Case Study: Fast What-If Analysis Tool of MPLS and SR Networks!

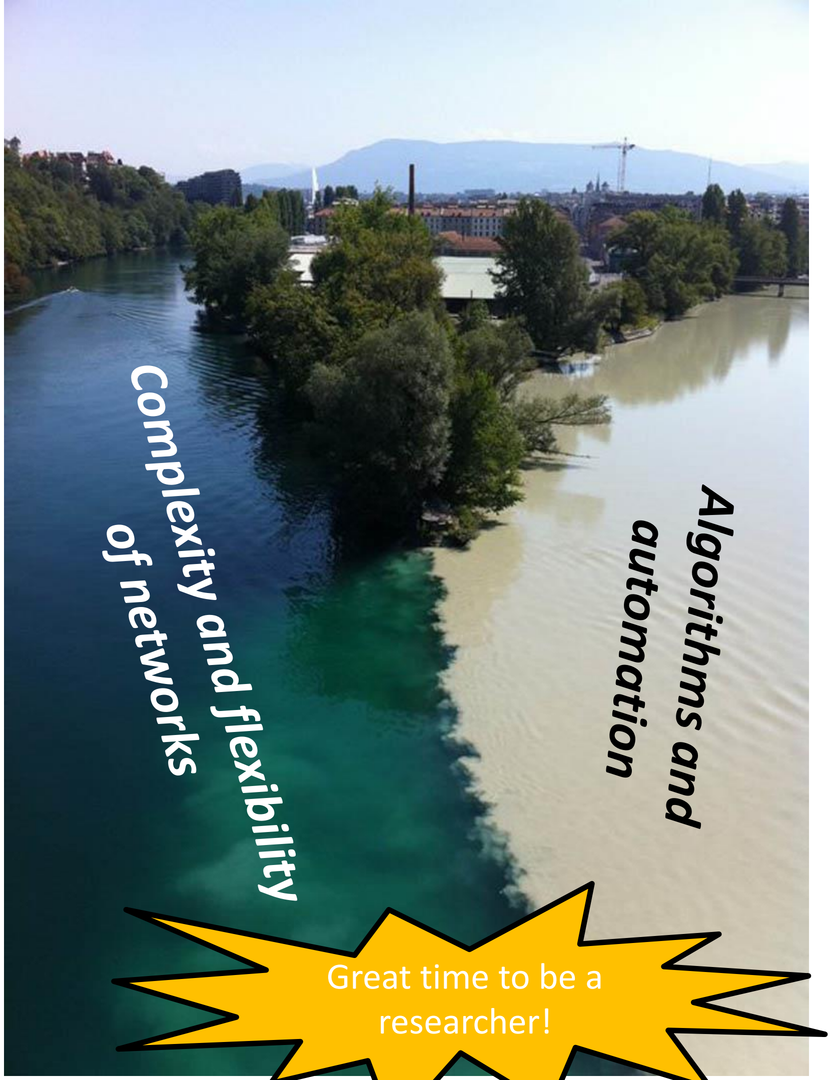
Automating Communication Networks

Stefan Schmid (Uni Vienna)

Communication networks:

- **Critical infrastructure**: stringent dependability requirements
- Opportunities (e.g., **flexibility**) and challenges (complexity)
- Impossible to address ***manually***

A case for **automation** and formal methods?



Rhone and Arve Rivers, Switzerland

Credits: George Varghese.

Part 1: Complexity

Motivation 1: Complexity and Human Errors

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

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.

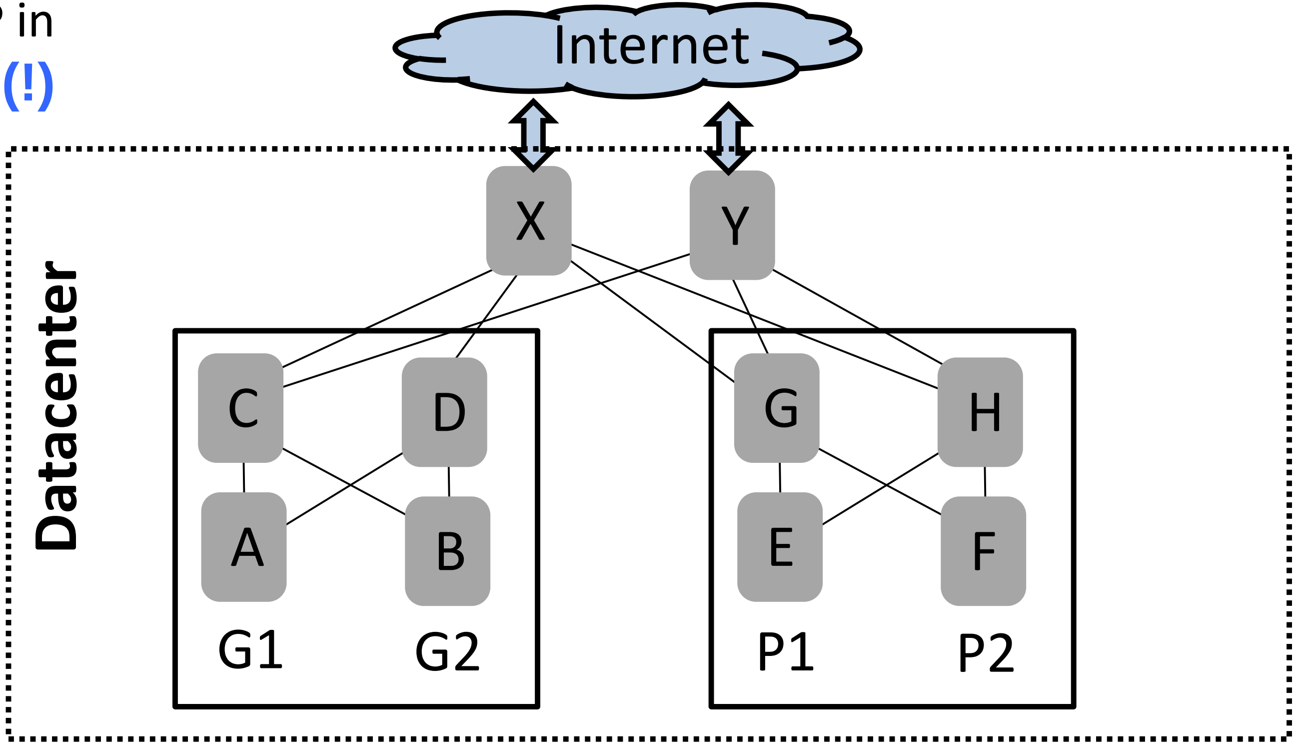
Service outage was due to a series of internal network events that corrupted router data tables.

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

Credits: Nate Foster

Example: Keeping Track of (Flexible) Routes Under Failures

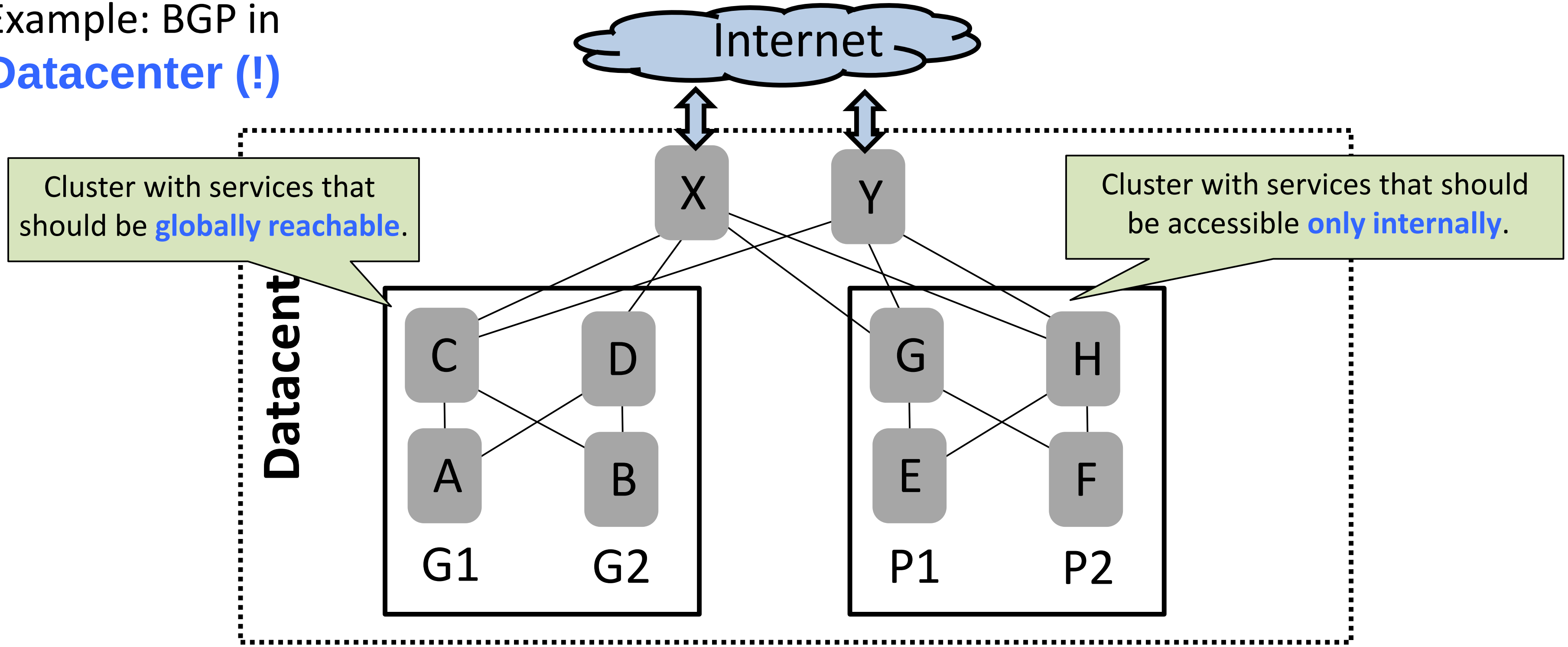
Example: BGP in
Datacenter (!)



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

Example: Keeping Track of (Flexible) Routes Under Failures

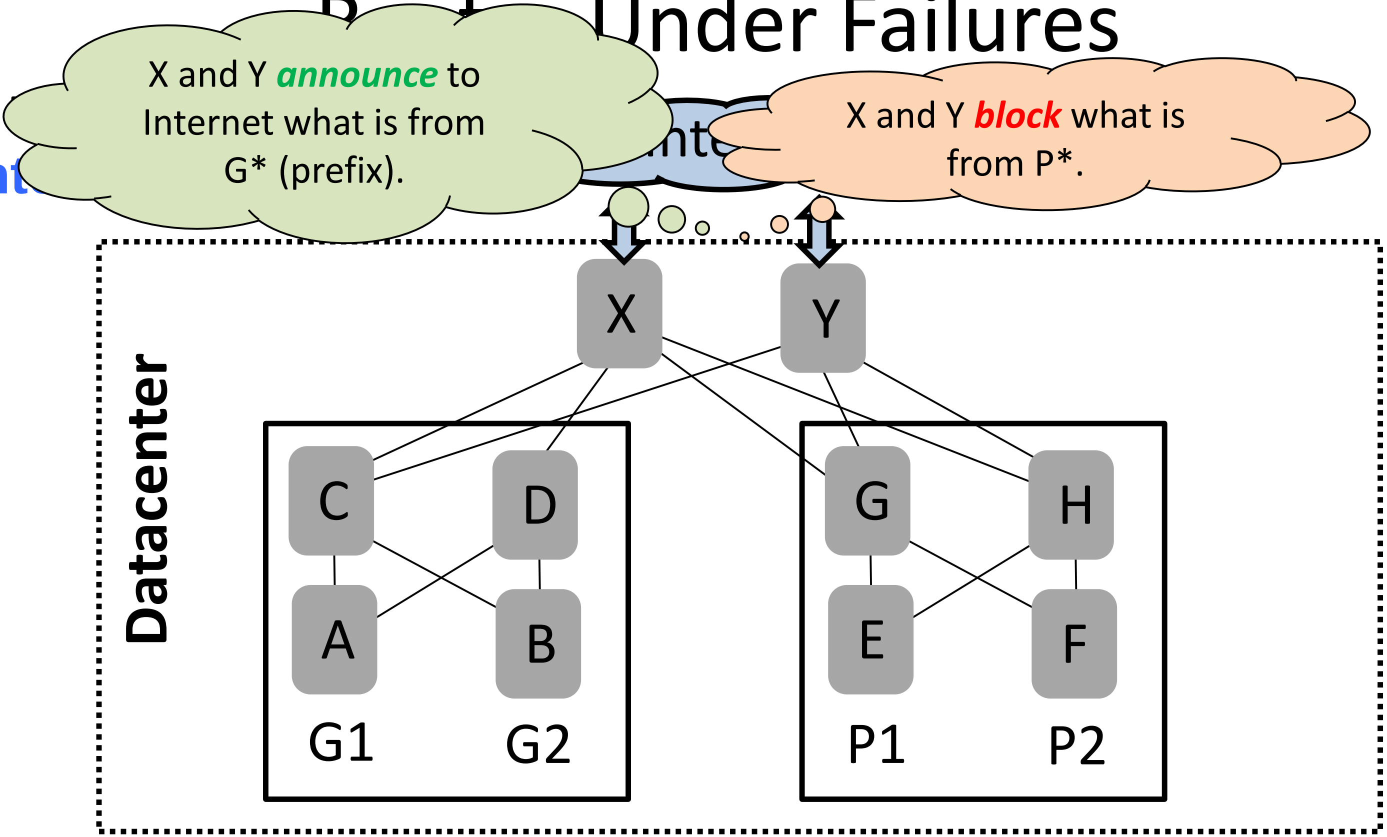
Example: BGP in
Datacenter (!)



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

Example: Keeping Track of (Flexible) Routes Under Failures

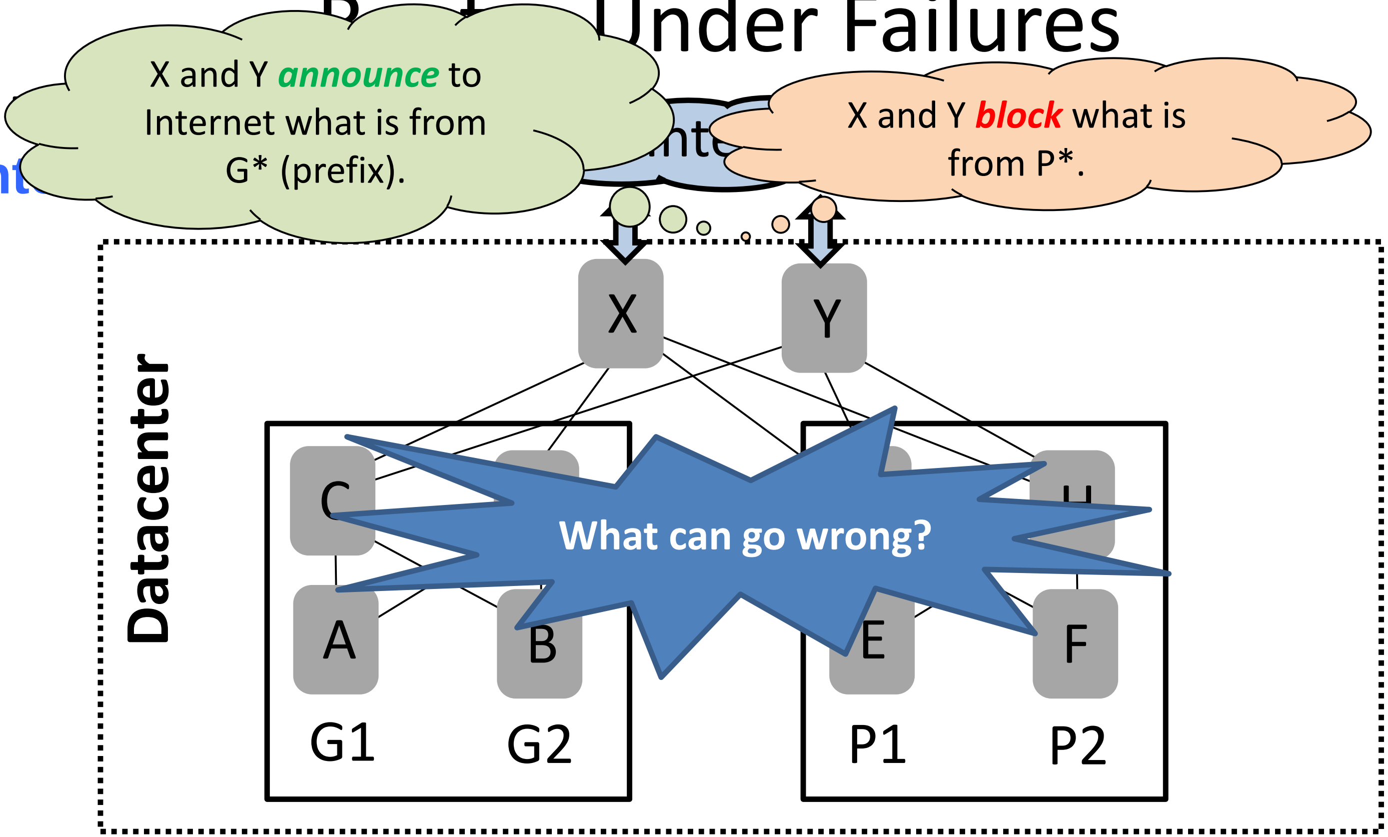
Example:
Datacent



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

Example: Keeping Track of (Flexible)
Under Failures

Example:
Datacent



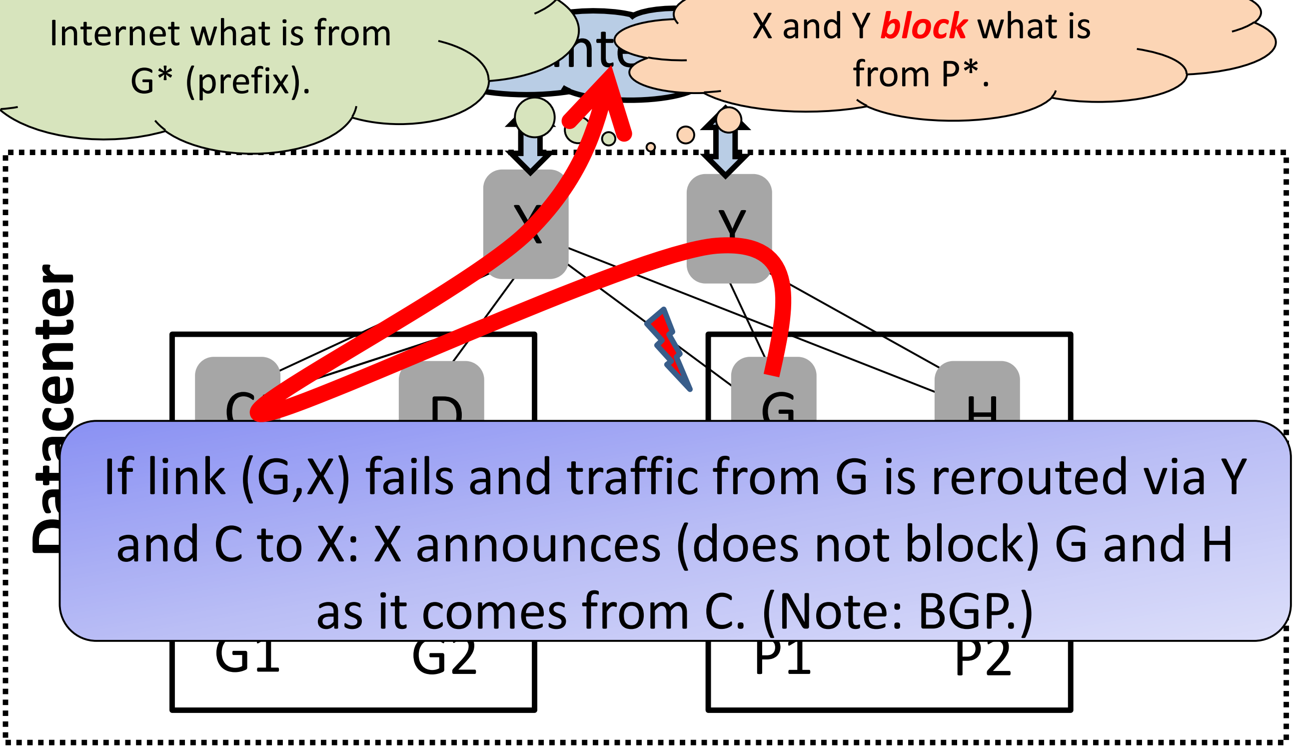
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Example: Keeping Track of (Flexible) Routes Under Failures

Example:
Datacent

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.

Managing Complex Networks is Hard for Humans

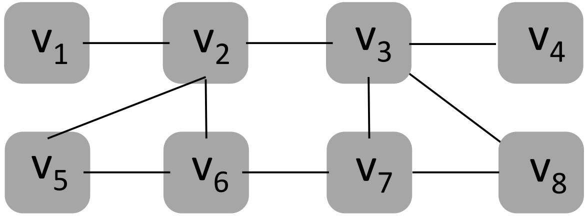
The Case for Automation!

Role of Formal Methods?

Efficiency?!

Example: MPLS Networks

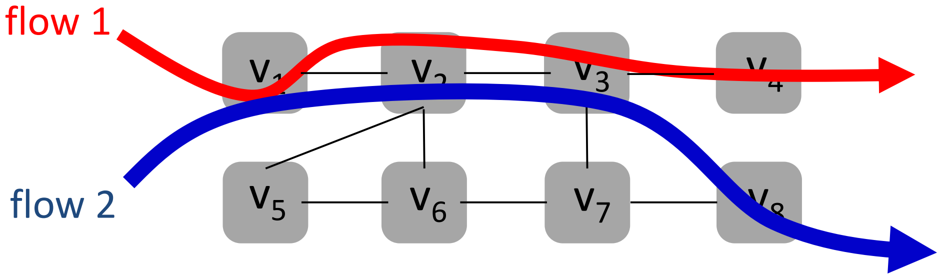
- MPLS: forwarding based on **top label** of label **stack**



Default routing of
two flows

Example: MPLS Networks

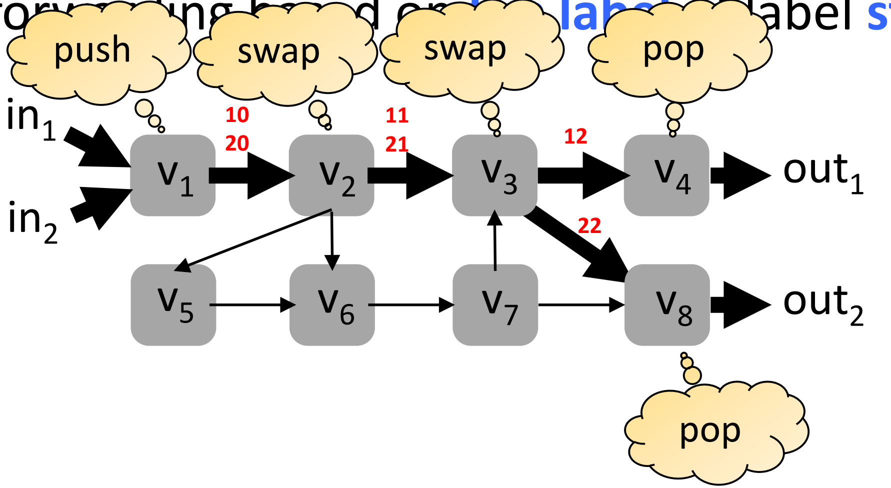
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

Example: MPLS Networks

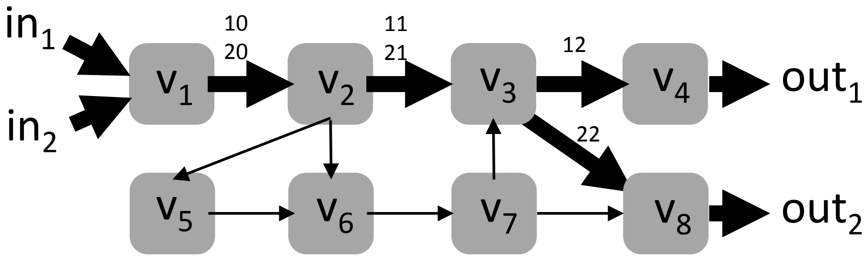
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

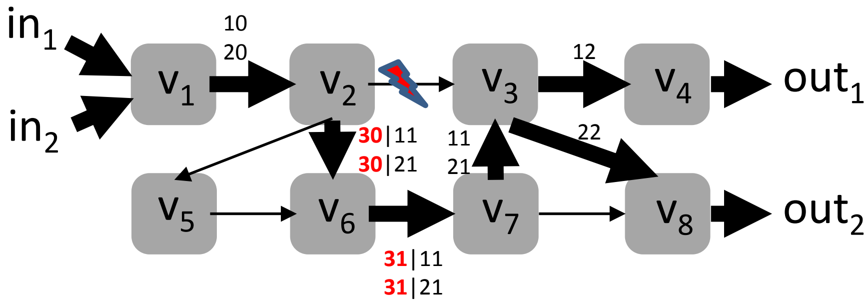
Fast Reroute Around *1 Failure*

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



Default routing of two flows

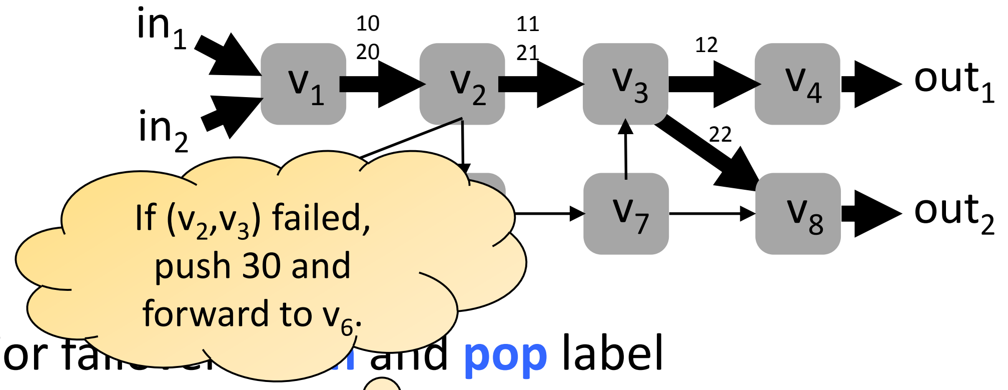
- For failover: **push** and **pop** label



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

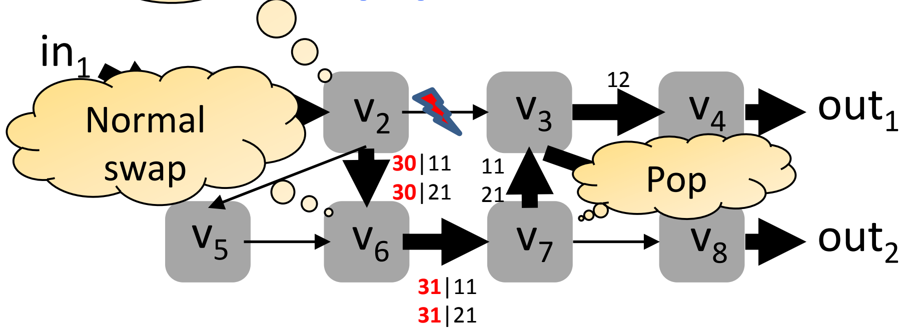
Fast Reroute Around *1 Failure*

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



Default routing of two flows

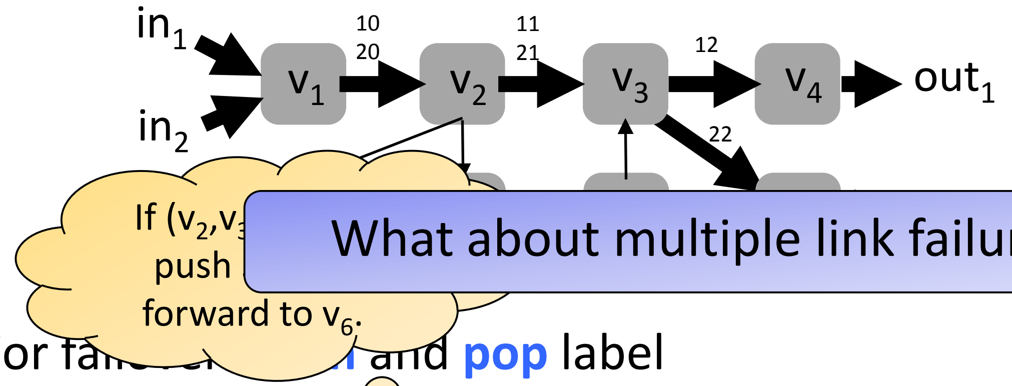
- For fa … and **pop** label



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

Fast Reroute Around *1 Failure*

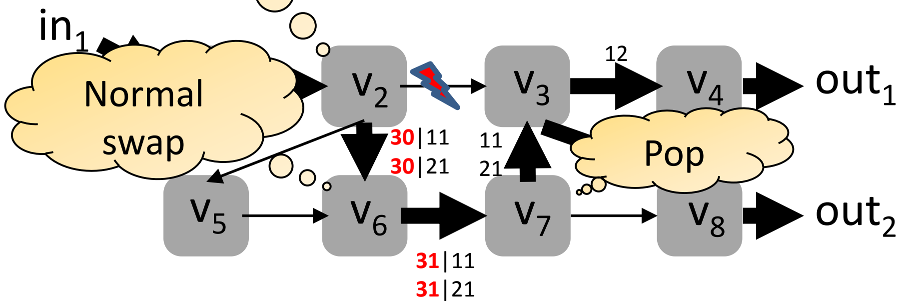
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

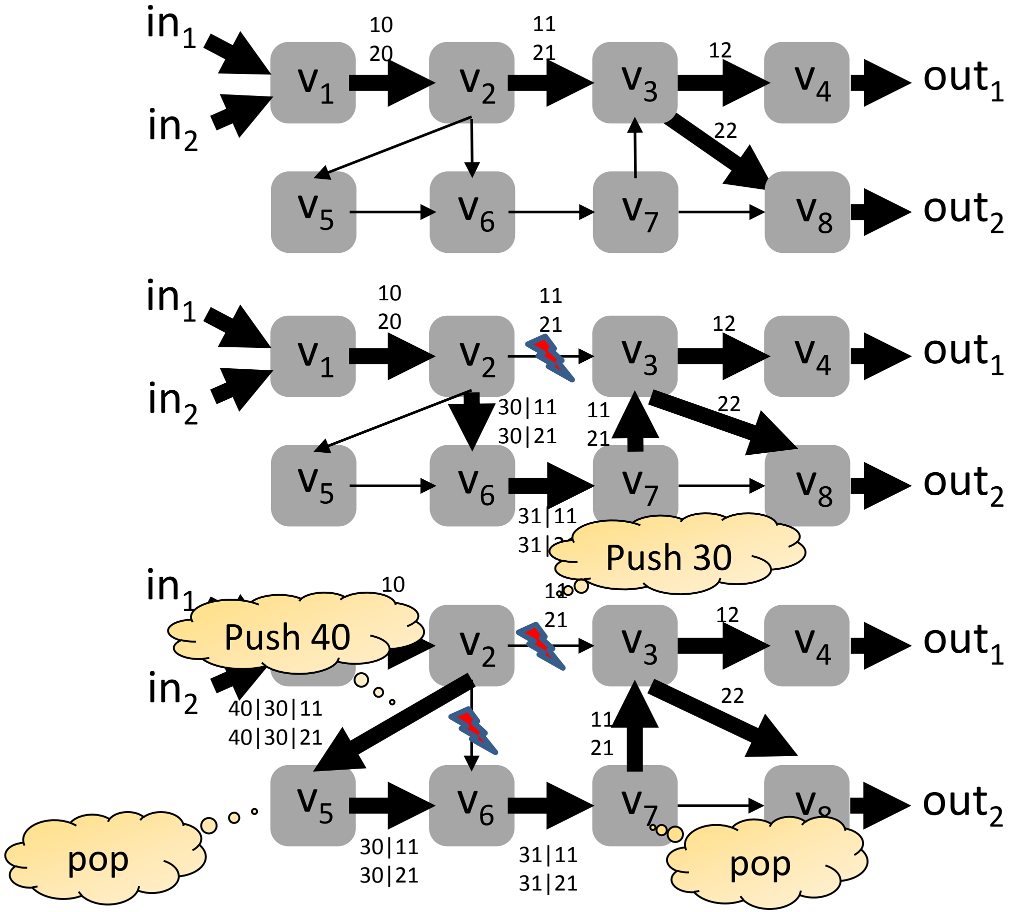
What about multiple link failures?

- For fa... and **pop** label



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

2 Failures: Push *Recursively*



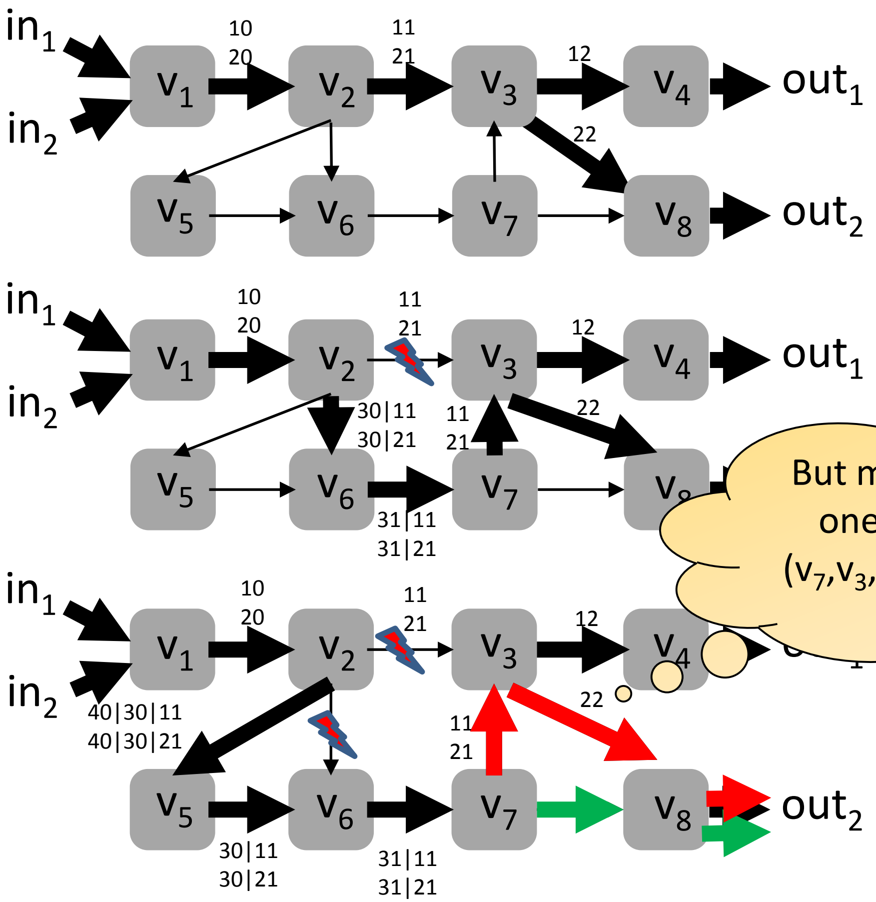
Original Routing

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

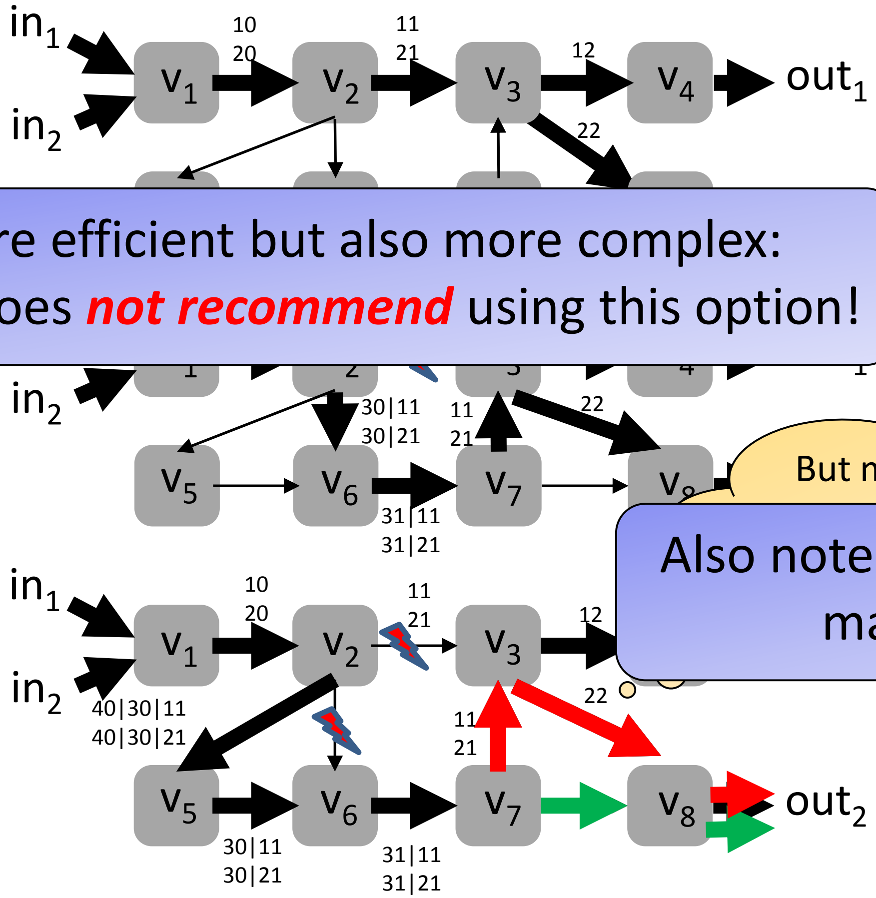
One failure: push 30:

30: route around (v_2,v_3)

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

Push recursively 40: route around (v_2,v_6)

2 Failures: Push *Recursively*



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

Original Routing

One failure: push 30:

But masking links one-by-

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)	$push(40)$

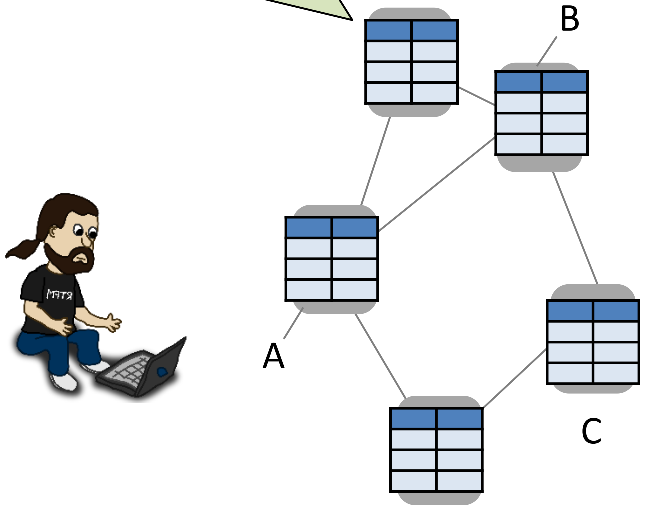
global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

Failover Tables

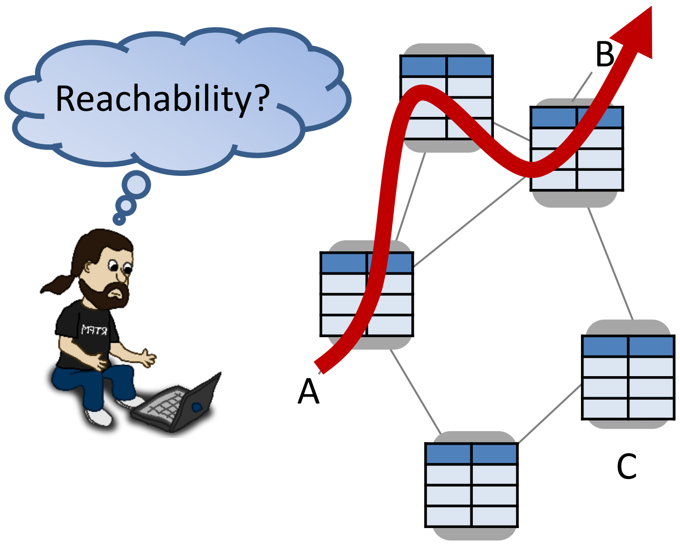
MPLS Tunnels in Today‘s ISP Networks

Responsibilities of a Sysadmin

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



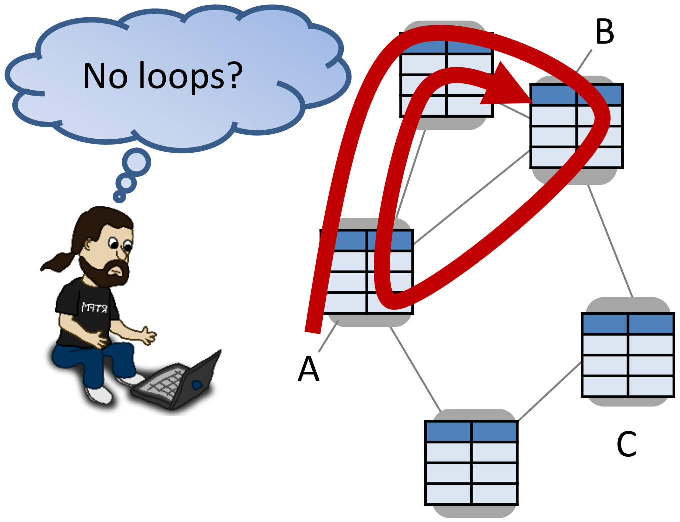
Responsibilities of a Sysadmin



Sysadmin responsible for:

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

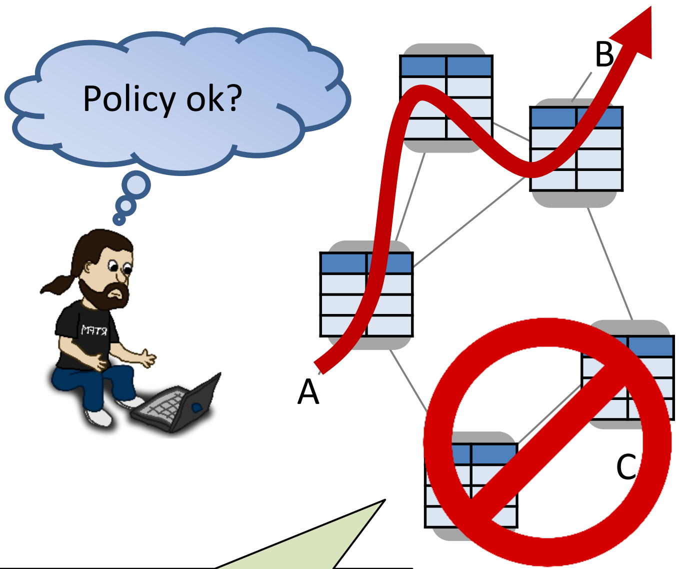
Responsibilities of a Sysadmin



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

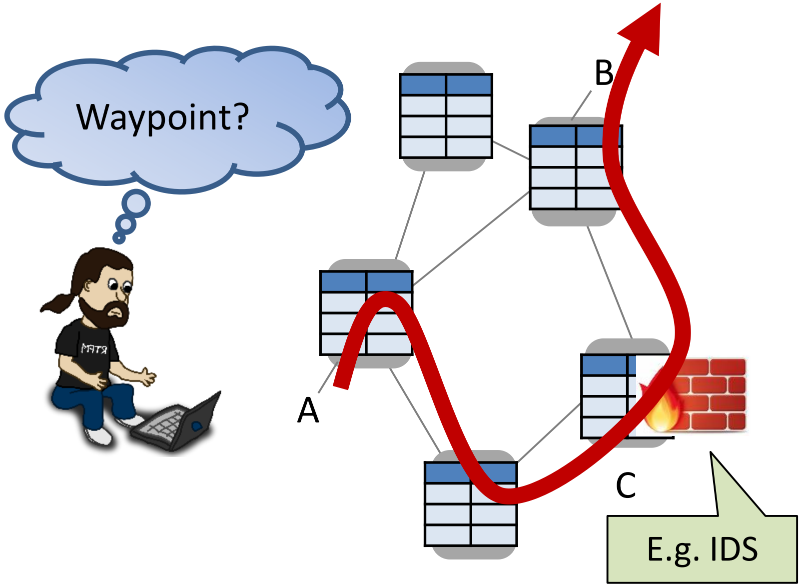


E.g. ***NORDUnet***: no traffic via Iceland (expensive!).

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?

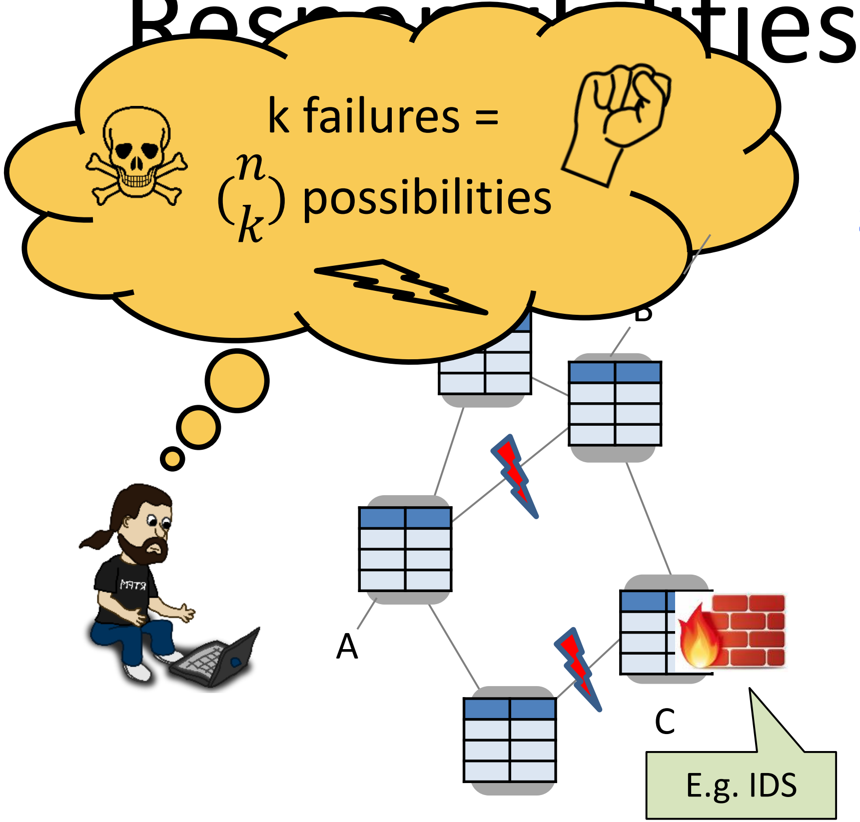
Responsibilities of a Sysadmin



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)?

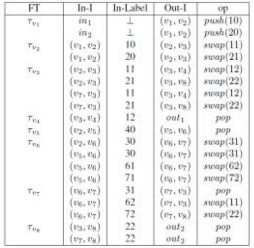
… and everything even under multiple failures?!

So what formal methods offer here?

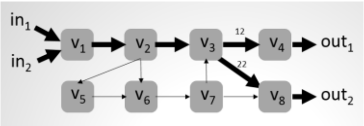
A lot! Automated **What-if Analysis Tool** for MPLS and SR in ***polynomial time***.

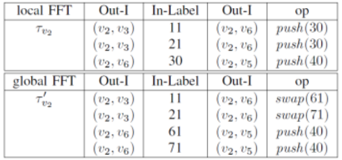
(INFOCOM 2018, CoNEXT 2018)

Leveraging Automata-Theoretic Approach



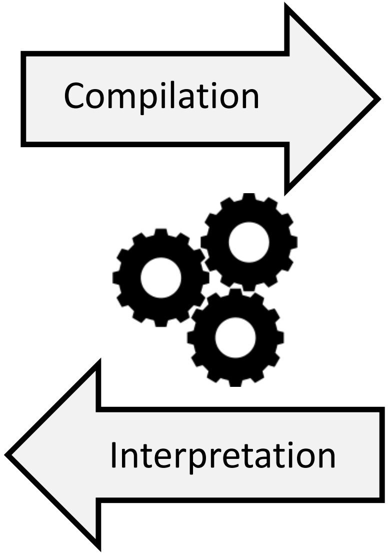
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\bot$	(v_1, v_2)	$push(10)$
	in_2	$\bot$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$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



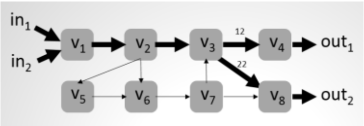
$$\mathrm{pX} \Rightarrow \mathrm{qXX}$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

Pushdown Automaton and **Prefix Rewriting Systems** Theory

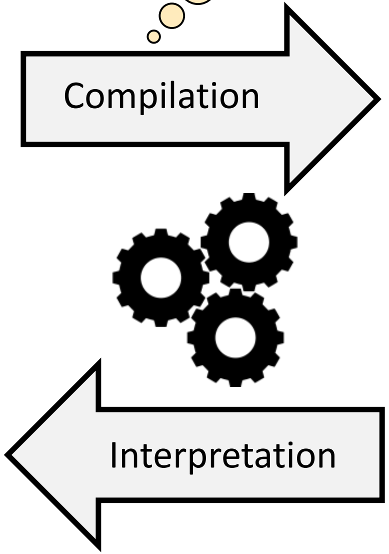
Leveraging Autom oach

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)	$push(40)$



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

MPLS **configurations**, Segment Routing etc.

Pushdown Automaton and **Prefix Rewriting Systems** Theory

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

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

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

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

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**:

... on interface...

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

... given that these links are down.

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

Node receives...

... packet with header...

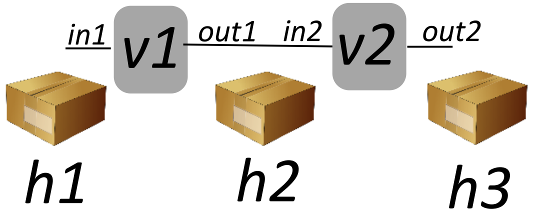
... with new header..

- 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 label on stack

Push:

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

Swap top of stack

Swap:

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

Pop top of stack

Pop:

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

Example *Failover* Rules

Emumerate all rerouting options

Failover-Push:

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

Failover-Swap:

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

Failover-Pop:

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

Example rewriting sequence:

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

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 (from Switzerland)!

- 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

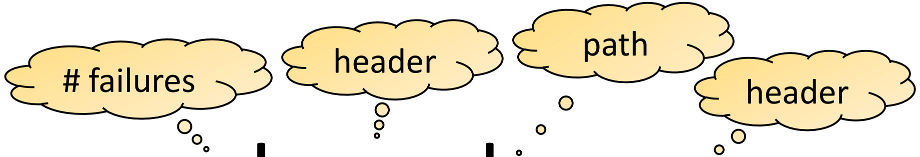
Preliminary Tool and Query Language

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

- In Python 3

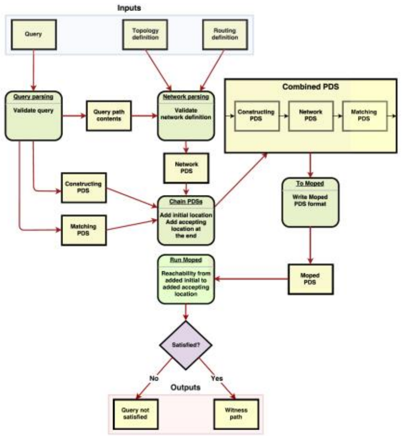
<u>Part 2:</u> Reachability analysis of constructed PDS

- Using ***Moped*** tool



k <a> b <c>

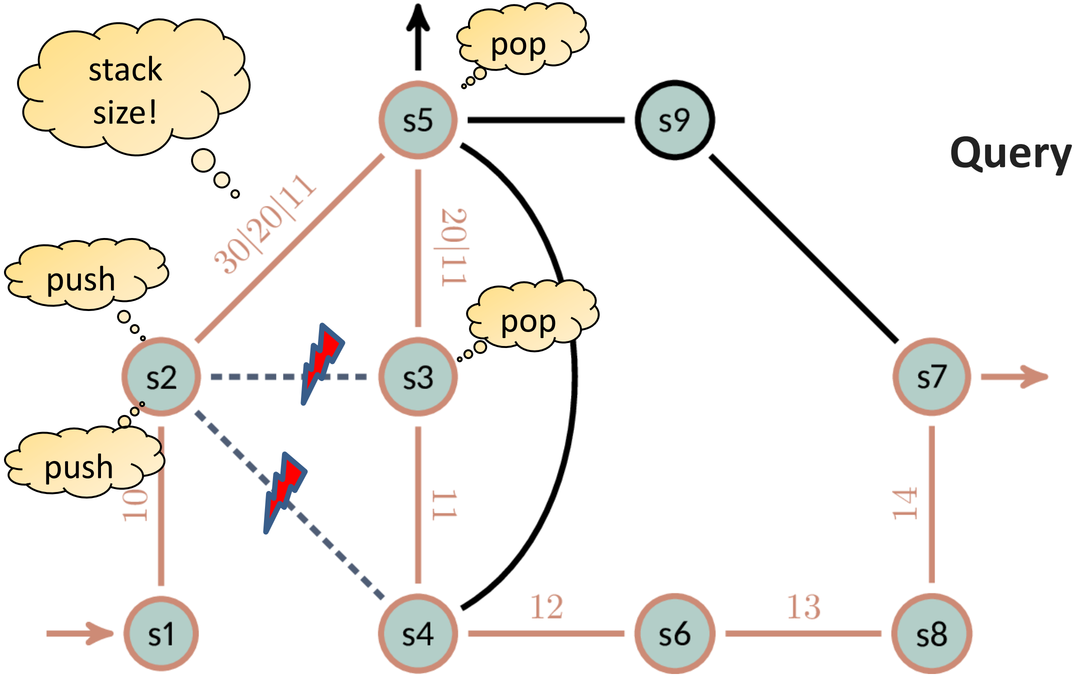
Regular query language



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!)

Industrial Case Study with NORDUnet

The „Switch.Ch“ of Scandinavia?

Much faster for many failures!

- 24 MPLS routers, across several countries
- ***1 million*** forwarding rules
- Queries like: „Is it ensured traffic **never routed via Iceland**?“
- ***20 million*** PDA transitions but fast!

P-Rex HSA	$k = 0$	$k = 1$	$k = 2$	$k = 3$
Nesting: 0 Routers: 5	0.6 0.2	0.6 0.1	0.6 0.1	0.6 0.2
Nesting: 1 Routers: 10	0.6 0.1	0.6 0.1	0.6 0.4	0.6 3.7
Nesting: 2 Routers: 15	0.6 0.1	0.6 0.3	0.6 1.9	0.6 55.9
Nesting: 3 Routers: 20	0.6 0.1	0.6 0.3	0.6 6.8	0.6 335.6
Nesting: 4 Routers: 25	0.6 0.1	0.6 0.6	0.6 16.4	0.6 567.2
Nesting: 5 Routers: 30	0.6 0.1	0.6 1.0	0.6 34.6	0.6 1901.1
Nesting: 6 Routers: 35	0.6 N/A	0.6 N/A	0.6 N/A	0.7 N/A

runtime in sec

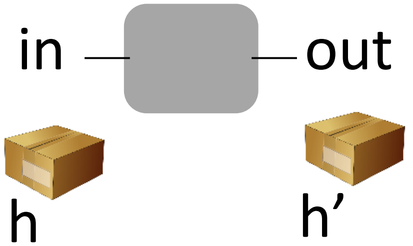
Related Work

	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

Our approach

But What About Other Networks?

The **clue**: exploit the specific structure of MPLS rules.



Rules match the header **h** of packets arriving at ***in***, and define to which port ***out*** to forward as well as new header **h'**.

Rules of general networks (e.g., SDN):

arbitrary header rewriting

in x L ⟶ out x L**

VS

(Simplified) MPLS rules:

prefix rewriting

*in x L ⟶ out x **OP***

where *OP* = {*swap,push,pop*}

What About Performance/QoS Properties?

WNetKAT: A Weighted SDN Programming and Verification Language*

Kim G. Larsen[1], Stefan Schmid[2], and Bingtian Xue[3]

1 **Aalborg University, Aalborg, Denmark**
kgl@cs.aau.dk
2 **Aalborg University, Aalborg, Denmark**
schmiste@cs.aau.dk
3 **Aalborg University, Aalborg, Denmark**
bingt@cs.aau.dk

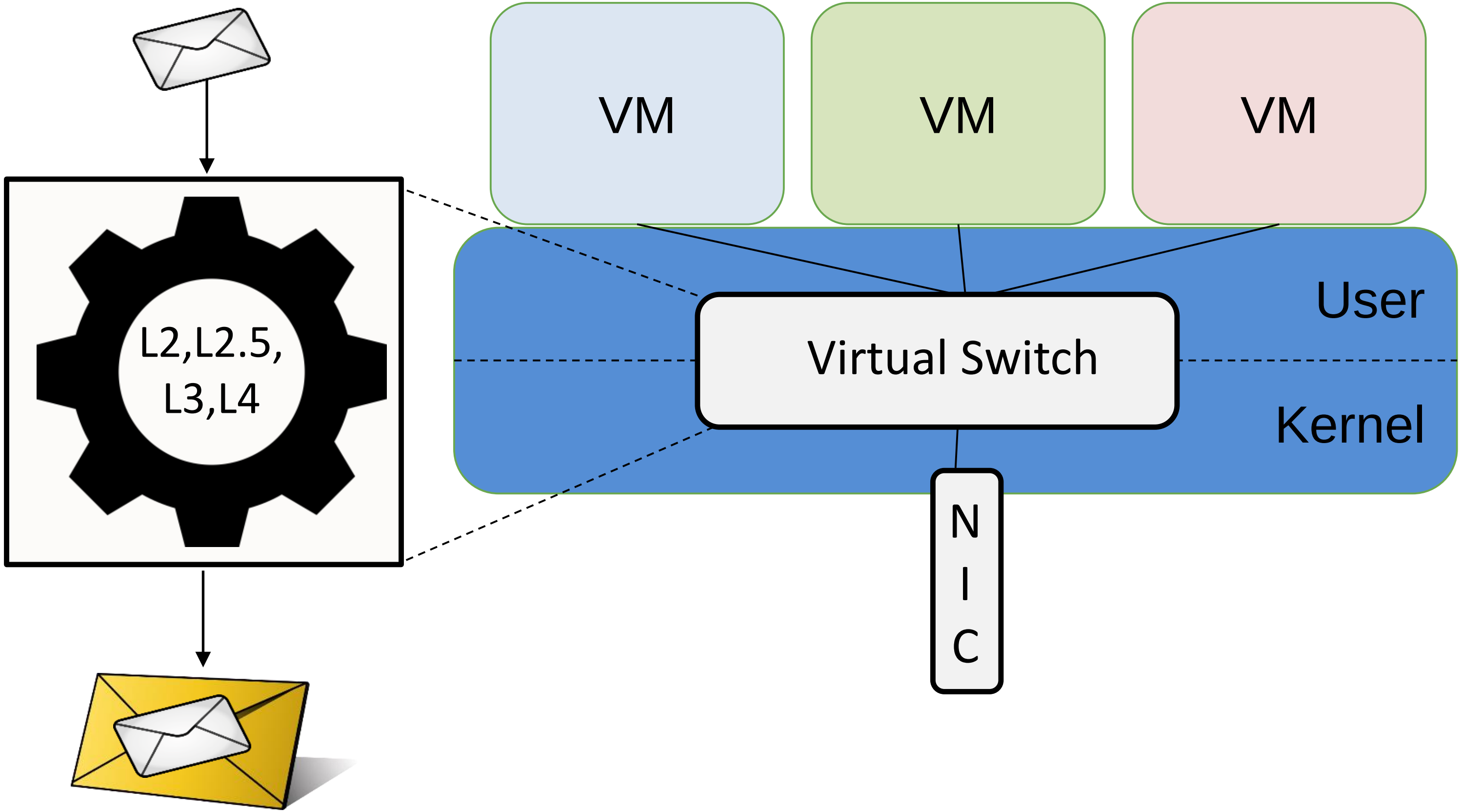
Abstract

Programmability and verifiability lie at the heart of the software-defined networking paradigm. While OpenFlow and its match-action concept provide primitive operations to manipulate hardware configurations, over the last years, several more expressive network programming languages have been developed. This paper presents *WNetKAT*, the first network programming language accounting for the fact that networks are inherently weighted, and communications subject to capacity constraints (e.g., in terms of bandwidth) and costs (e.g., latency or monetary costs). *WNetKAT* is based on a syntactic and semantic extension of the NetKAT algebra. We demon-

OPODIS 2016

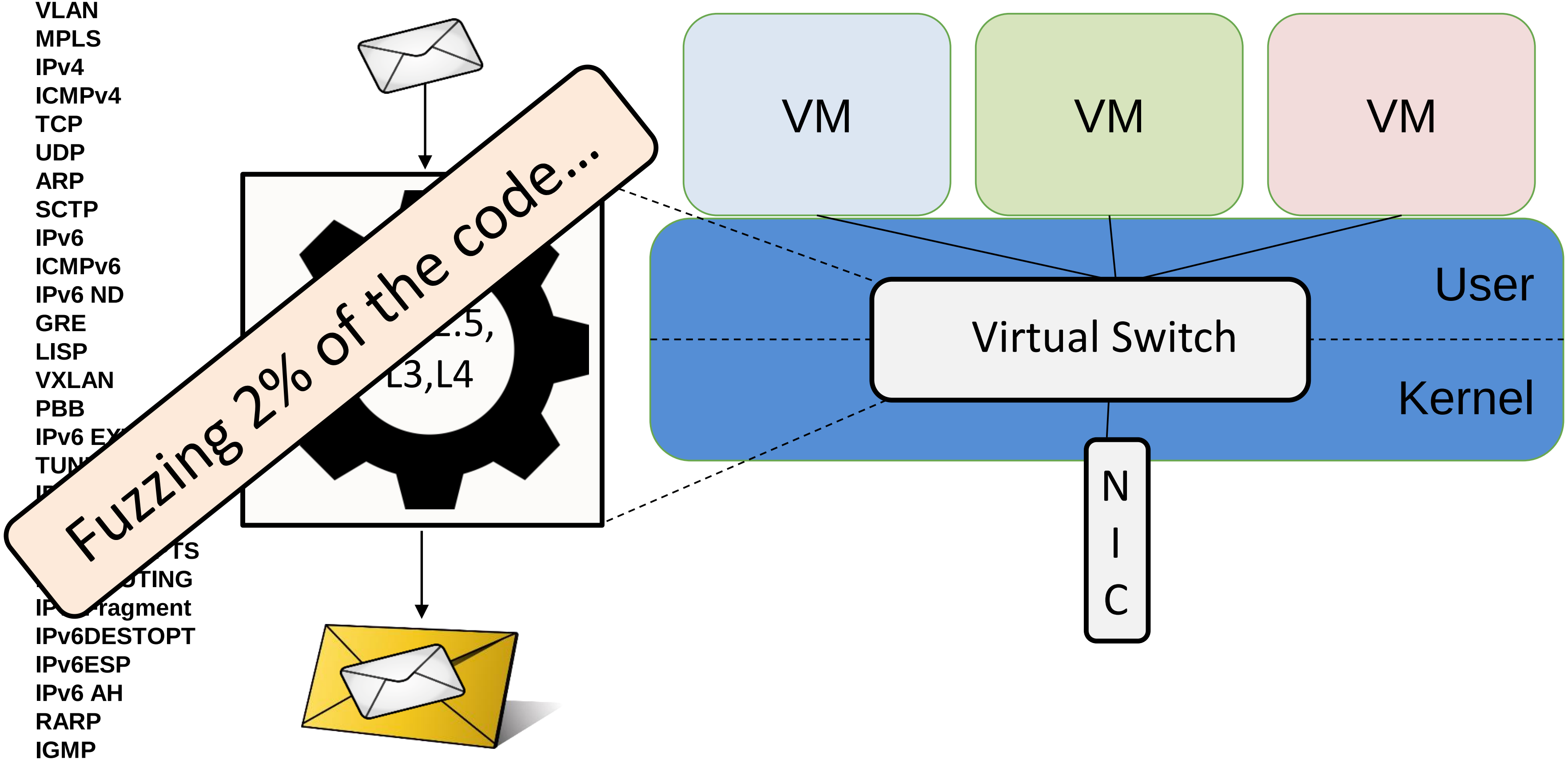
Further Complexities: (Unified) Packet Parsing
Virtual Switches, e.g., MPLS

Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP

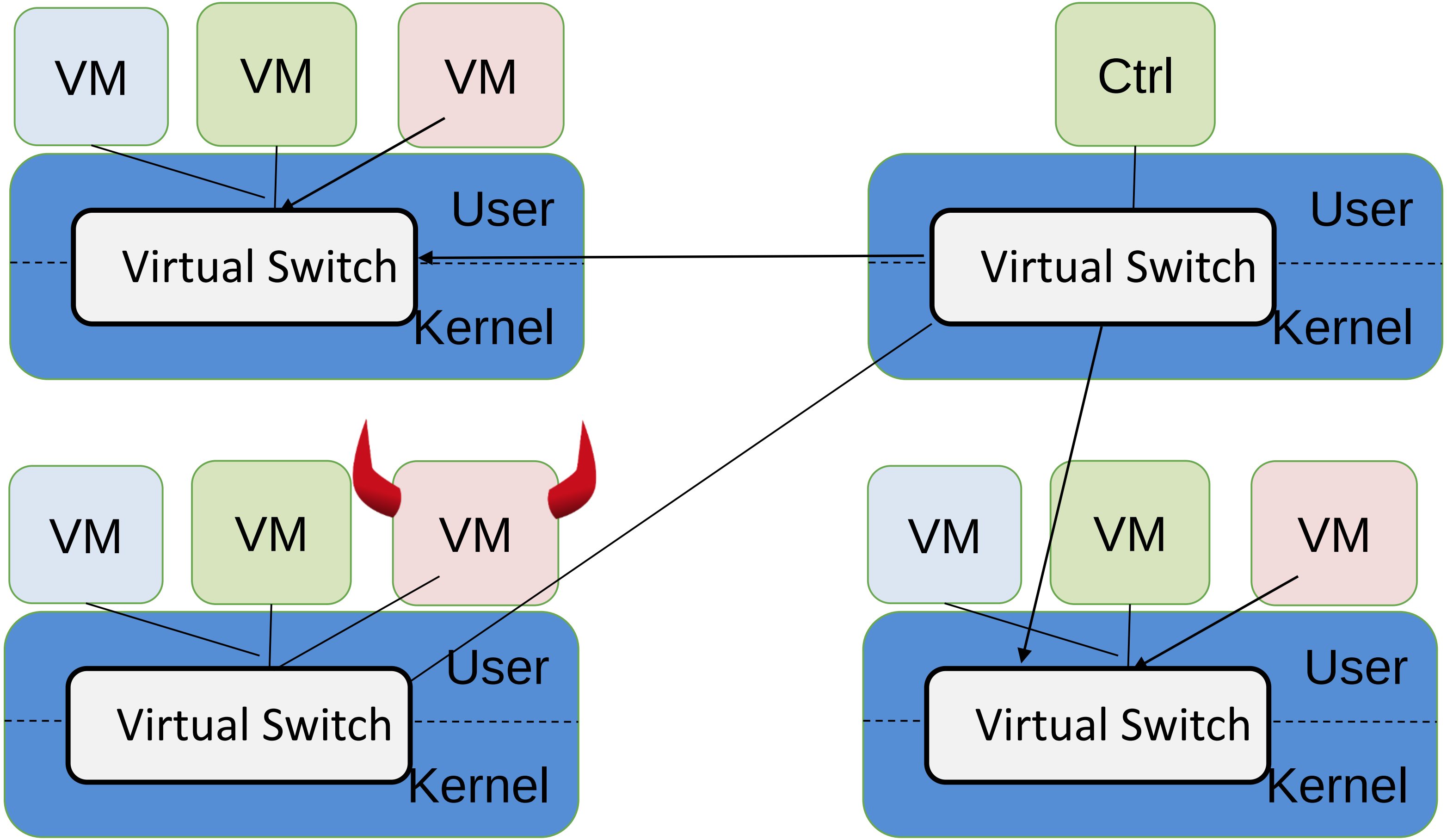


Further Complexities: (Unified) Packet Parsing
Virtual Switches, e.g., MPLS

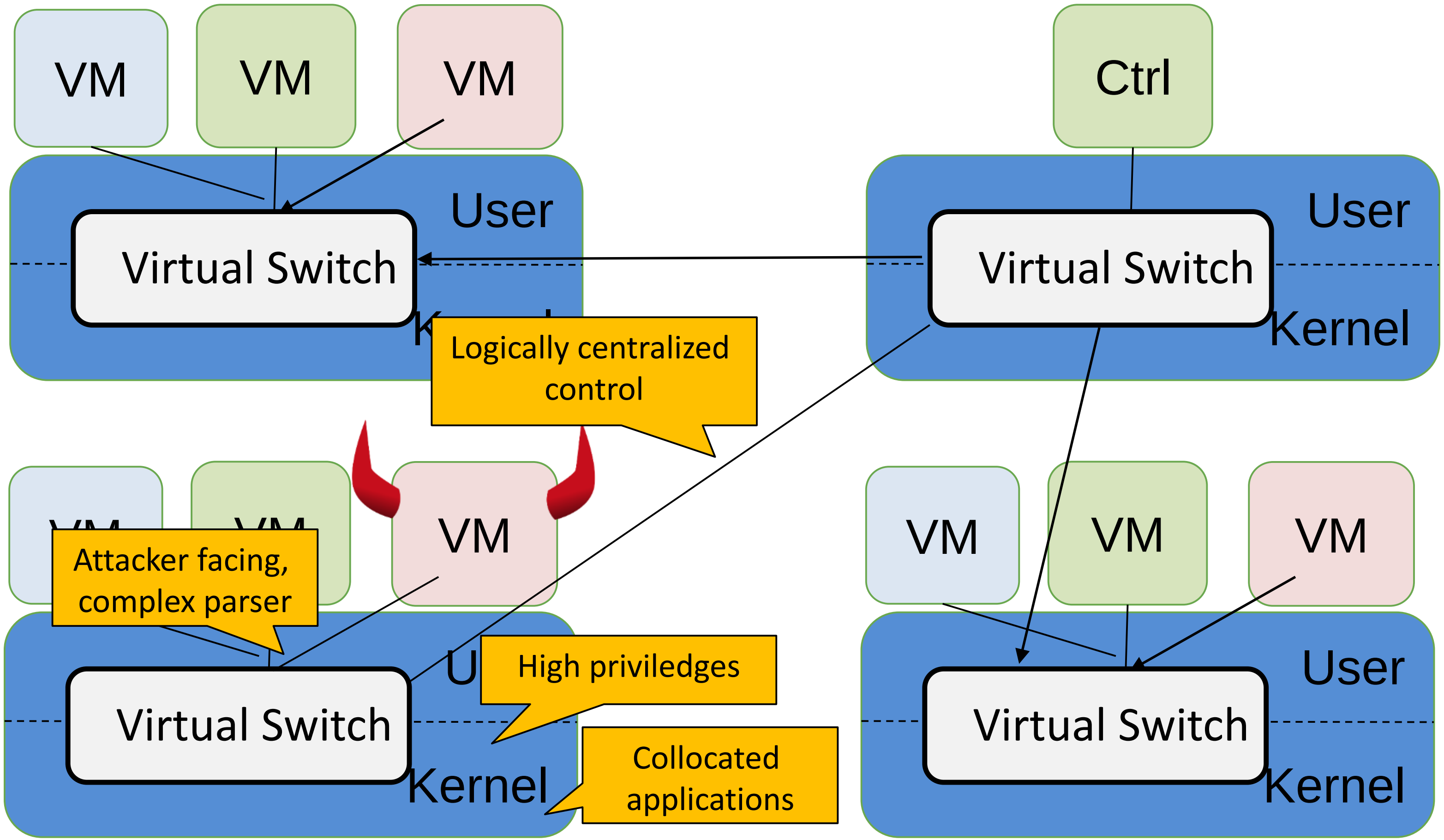
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EX
TUN
IP
TS
UTING
IP ragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



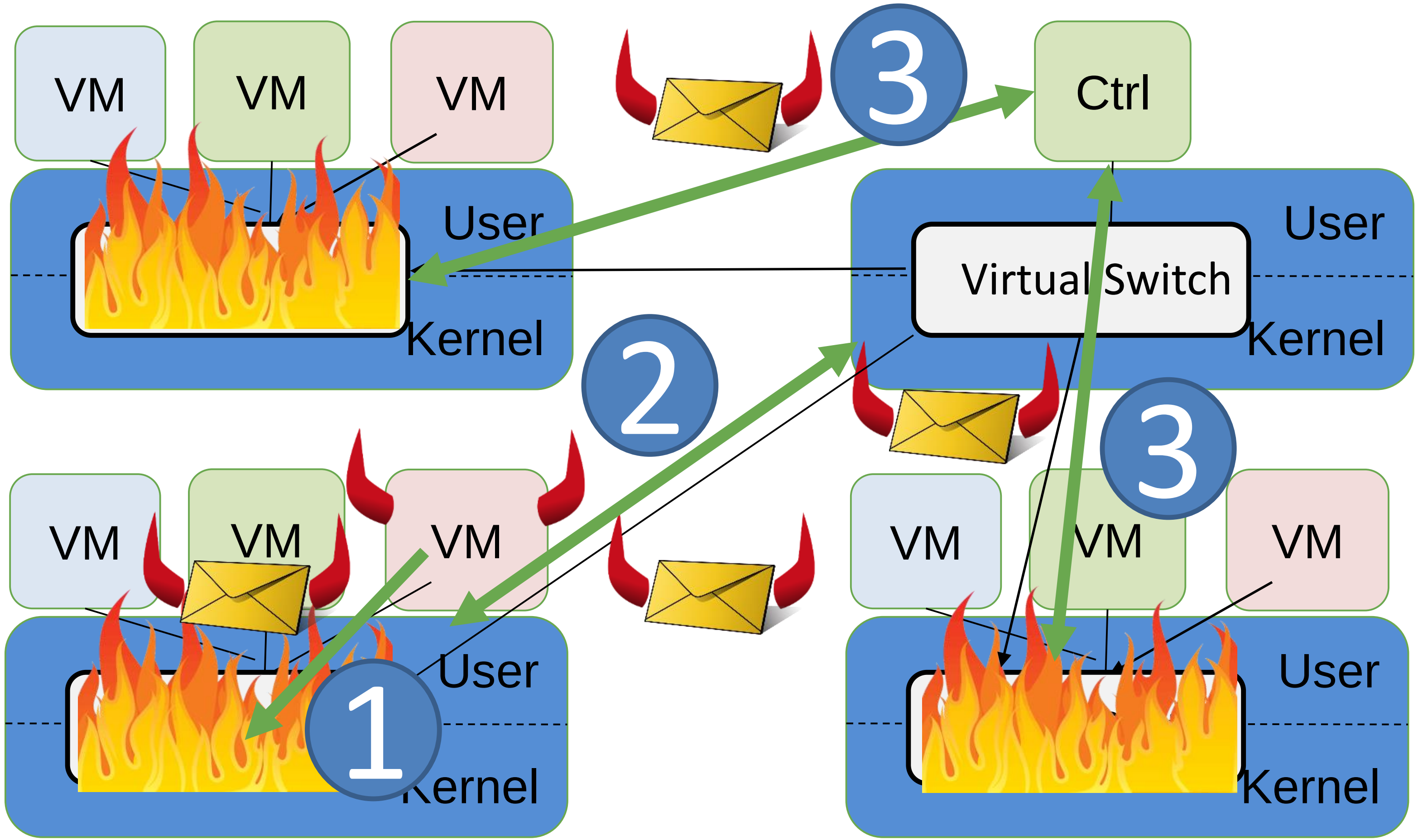
Compromising the Cloud (SOSR 2018)



Compromising the Cloud (SOSR 2018)



Compromising the Cloud (SOSR 2018)



Part 2: Flexibility

Motivation 2: Flexibility

The next frontier!

Traffic engineering beyond "***weights***": SDN, MPLS, …

Centralized control, application-specific forwarding, **virtualization**

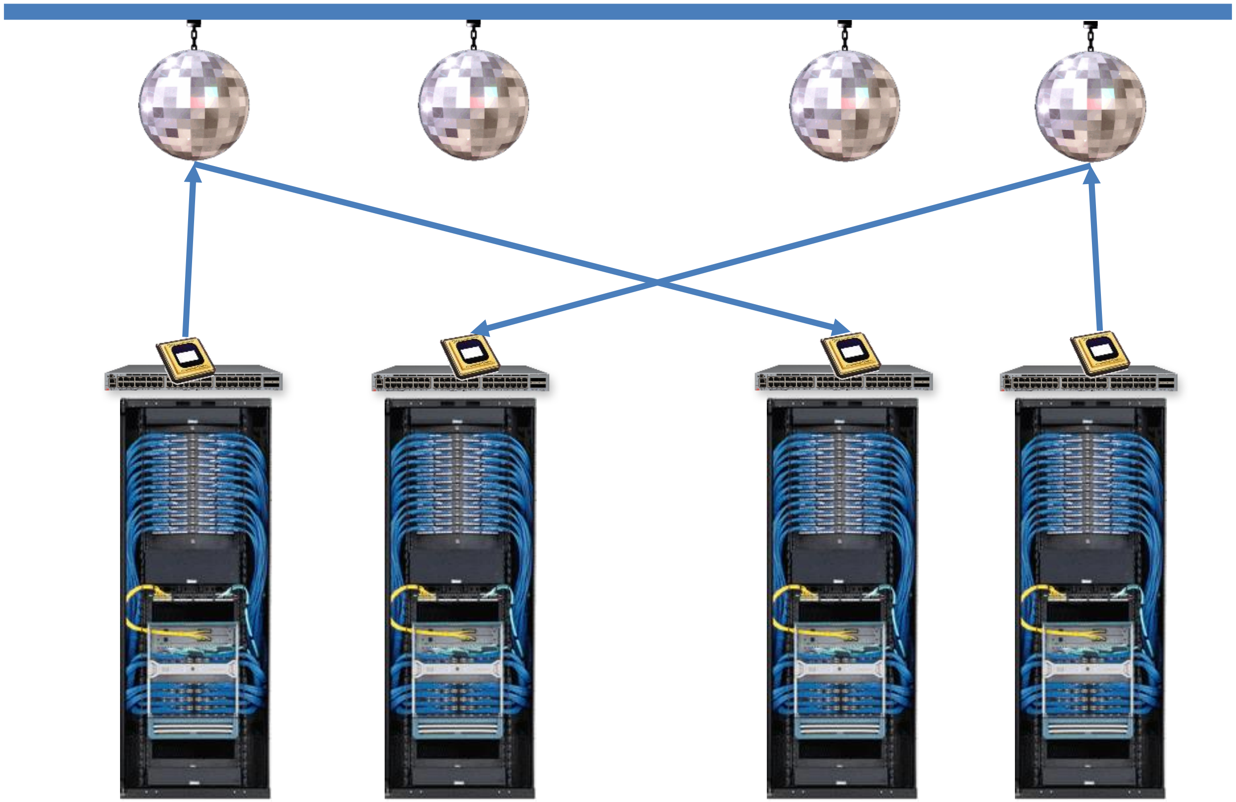
Complex **network services**, e.g., service chains

Even topology **reconfiguration**

Flexibility of communication networks

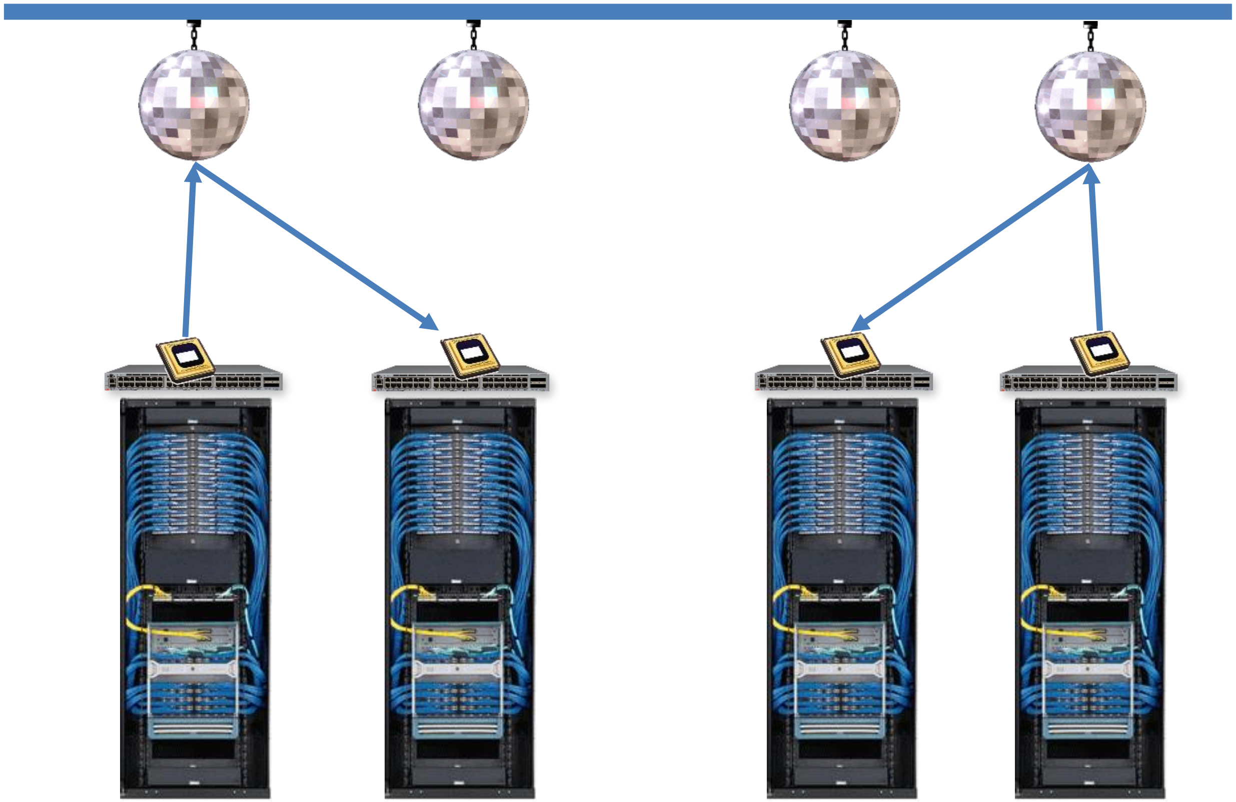
Example: Free-Space Optics (*ProjecToR*)

t=1



Example: Free-Space Optics (*ProjecToR*)

t=2



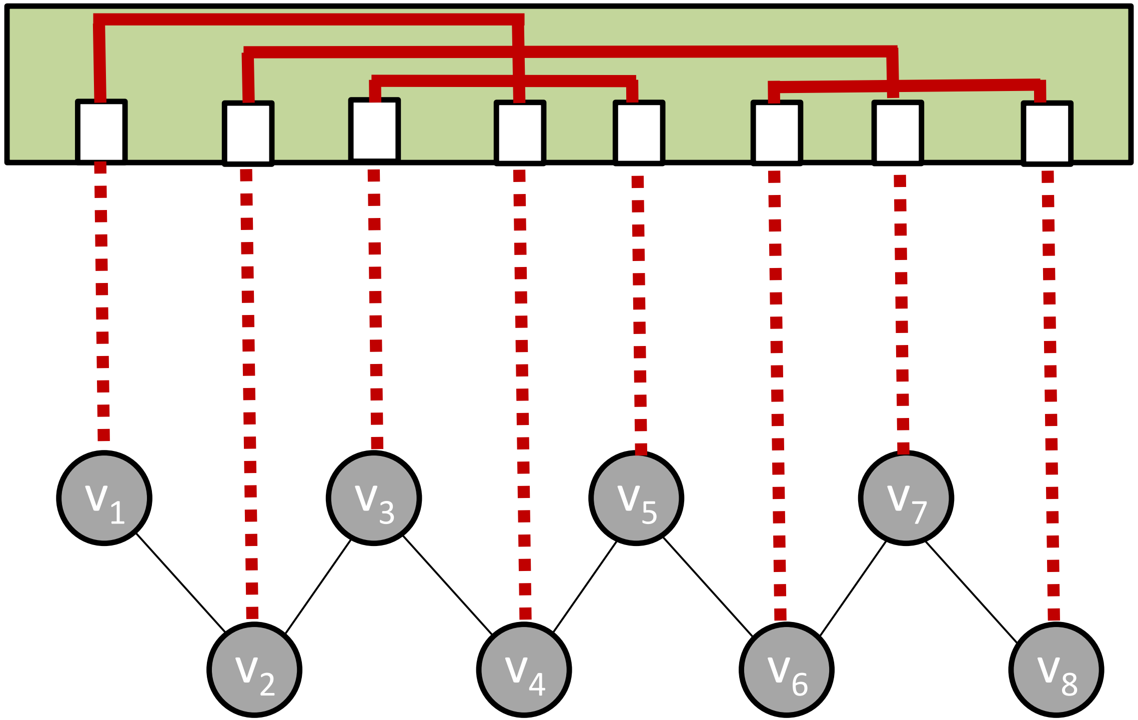
Example: Reconfigurable Optical Switches (*Helios*, *c-Through*, etc.)

Matching!

Dynamic topology:
optical switch
(e.g. matching)

t=1

Static topology:
electric



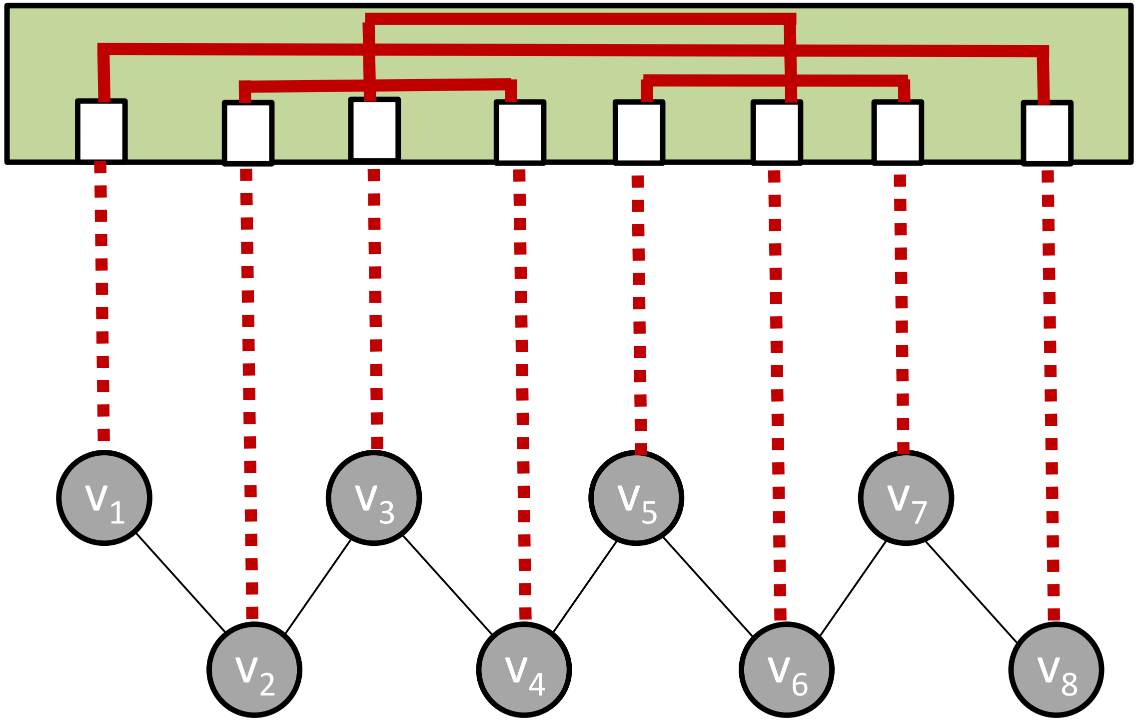
Example: Reconfigurable Optical Switches (*Helios*, *c-Through*, etc.)

Matching!

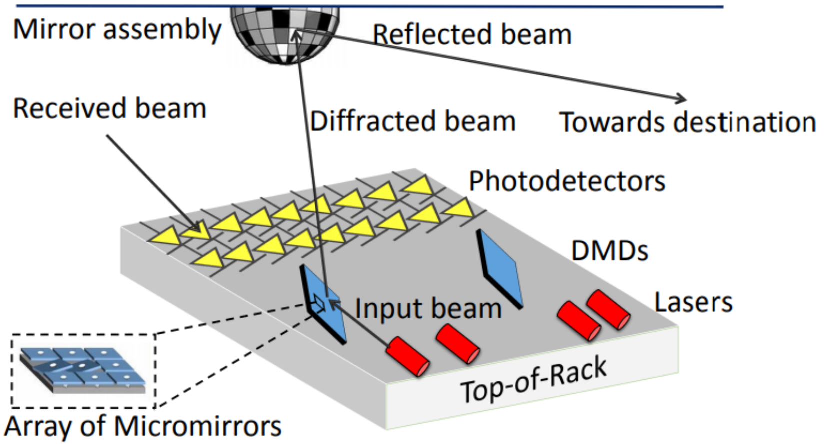
Dynamic topology: optical switch (e.g. matching)

t=2

Static topology: electric



Much Technology



Free-Space Optics

- Ghobadi et al., "***Projector***: Agile reconfigurable data center interconnect," SIGCOMM 2016.
- Hamedazimi et al. "***Firefly***: A reconfigurable wireless data center fabric using free-space optics," CCR 2014.

Optical Circuit Switches

- Farrington et al. "***Helios***: a hybrid electrical/optical switch architecture for modular data centers," CCR 2010.
- Mellette et al. "***Rotornet***: A scalable, low-complexity, optical datacenter network," SIGCOMM 2017.
- Farrington et al. "Integrating microsecond circuit switching into the data center," SIGCOMM 2013.
- Liu et al. "Circuit switching under the radar with reactor.," NSDI 2014

Movable Antennas

- Halperin et al. "Augmenting data center networks with multi-gigabit wireless links," SIGCOMM 2011.

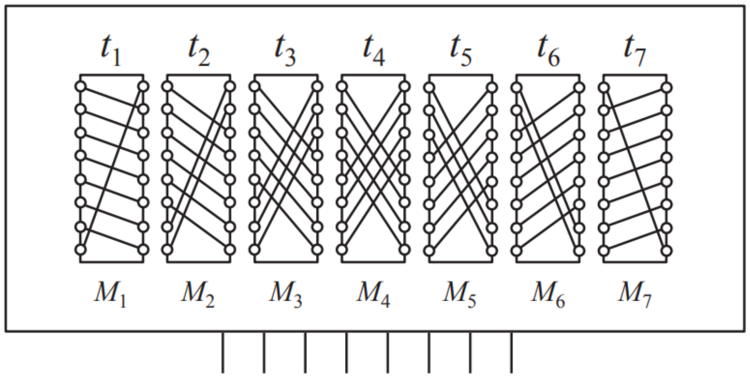
60GHz Wireless Communication

- Zhou et al. "Mirror mirror on the ceiling: Flexible wireless links for data centers," CCR 2012.
- Kandula et al. "***Flyways*** to de-congest data center networks," 2009.

Etc.!

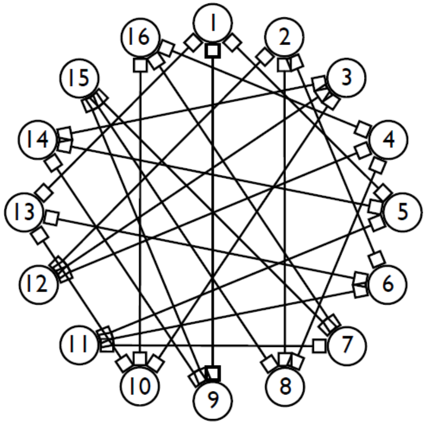
Also for the WAN!

Rotor switch



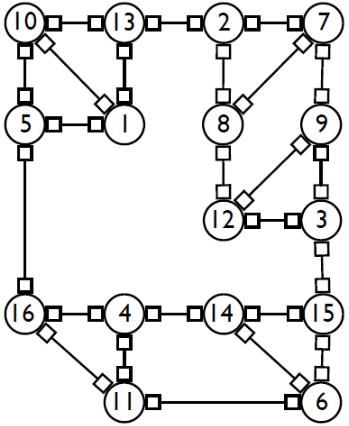
Resulting Vision: Exploiting *Locality of Demand*
Demand-Aware and Self-Adjusting Networks

Oblivious



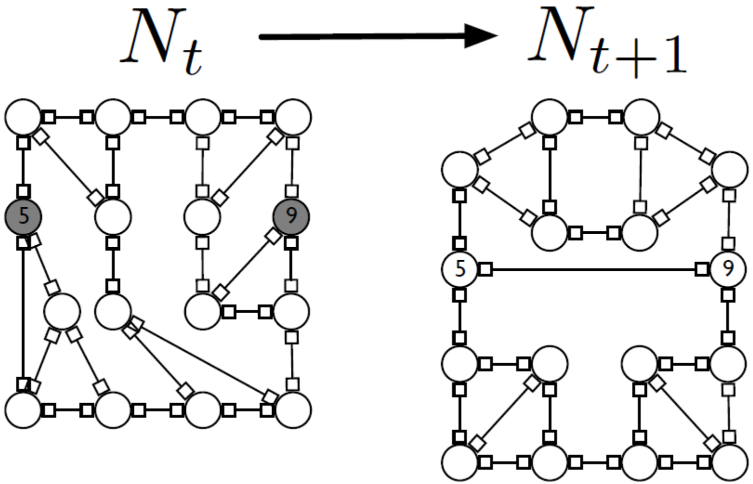
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**

SAN



Exploit **temporal locality** as well

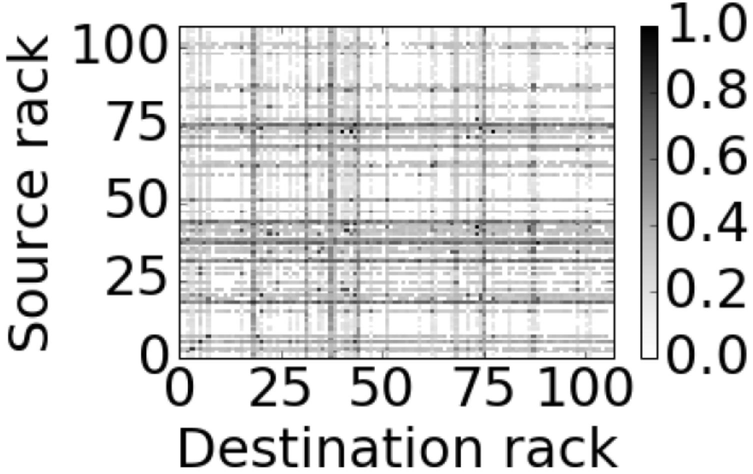
How much does it help?

How much does it help?

Depends on the "**entropy**"!
The less entropy, e.g., the shorter the routes!

Motivation:
Much Structure = Little Entropy

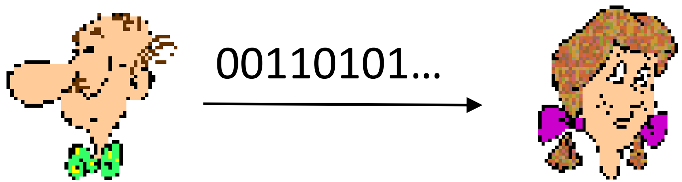
Heatmap rack-to-rack traffic:



ProjecToR @ SIGCOMM 2016

An Analogy to Coding

„Coming to MIR^3?“



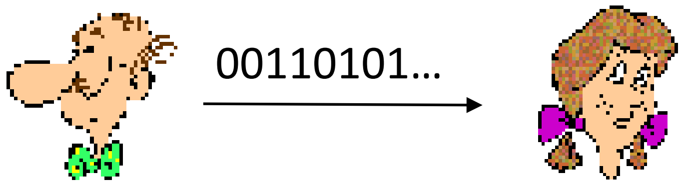
structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

An Analogy to Coding

„Coming to MIR^3?“



structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

Requires **statistics!**

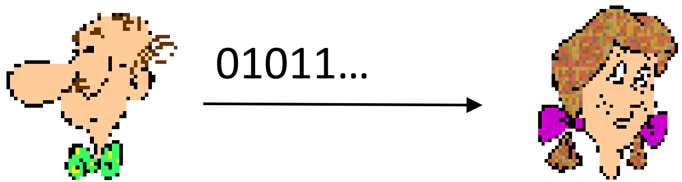
static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

An Analogy to Coding

„Coming to MIR^3?“



structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

Requires **statistics!**

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

DAN!

An Analogy to Coding

„Coming to MIR^3?“

Better or worse?

structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static Demand-Aware Nets

static Huffman:
1, 01, 001, 000

dynamic / unknown

dynamic Demand-Aware Nets

dynamic Huffman codes

An Analogy to Coding

„Coming to MIR^3?“

Better or worse?

structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

dynamic / unknown

dynamic
Demand-Aware Nets

dynamic
Huffman codes

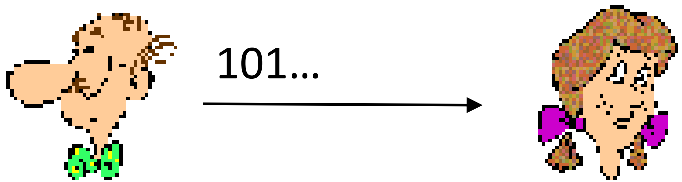
It depends:

Can exploit
temporal locality!

But: No statistics:
online!

An Analogy to Coding

„Coming to MIR^3?“



structure: static / future demand: unknown

worst case network: Full BW

worst case coding: 00, 01, 10, 11

static / known

static Demand-Aware Nets

static Huffman: 1, 01, 001, 000

dynamic / unknown

dynamic Demand-Aware Nets

dynamic Huffman codes

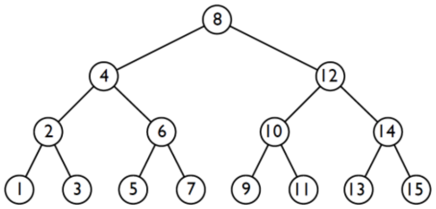
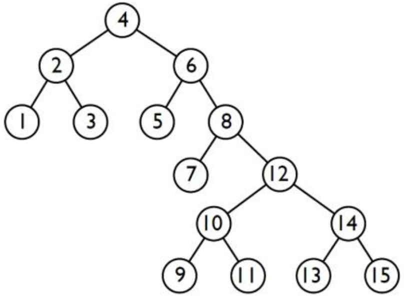
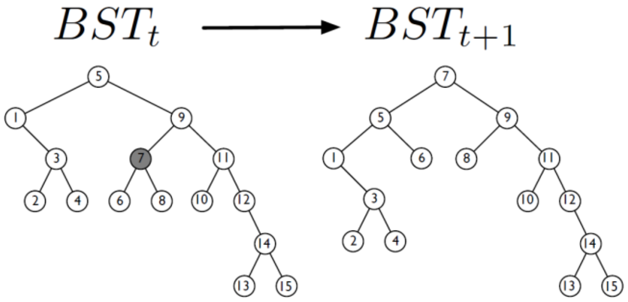
DAN!

Can exploit *spatial locality*!

SAN!

Additionally exploit *temporal locality*!

Analogy to Datastructures, e.g., BST

Oblivious BST	Demand-Aware (aka „Biased“ BST)	Self-Adjusting e.g., splay trees
		
Lookup ***O(log n)***	Exploit **spatial locality**: e.g., if only log(n) elements accessed lookup ***O(loglog n)***	Exploit **temporal locality** as well, e.g.,: ***O(1)***

Conclusion

- **Complexity and flexibilities:** The case of algorithmic approaches and automation

- Formal methods can be efficient! Case study **What-if Analysis** for MPLS and SR
  - Other examples: verified packet ***parsers***, verified „***self-driving*** networks“, consistent network ***updates***, …

- The next frontier: **topological flexibility**
  - Requires new algorithms: largely unexplored

Thank you!
Question?

Further Reading

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.
Taking Control of SDN-based Cloud Systems via the Data Plane (Best Paper Award)
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
ACM Symposium on SDN Research (**SOSR**), Los Angeles, California, USA, March 2018.
Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks
Chen Avin and Stefan Schmid.
ArXiv Technical Report, July 2018.
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.
Online Balanced Repartitioning
Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid.
30th International Symposium on Distributed Computing (**DISC**), Paris, France, September 2016.
rDAN: Toward Robust Demand-Aware Network Designs
Chen Avin, Alexandr Hercules, Andreas Loukas, and Stefan Schmid.
Information Processing Letters (**IPL**), Elsevier, 2018.
SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.
Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.
Charting the Complexity Landscape of Virtual Network Embeddings
Matthias Rost and Stefan Schmid. **IFIP Networking**, Zurich, Switzerland, May 2018.