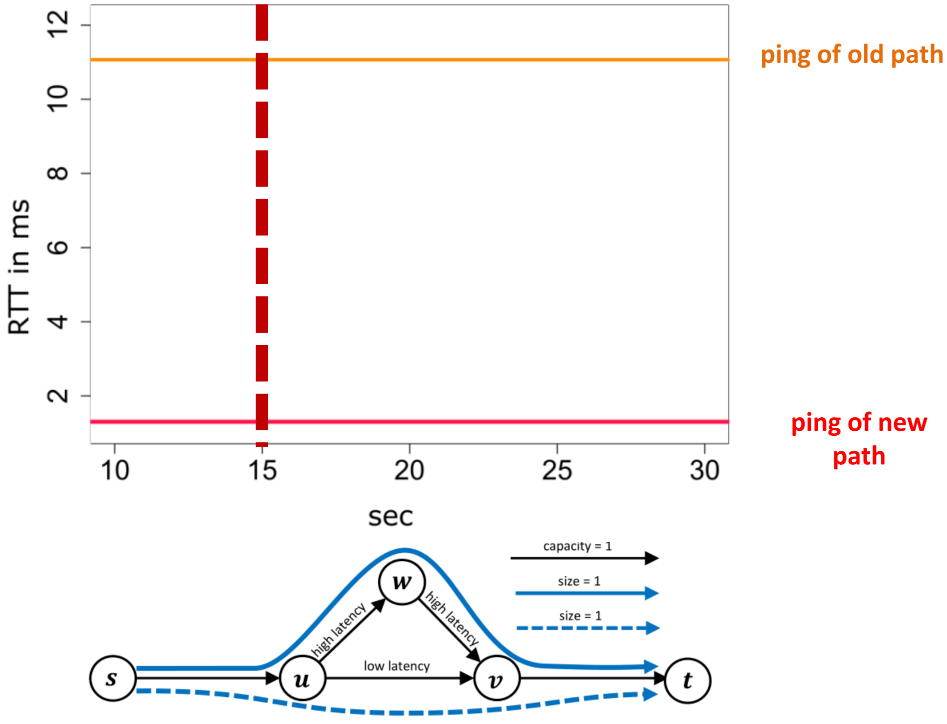
- Feasibility algorithm: **P**
  - Even for super-polynomial schedules
  - For shared destination flows*: schedule length in O(#flows * #edges)

Consistent Migration = Lossless Migration? What are the effects of *time*?

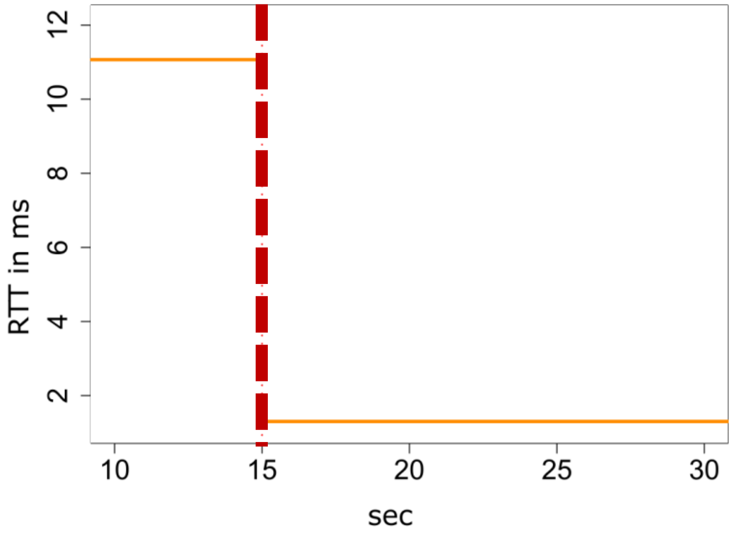
Moving Flows with High Latency

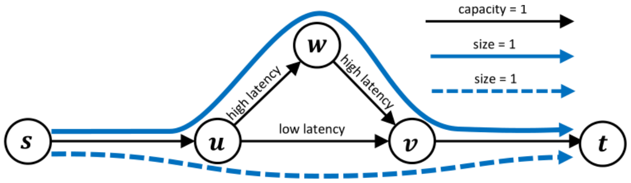


Moving Flows with High Latency

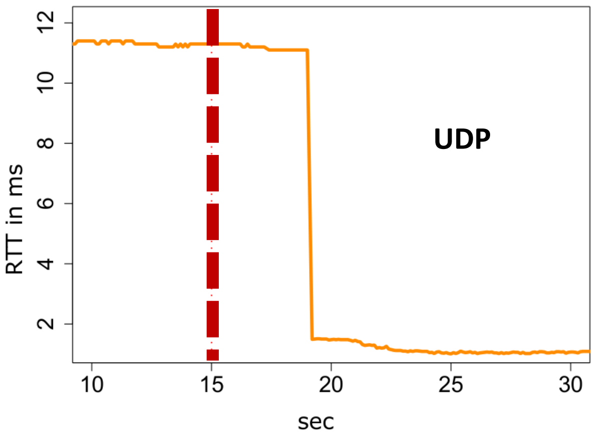


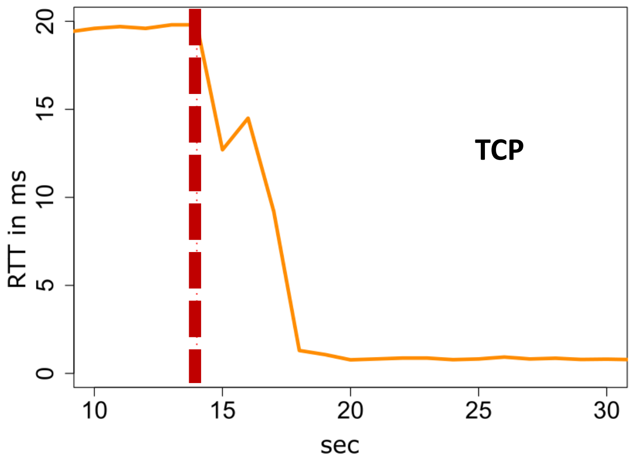
Moving Flows with High Latency

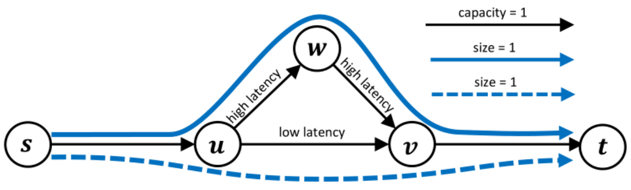




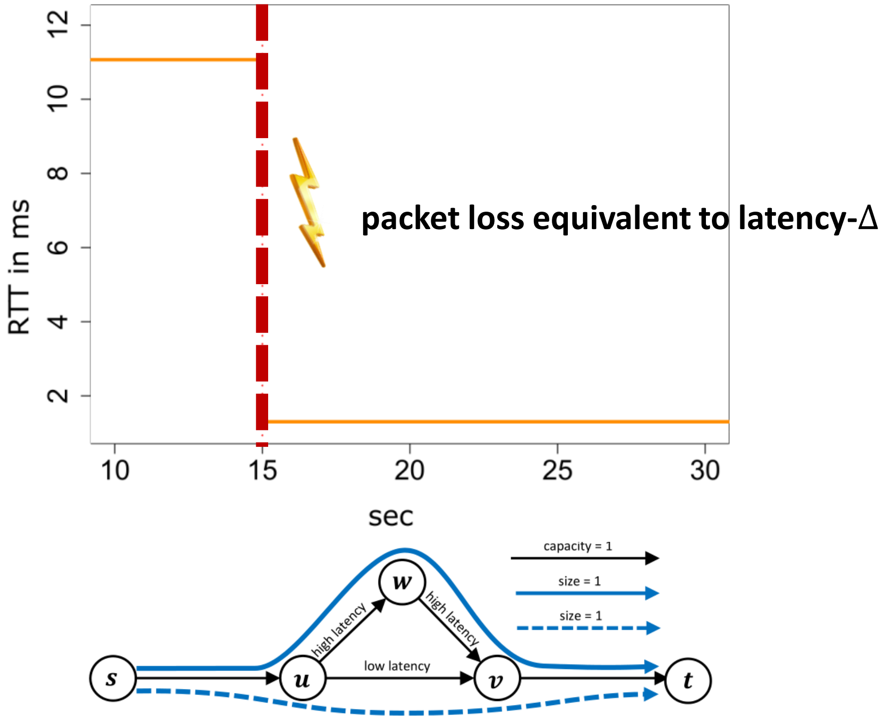
Moving Flows with High Latency







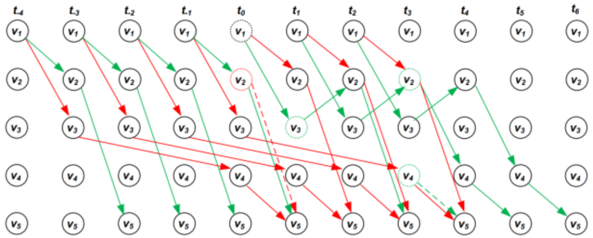
Moving Flows with High Latency



Consistent Flow Migration & ***Time***

- Fixed latencies & single splittable flow: **NP-hard**
  - *Relax:* account for all imaginable latencies: Situation *without* Time

- What if we could enforce *synchronous** updates?
  - Introduced recently in Open Flow by Mizrahi et al. (TimedSDN project)
  - Still NP-hard in general
  - But e.g. for 1 flow and unit latencies:
    - Feasibility in P
      - Using Time-extended graphs



Open Question:
How to synthesize Congestion and/or Time constraints?

Conclusion

- Networks are **critical infrastructures** of our digital society
- But complex to manage and operate today
- Opportunities for **automation** and **formal methods**?
- Challenges, e.g.,
  – Can self-* networks notice their limits? Or **fall back** to „safe/oblivious mode“?
  – Can we learn from self-driving **cars**?

Further Reading

Automated What-if Analysis

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 International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Heraklion, Greece, December 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.

Automated Network Updates

Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio.
IEEE Communications Surveys and Tutorials (**COMST**), to appear.