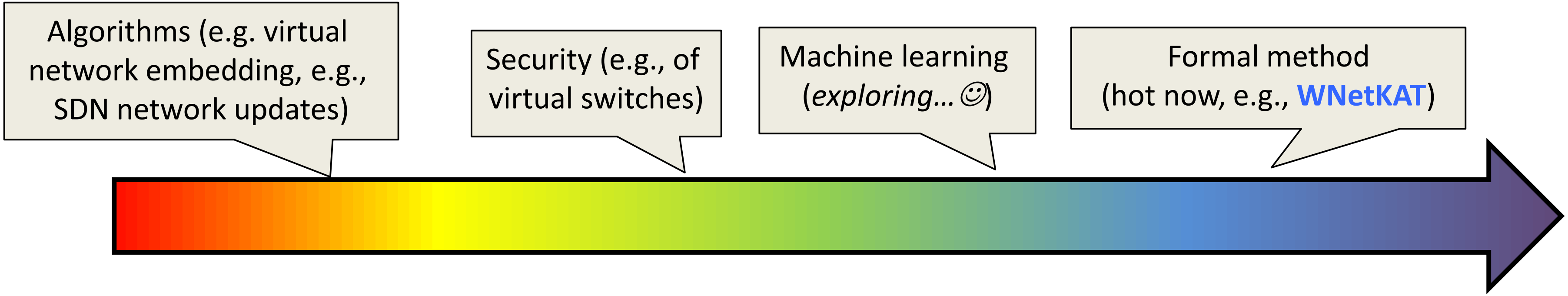
From Self-Verifying to Self-Adjusting Networks

Stefan Schmid (University of Vienna, Austria)

From Self-Verifying to Self-Adjusting Networks

Stefan Schmid *et al.*, most importantly Jiri Srba (Aalborg University, Denmark) and Chen Avin (BGU, Israel)

Spectrum of Networking Research at Uni Vienna



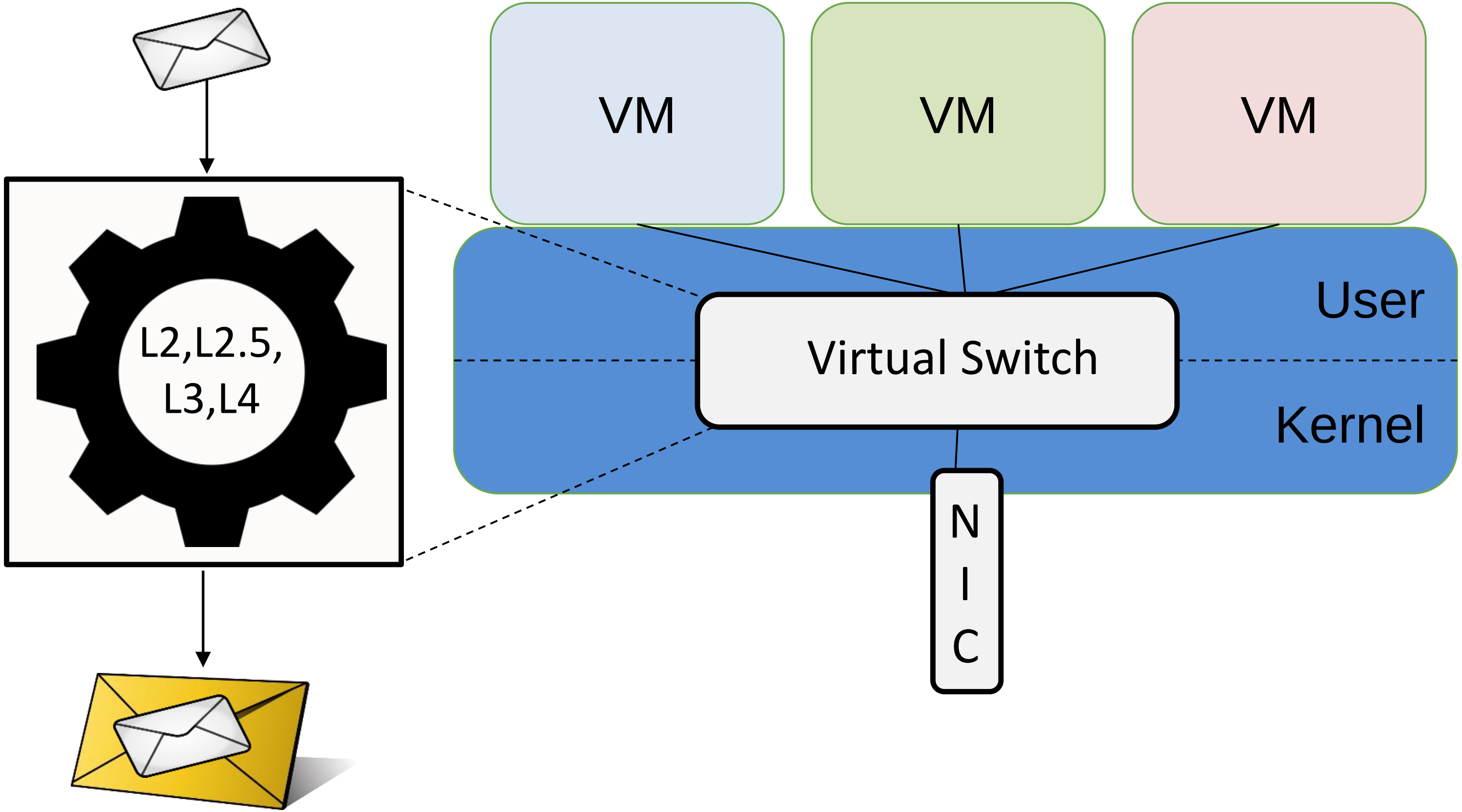
Looking for PhD students, Summer interns, project partners, …

E.g., Security Analysis of OVS

ACM SOSR 2018 Best Paper

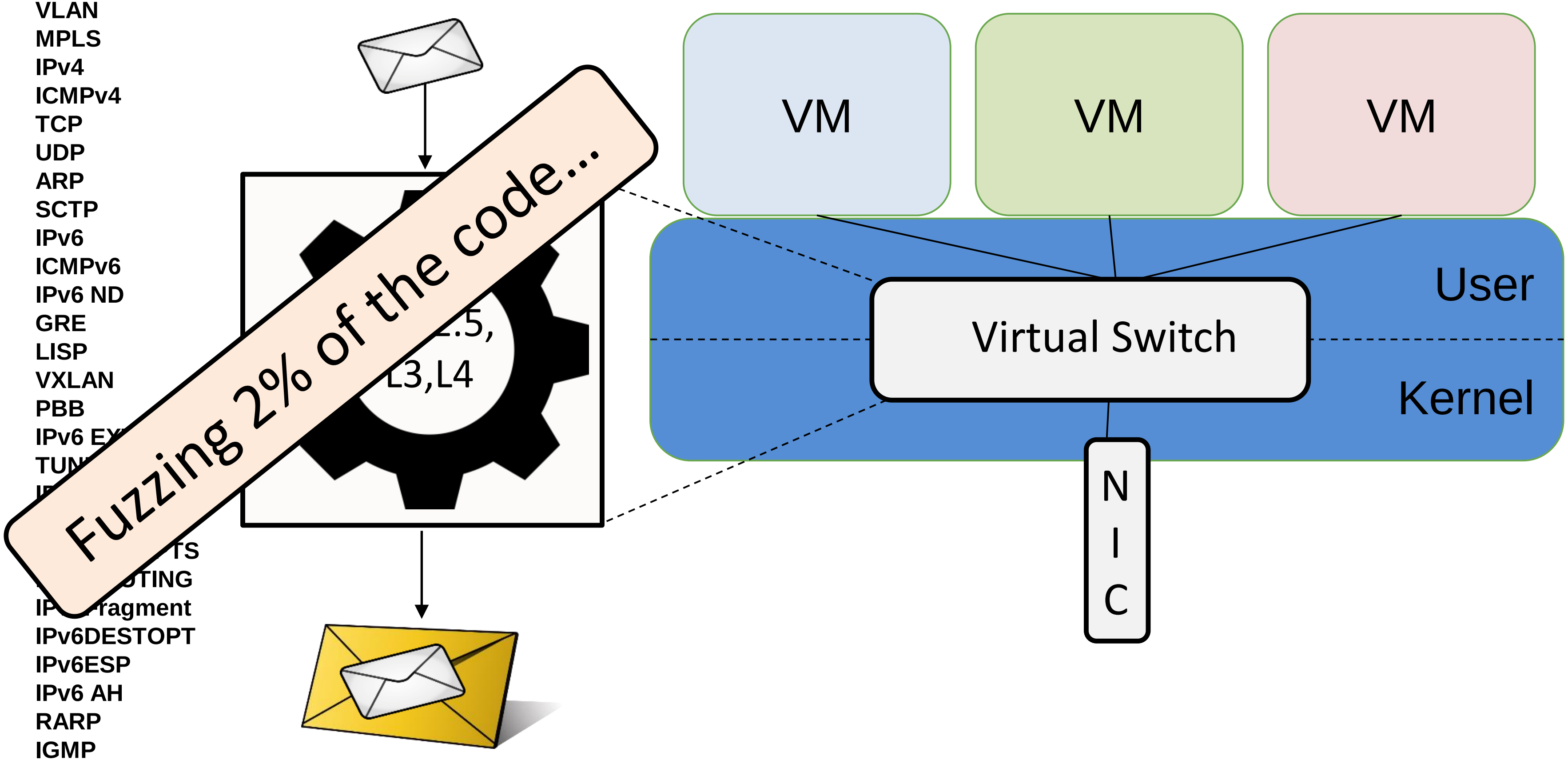
Virtual Switches are Complex, e.g.: (Unified) Packet Parsing

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

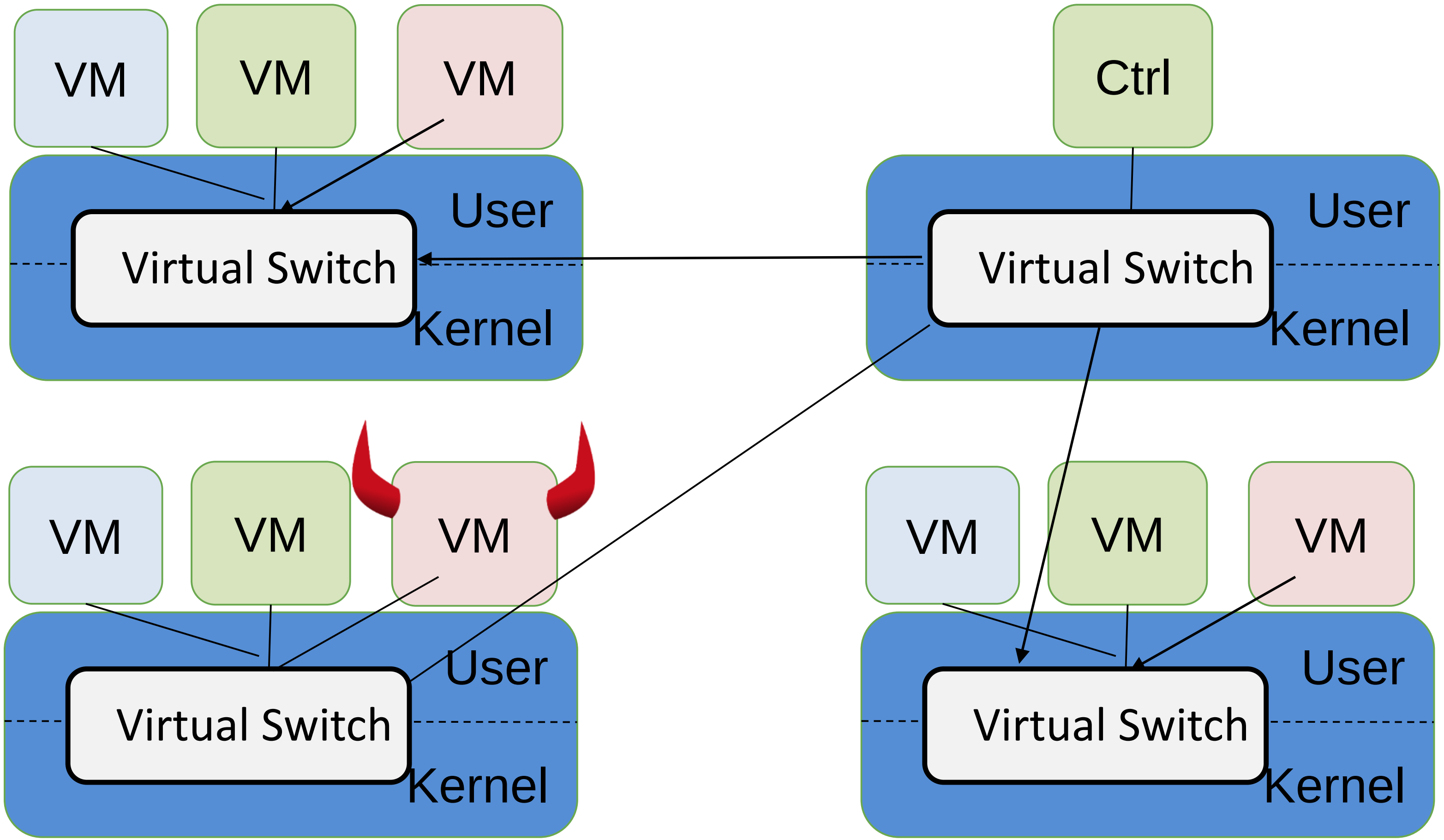


Virtual Switches are Complex, e.g.: (Unified) Packet Parsing

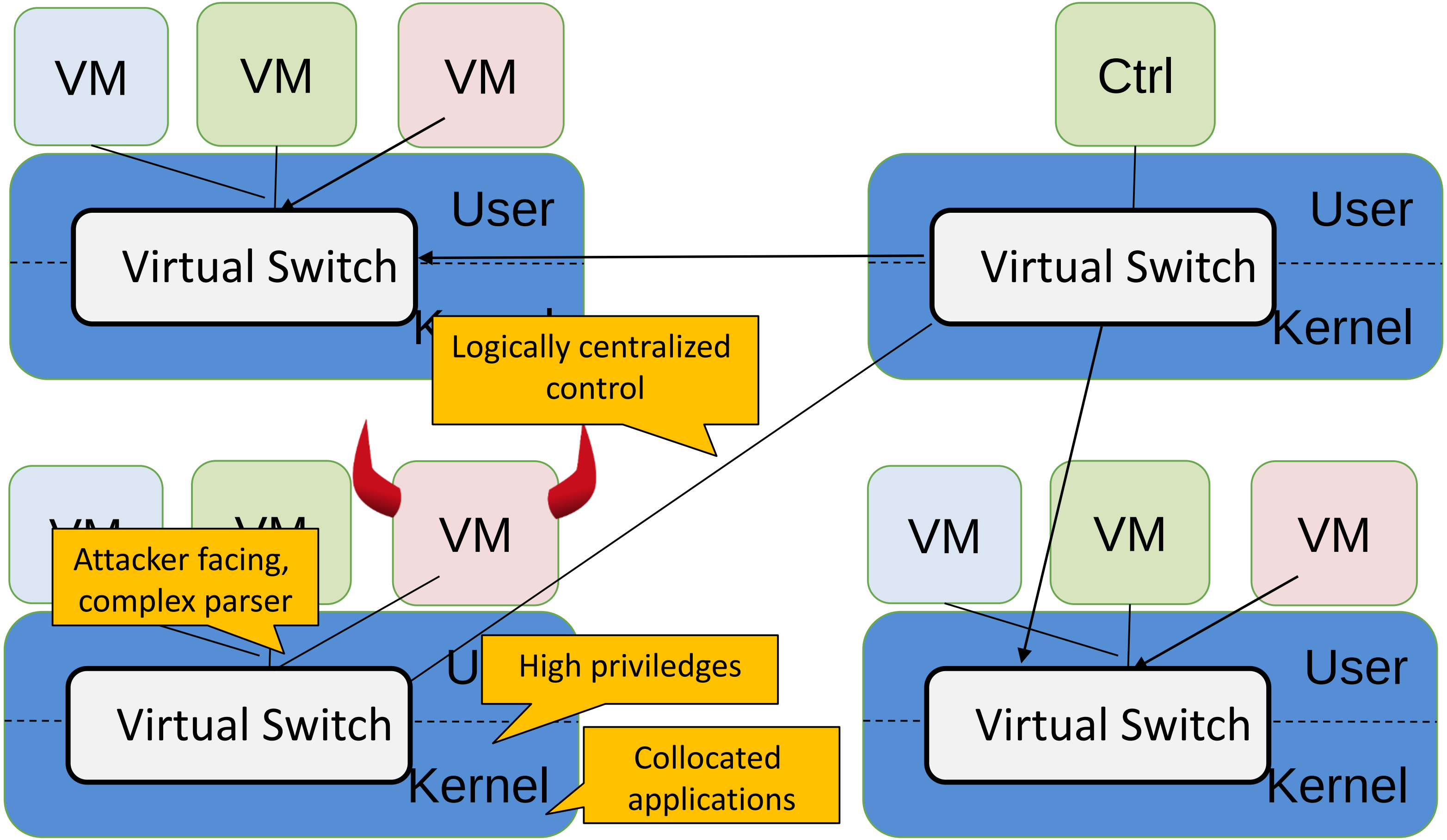
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EX
TUN
IP
TS
UTING
IP ragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



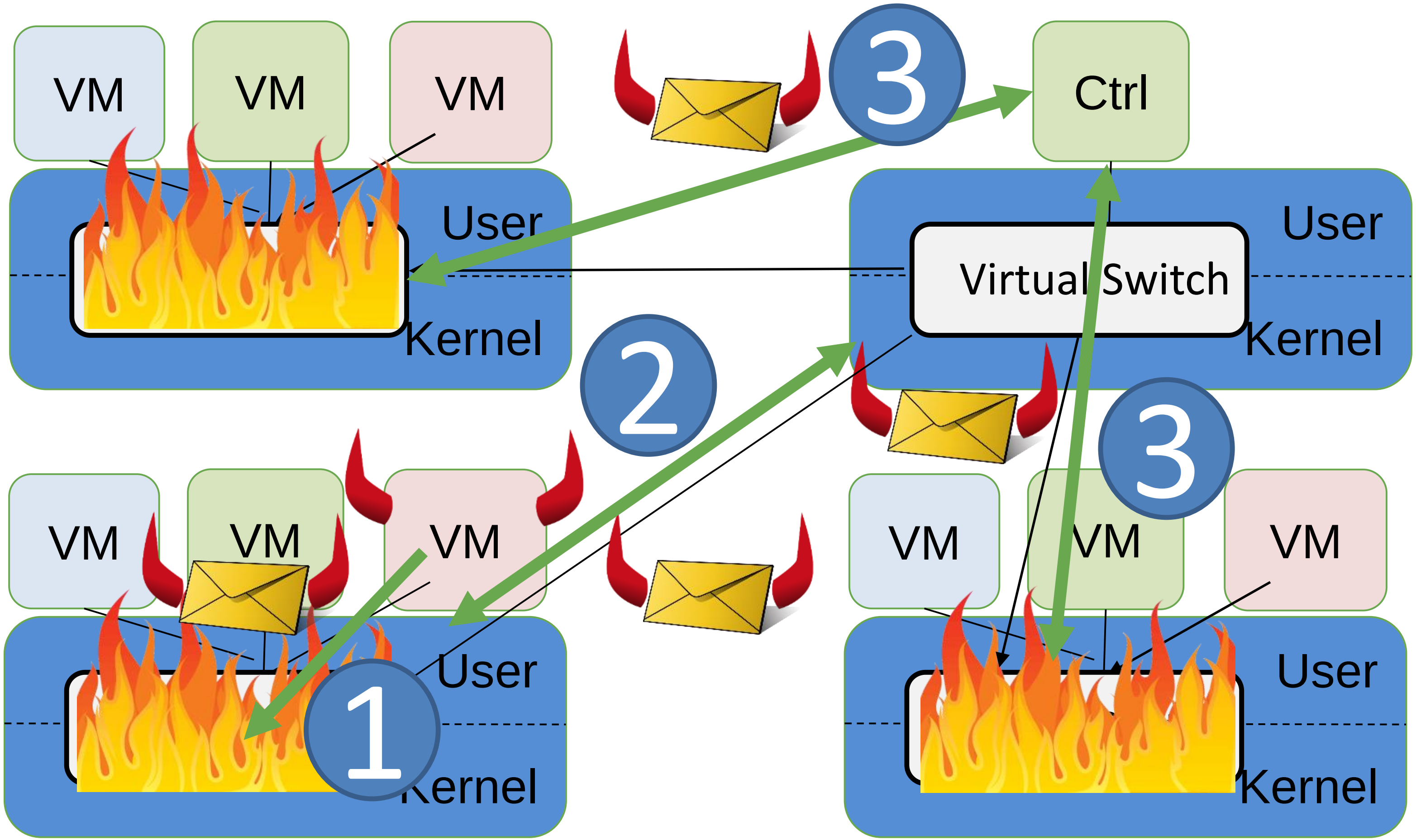
Compromising the Cloud



Compromising the Cloud



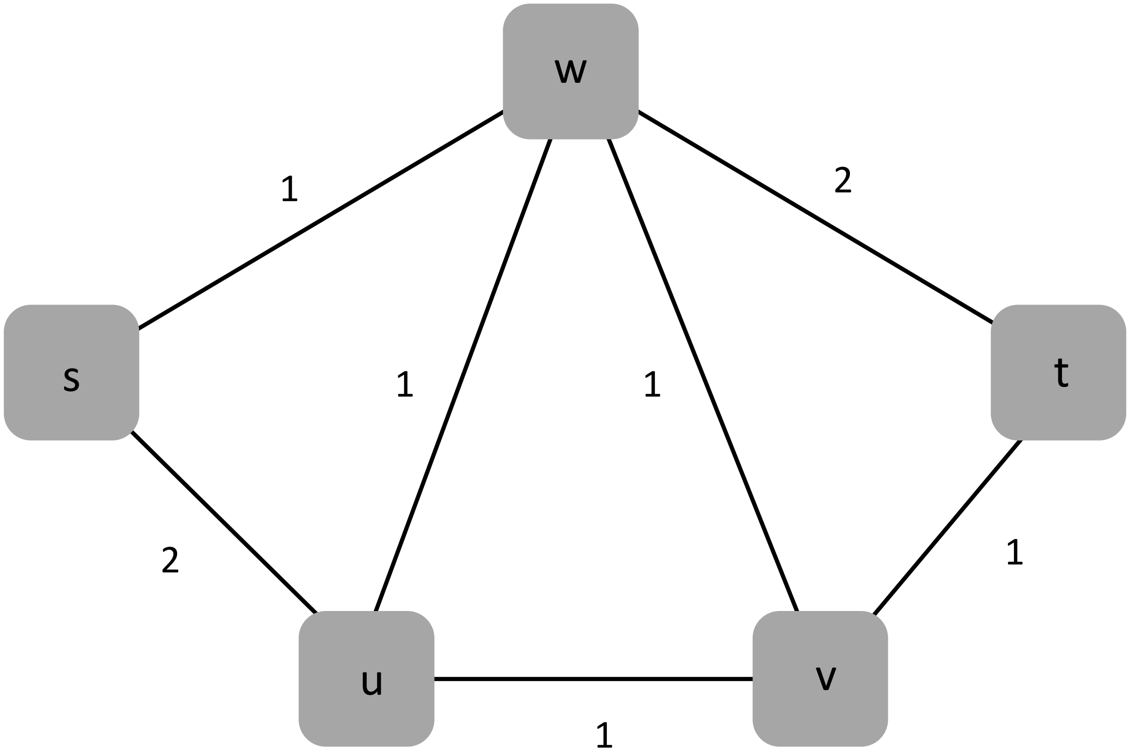
Compromising the Cloud



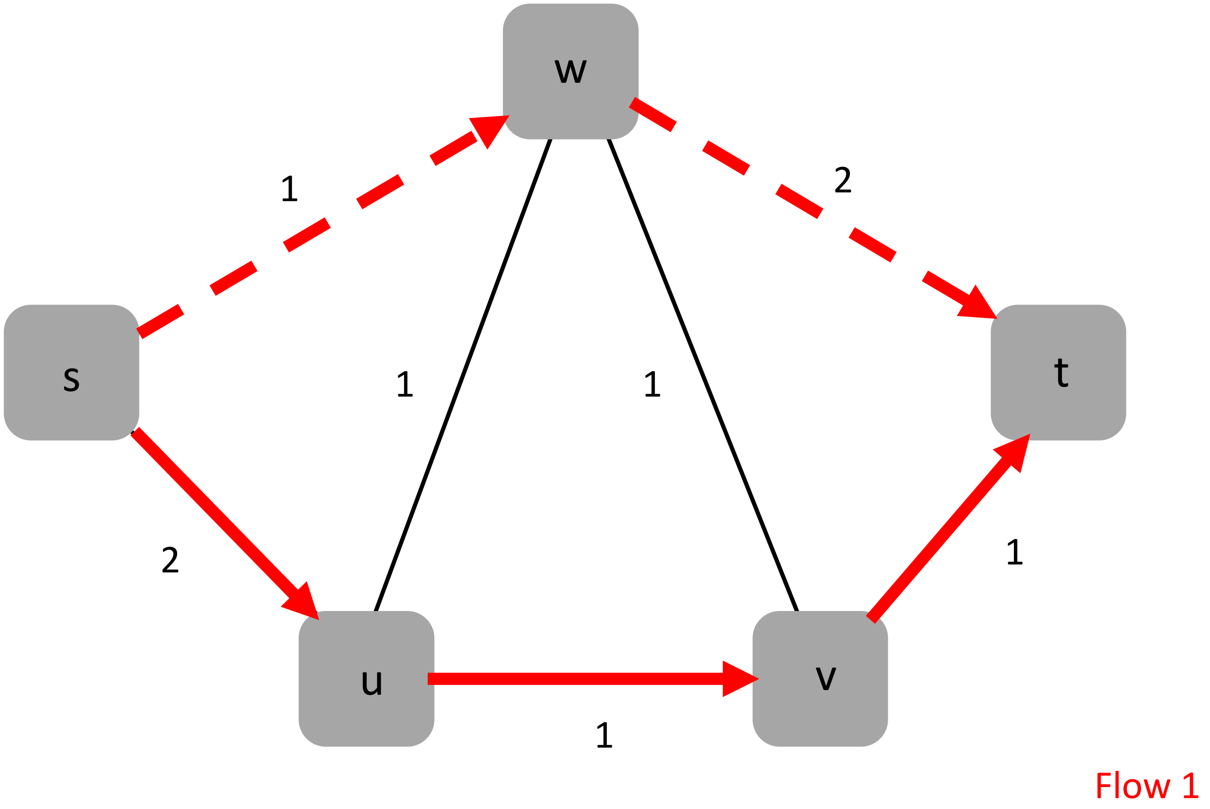
E.g., Consistent Flow Rerouting for SDN

PODC 2015, SIGMETRICS 2016, ICALP 2018

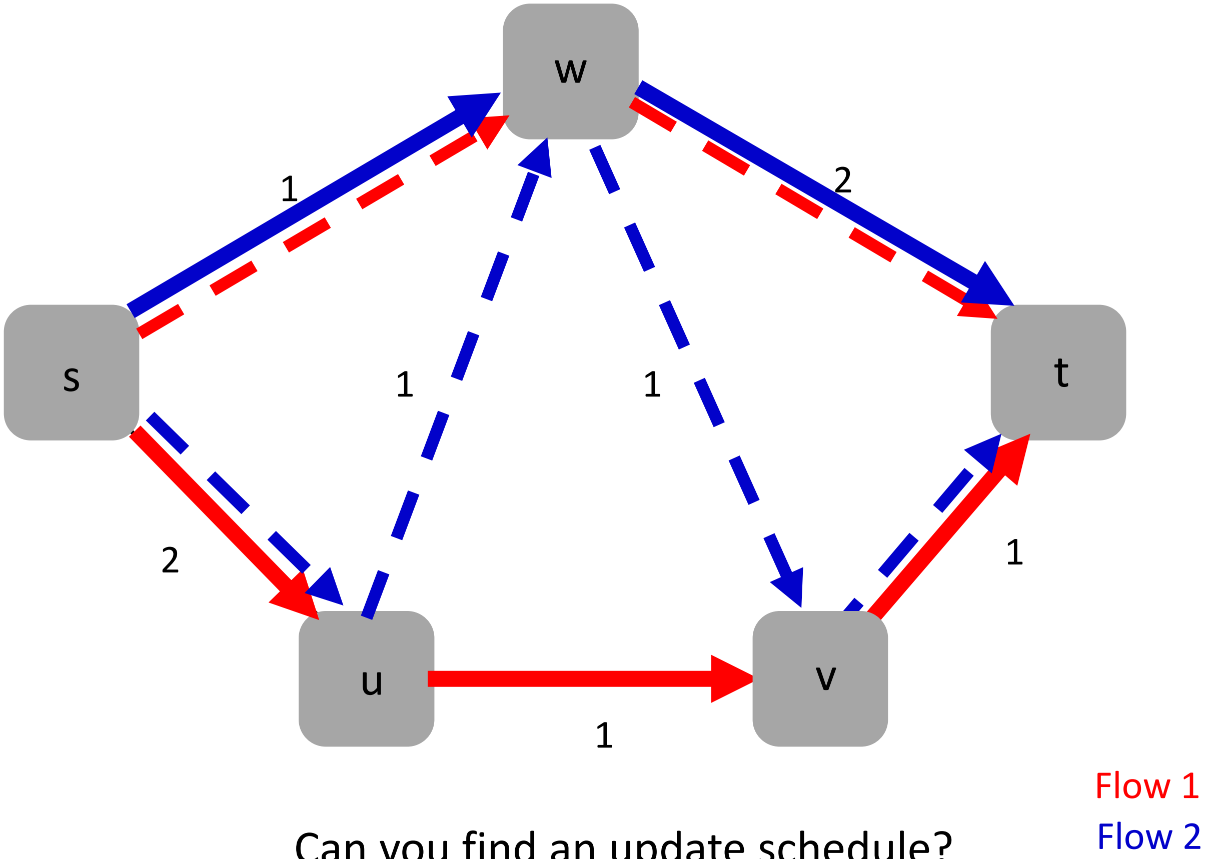
Consistent Network Updates



Consistent Network Updates



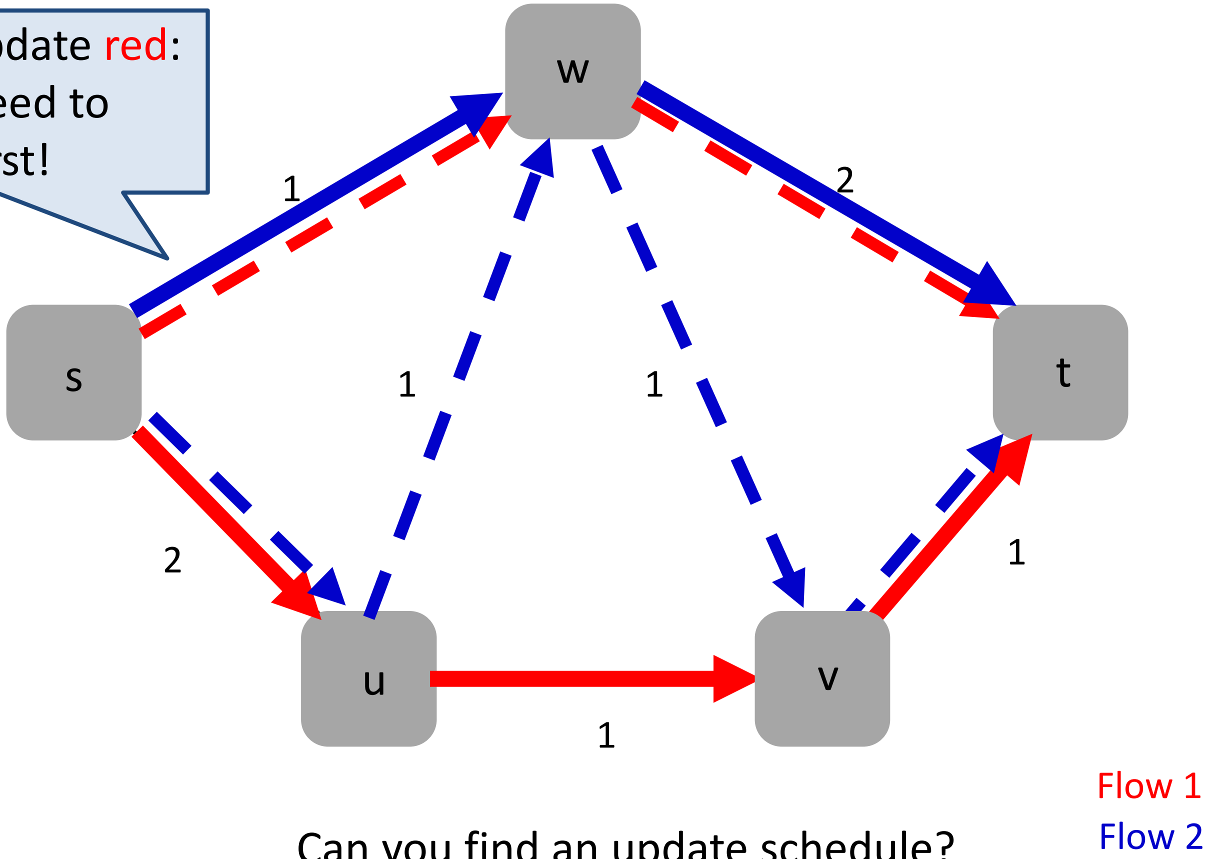
Consistent Network Updates



Can you find an update schedule?

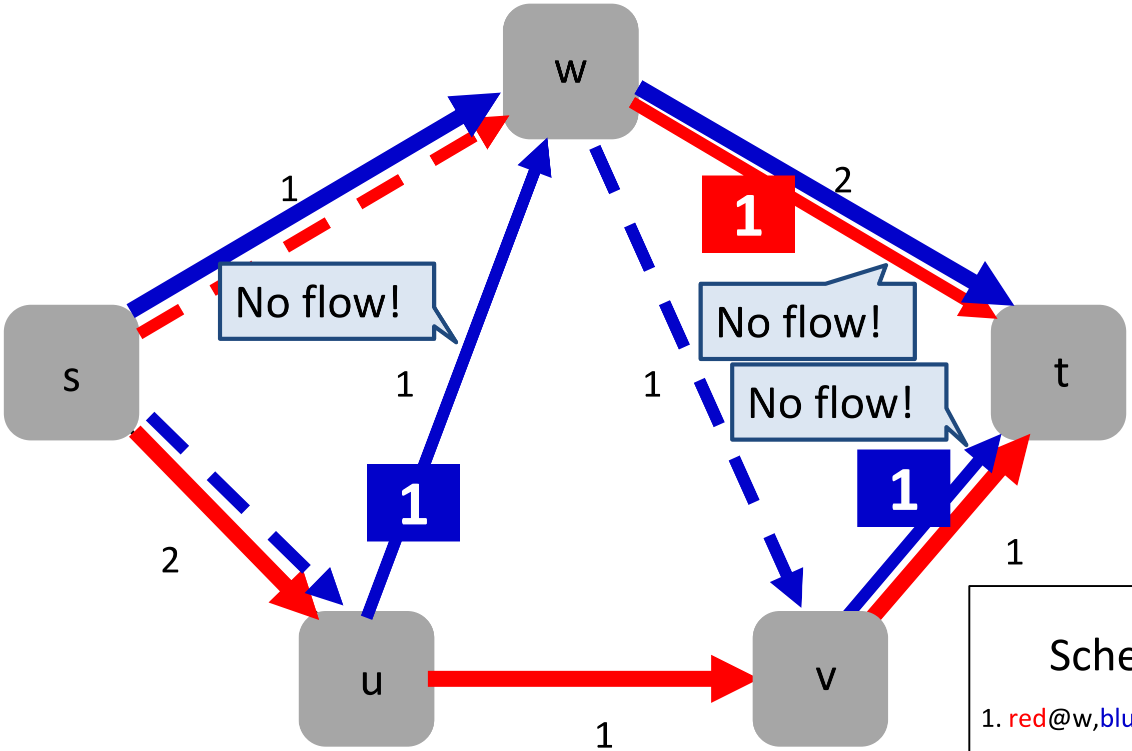
Consistent Network Updates

e.g., cannot update red: congestion! Need to update blue first!



Can you find an update schedule?

Consistent Network Updates

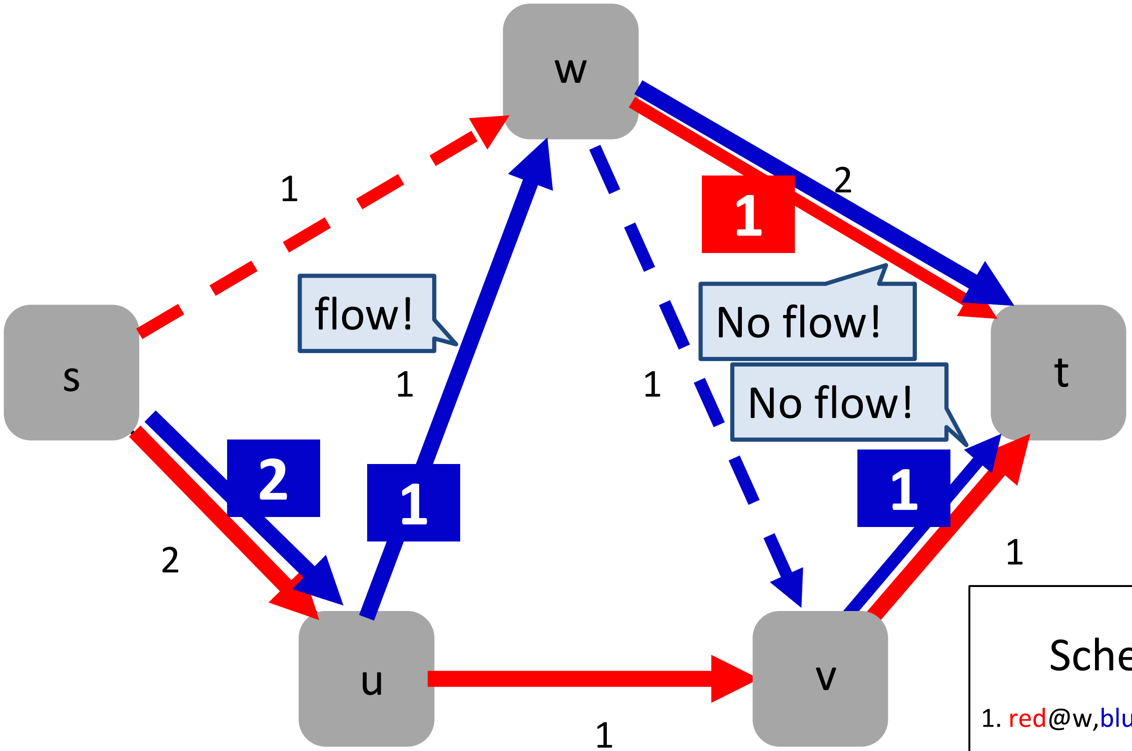


Round 1: prepare

Schedule:

1. red@w,blue@u,blue@v

Consistent Network Updates

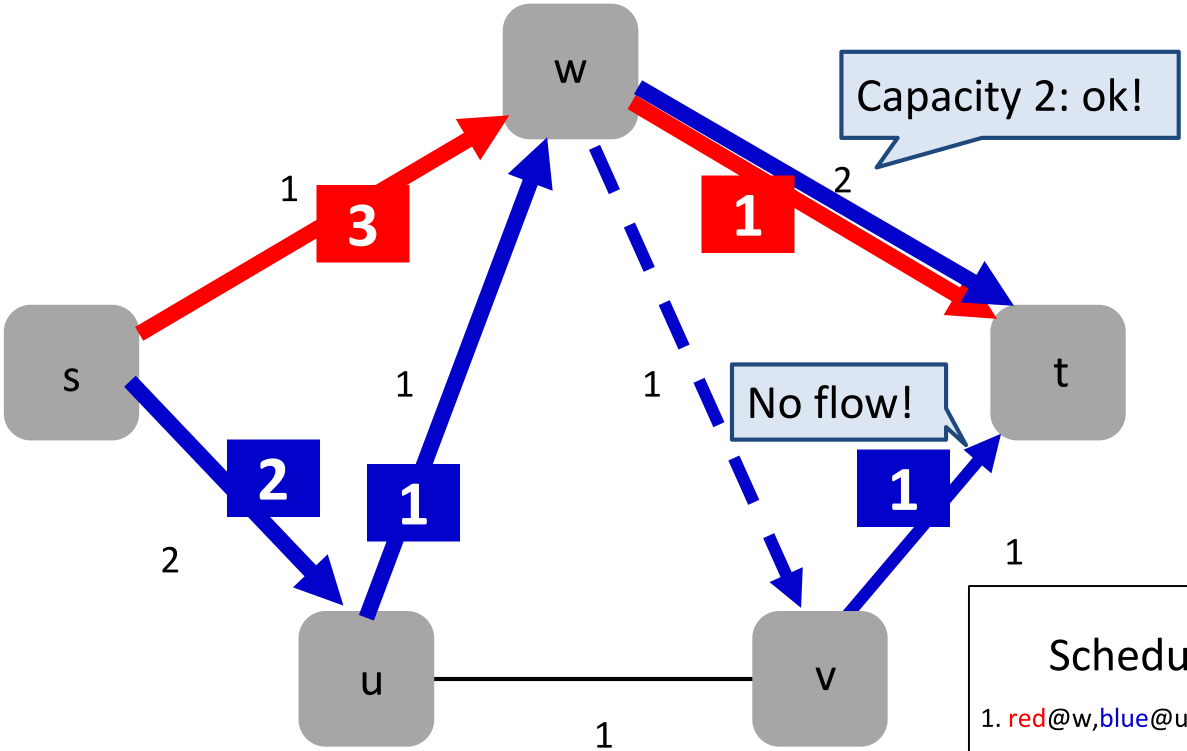


Round 2

Schedule:

1. red@w,blue@u,blue@v
2. blue@s

Consistent Network Updates

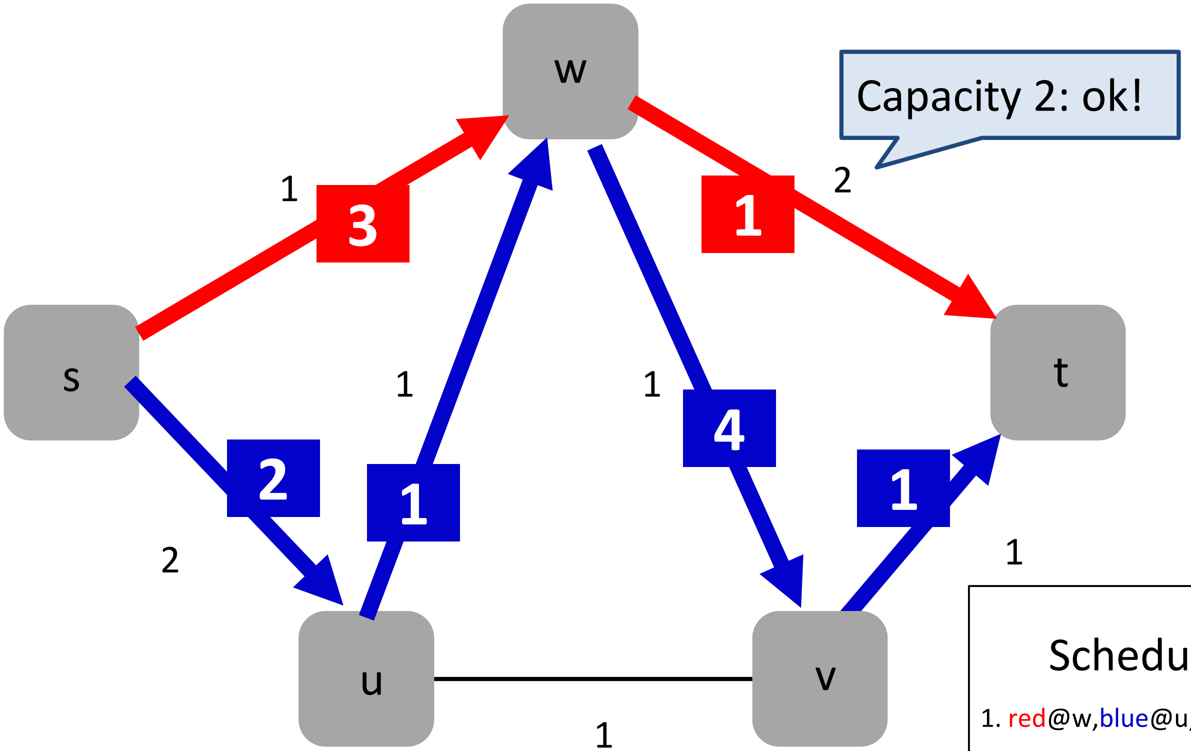


Round 3

Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s

Consistent Network Updates

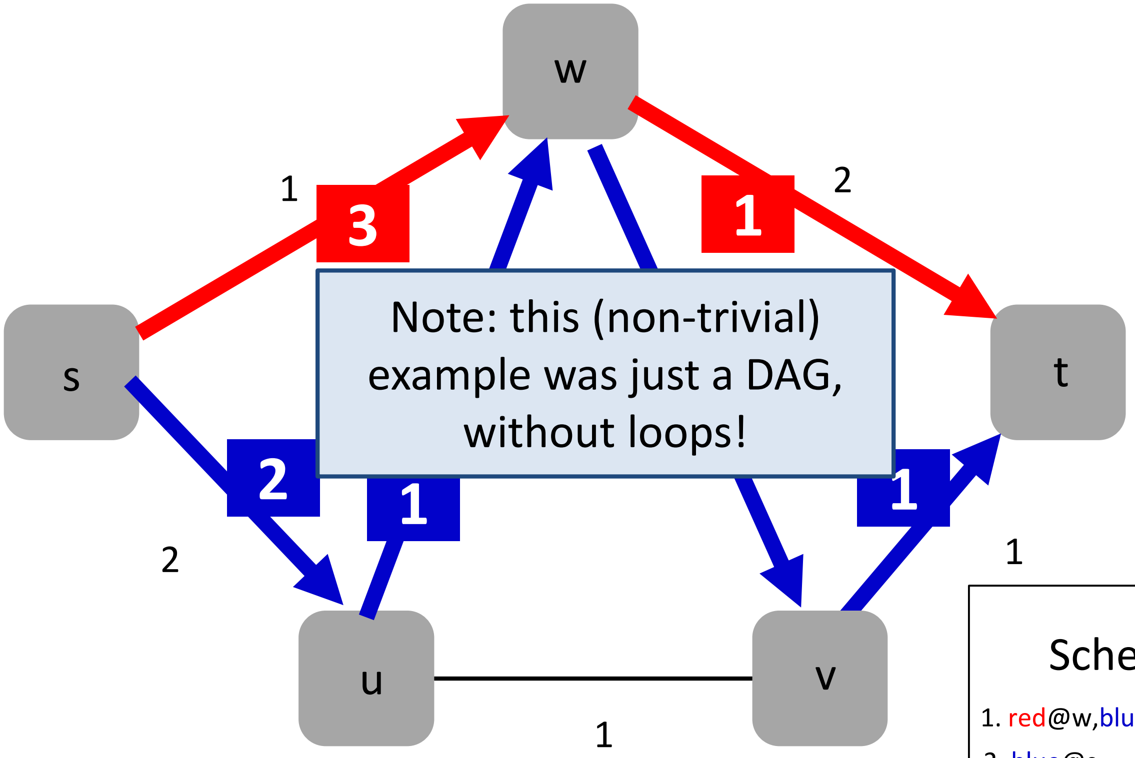


Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

Round 4

Consistent Network Updates



Note: this (non-trivial) example was just a DAG, without loops!

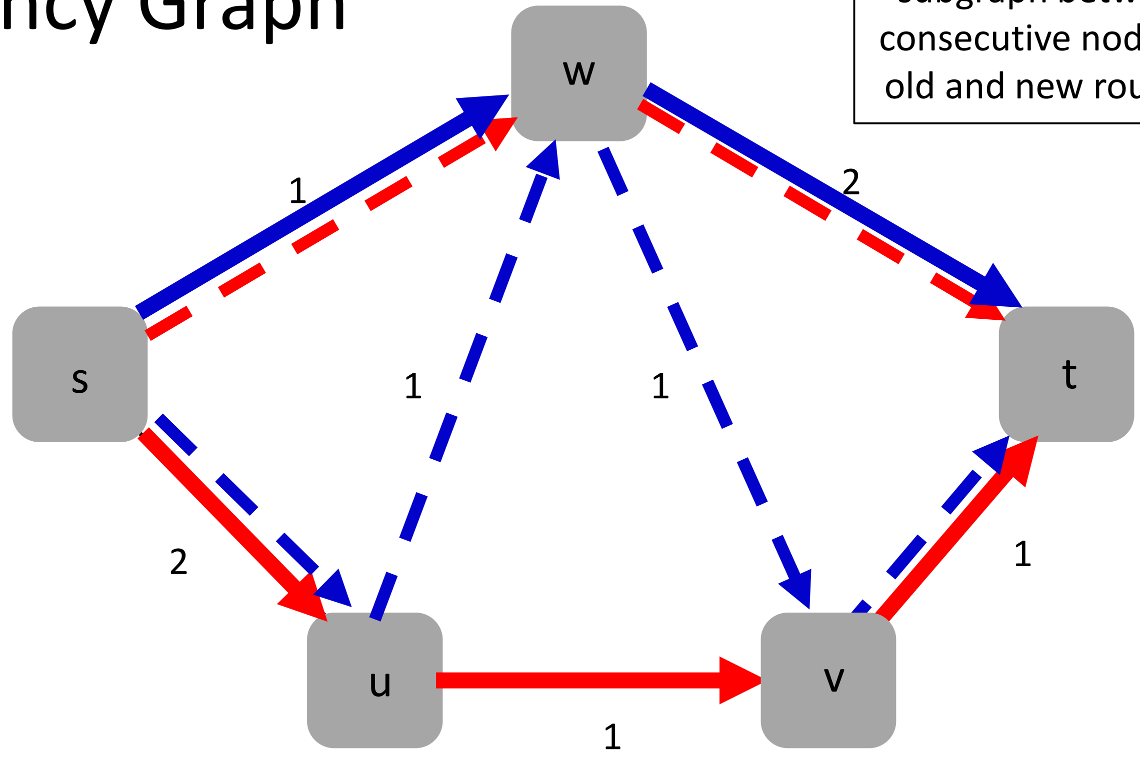
Round 4

Schedule:

1. red@w,blue@u,blue@v
2. blue@s
3. red@s
4. blue@w

Block Decomposition and Dependency Graph

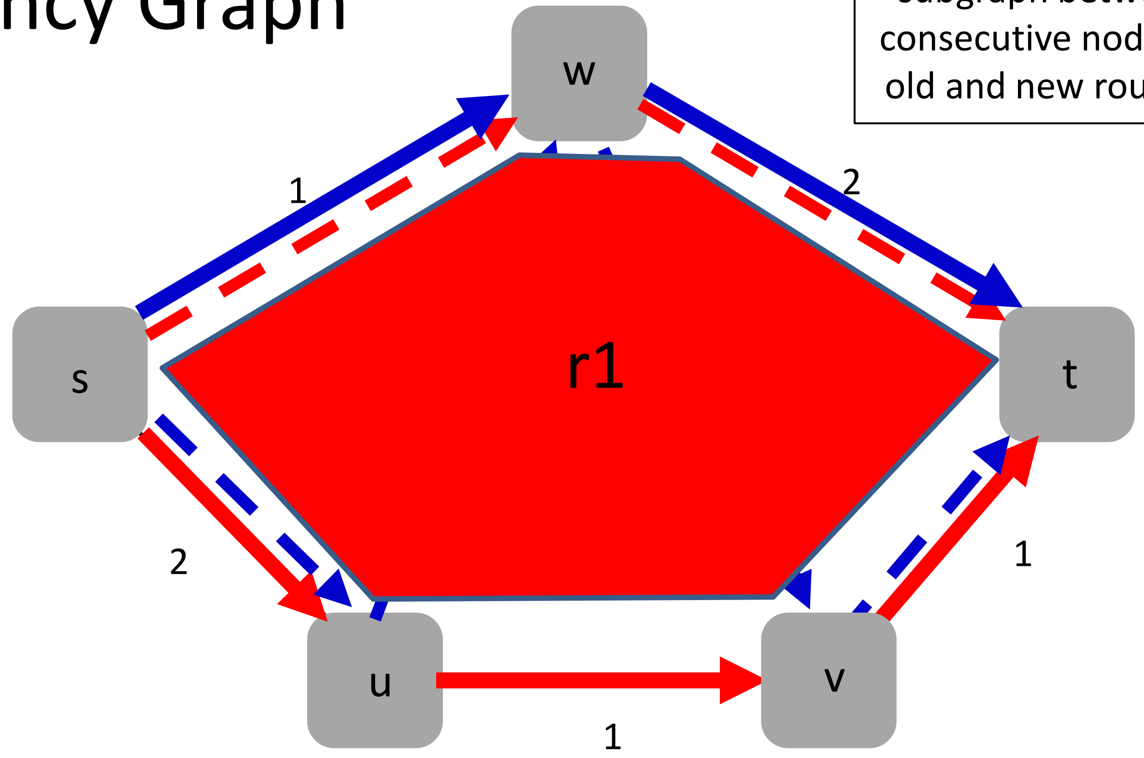
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Flow 1
Flow 2

Block Decomposition and Dependency Graph

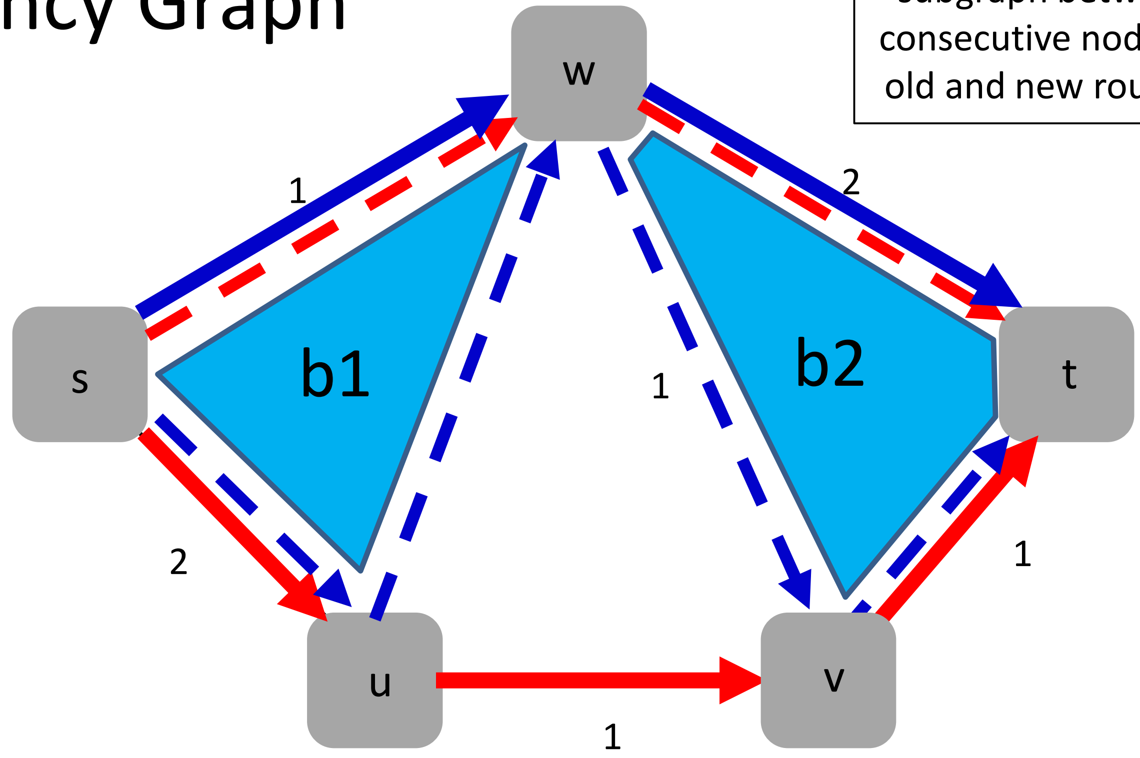
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Just one red block: r1

Block Decomposition and Dependency Graph

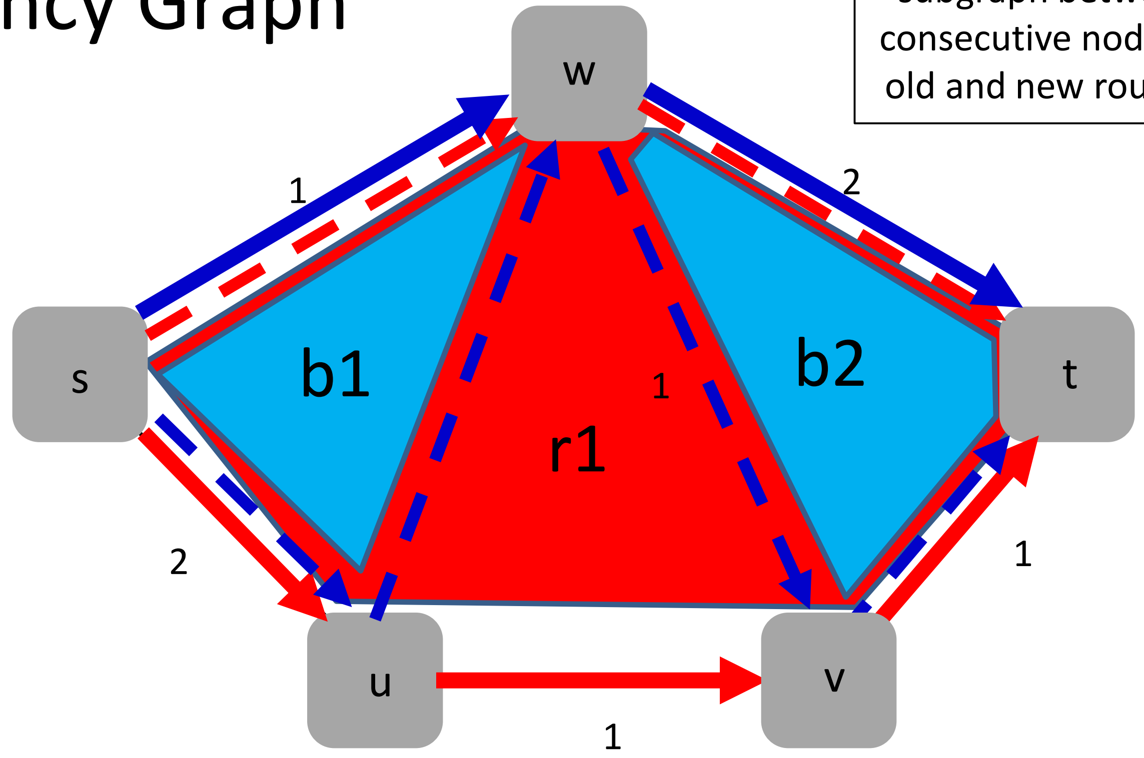
Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



Two blue blocks: b1 and b2

Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.



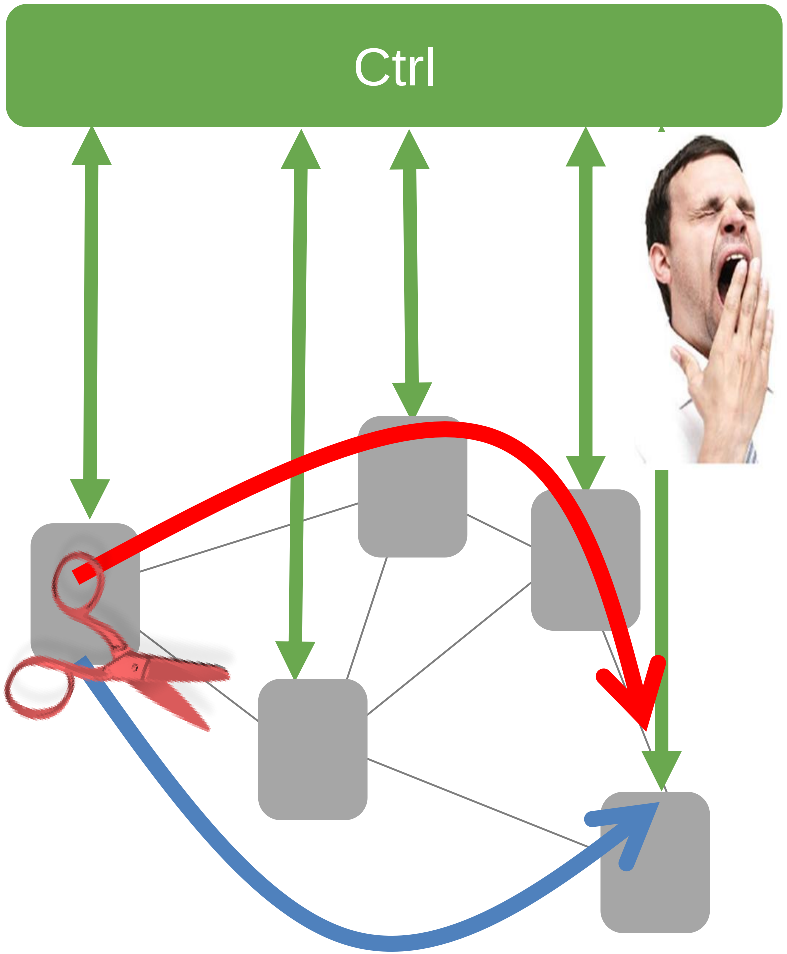
Dependencies: update b2 after r1 after b1.

E.g., following up on HUJI…

DSN 2017, CCR 2018

Local Fast Rerouting

- Failover without invoking control plane
- Perfect resiliency: ***Schapira et al.*** (INFOCOM, ICALP, etc.)
- But what about load? Related to symmetric block design theory (BIBDs) and **distributed computing without communication**!
  - Order in which to choose arborescences



Roadmap

- Networks are increasingly ***complex***: a case for **formal methods**?

- Networks are increasingly ***flexible***: a case for **self-adjusting networks**?

Managing Complex Networks is Hard for Humans

Human Errors

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.

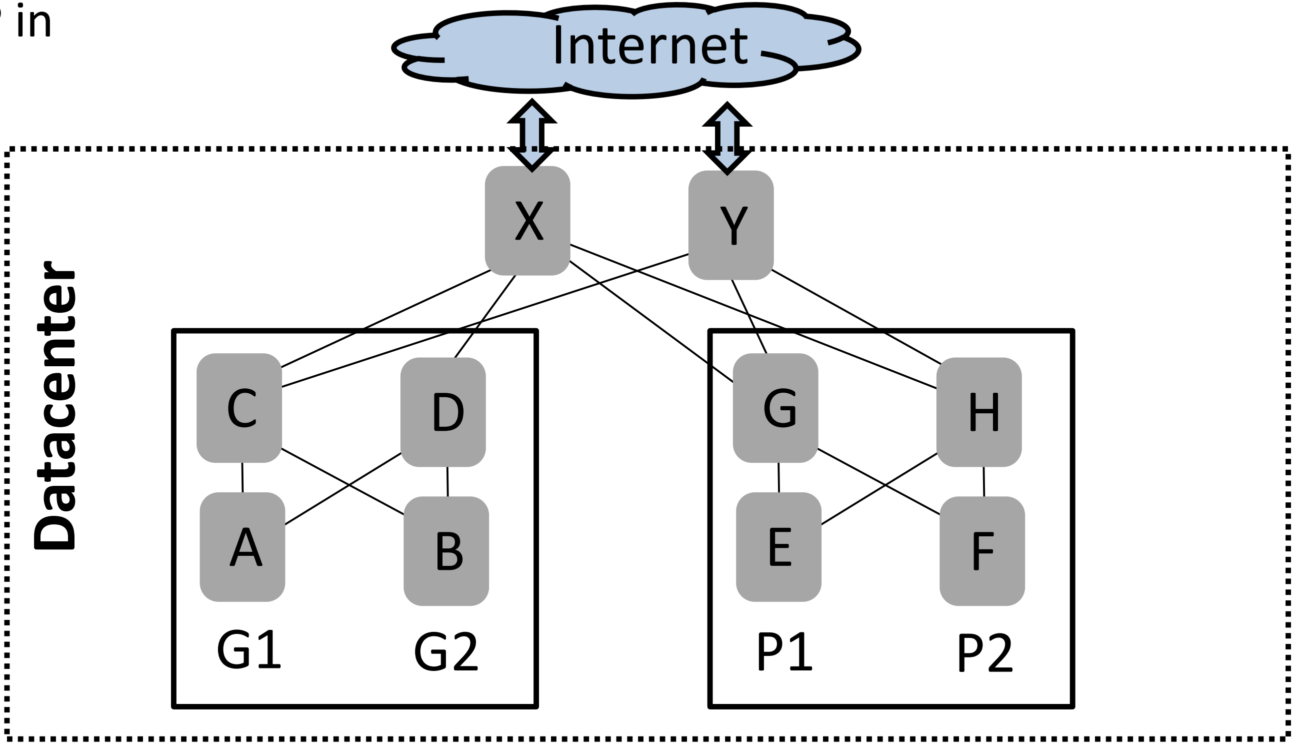
Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems

United Airlines

Credits: Nate Foster

Example: Keeping Track of (Flexible) Routes Under Failures

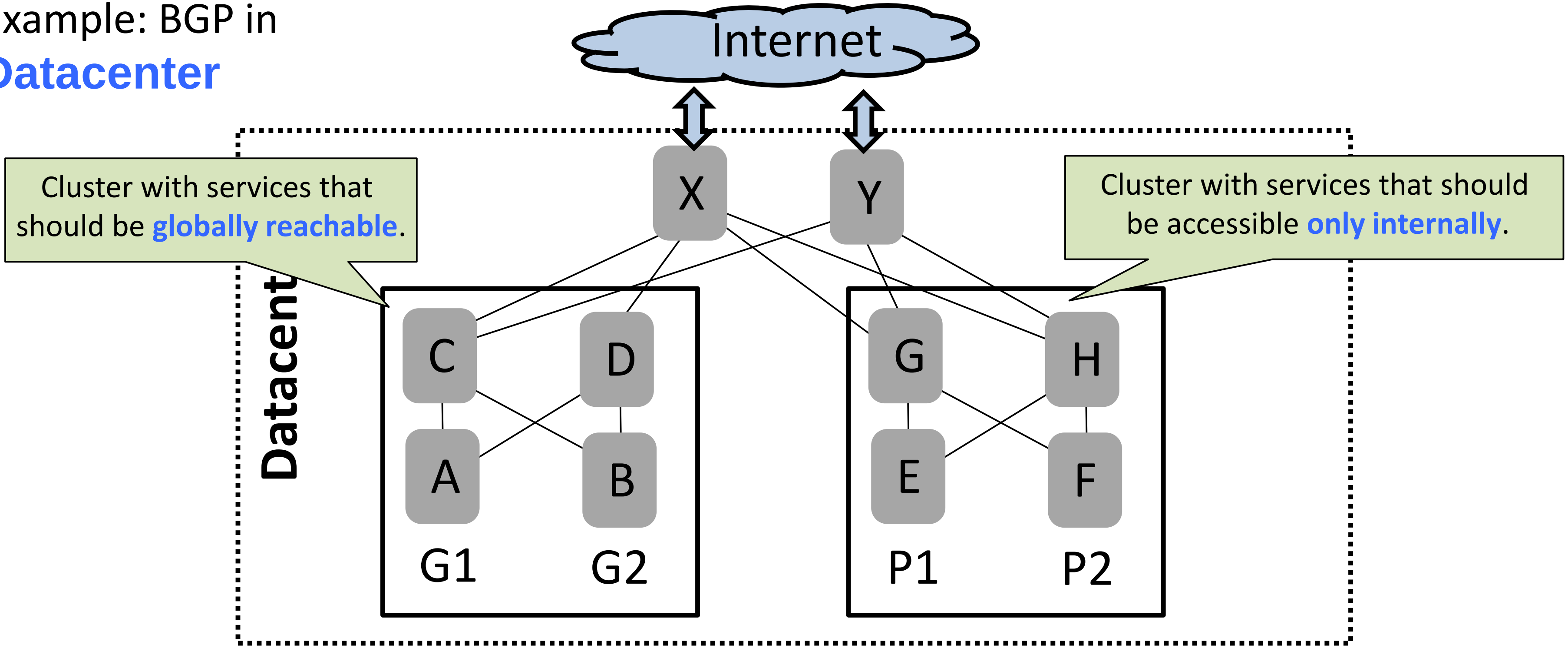
Example: BGP in
Datacenter



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

Example: Keeping Track of (Flexible) Routes Under Failures

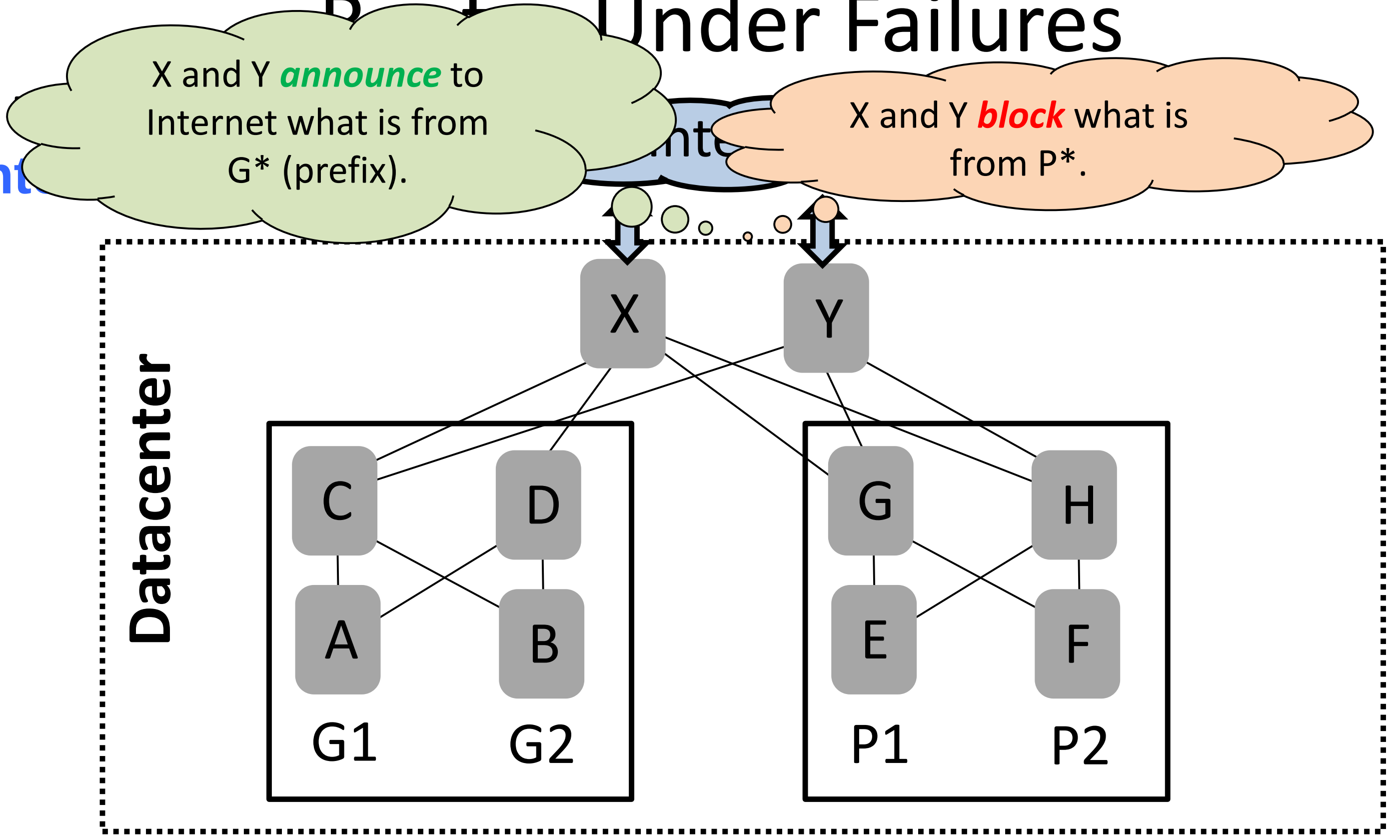
Example: BGP in **Datacenter**



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

Example: Keeping Track of (Flexible) Ro… Under Failures

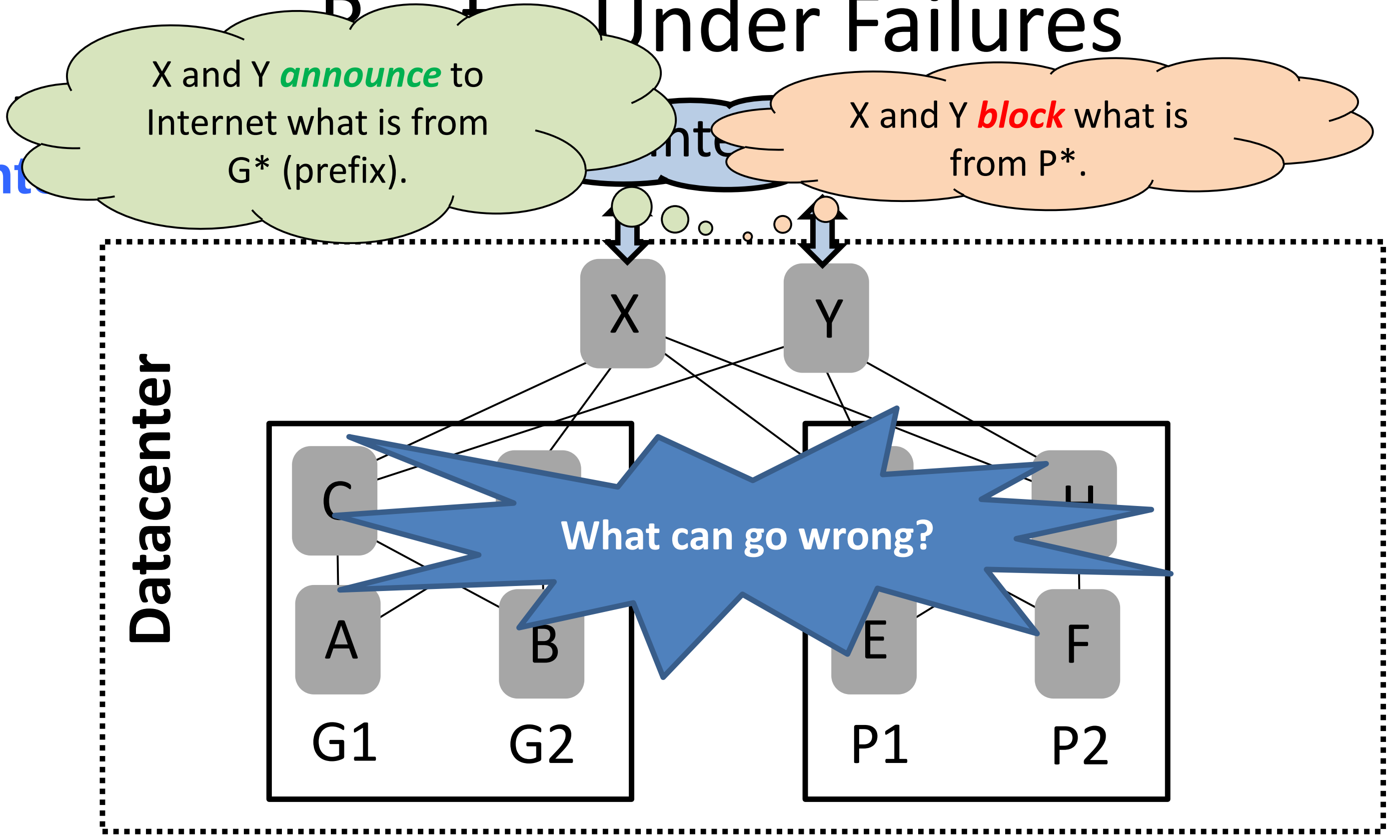
Example: …
Datacent…



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

Example: Keeping Track of (Flexible) Routes Under Failures

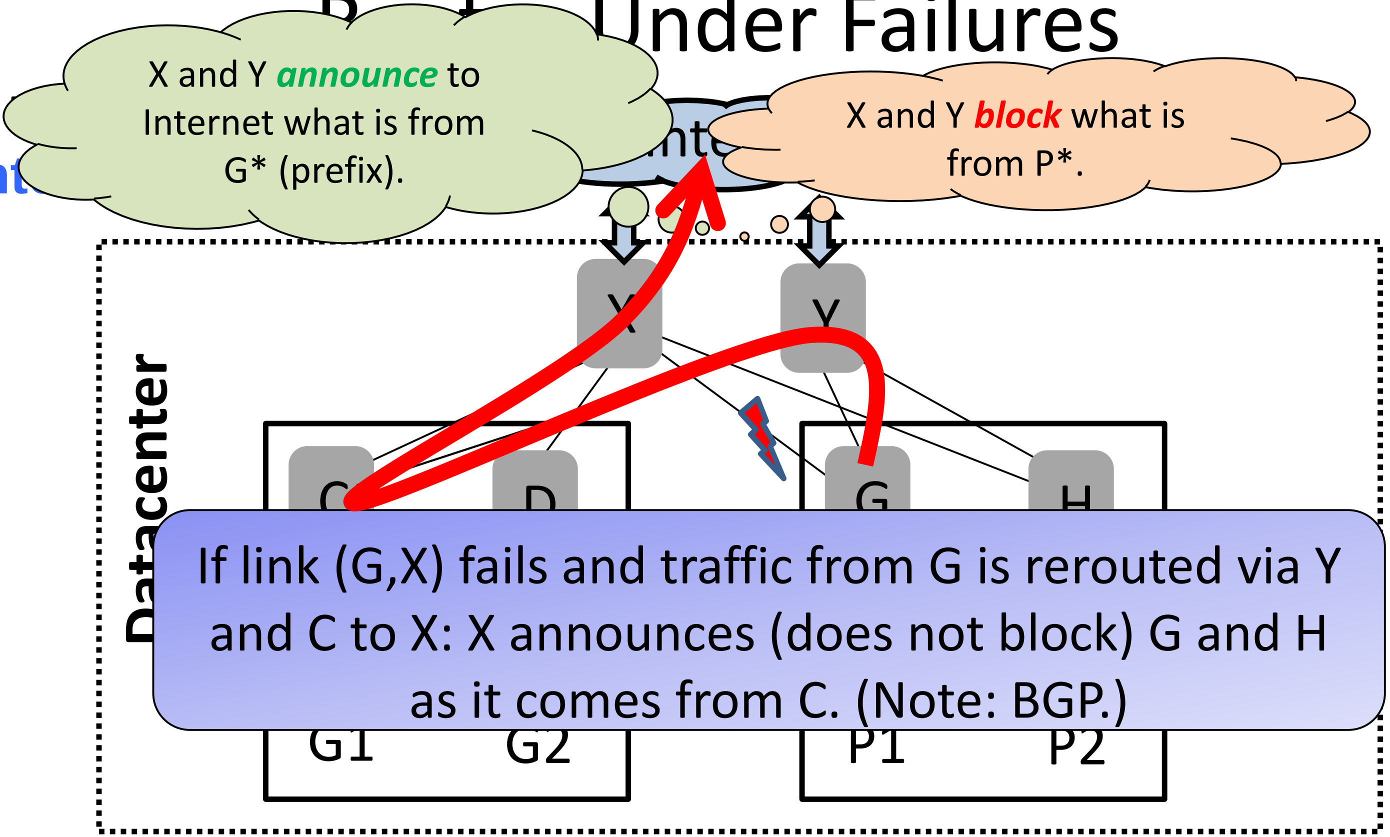
Example:
Datacent



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

Example: Keeping Track of (Flexible) ... Under Failures

Example:
Datacent



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.

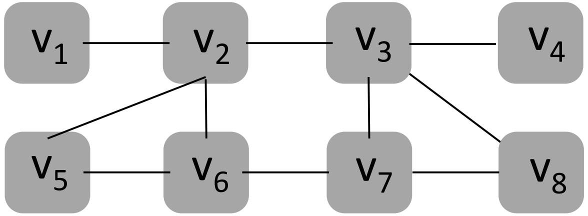
Managing Complex Networks is Hard for Humans

The Case for Automation!

Role of Formal Methods?

Example: MPLS Networks

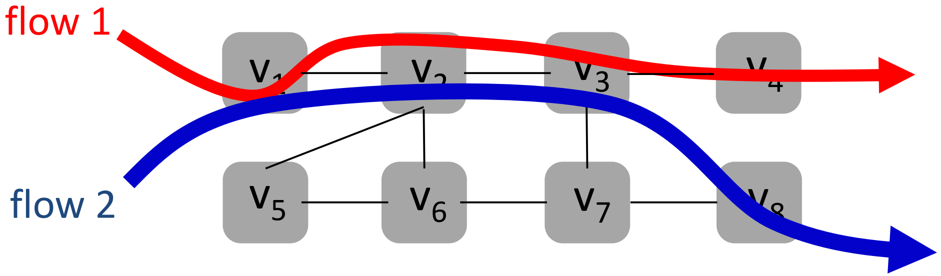
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

Example: MPLS Networks

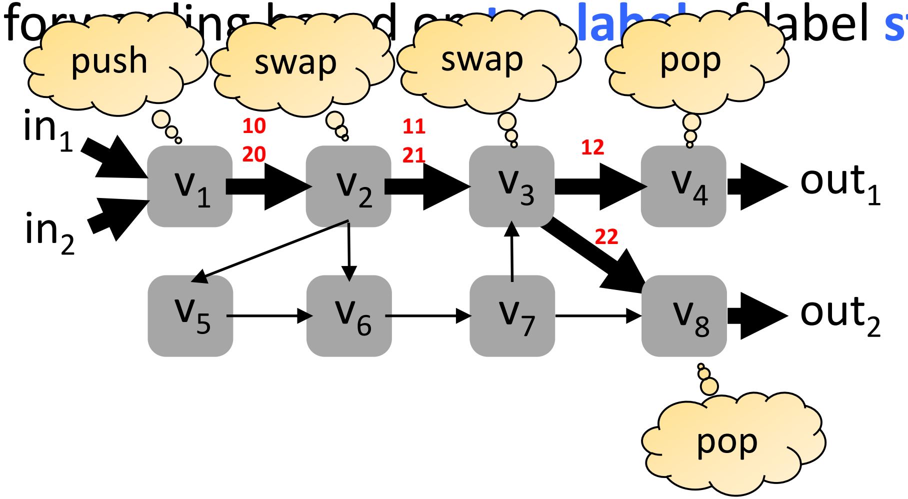
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

Example: MPLS Networks

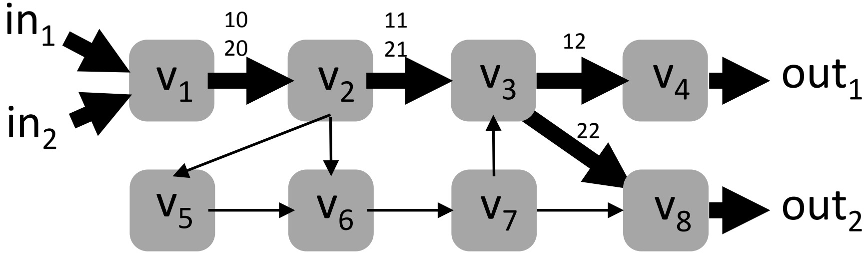
- MPLS: forwarding based on top **label** of label **stack**



Default routing of two flows

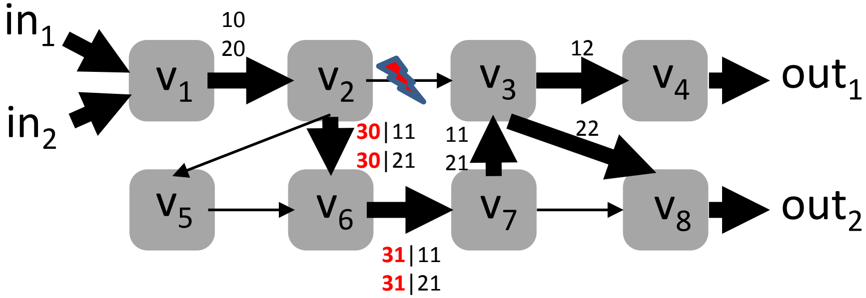
Fast Reroute Around *1 Failure*

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



Default routing of two flows

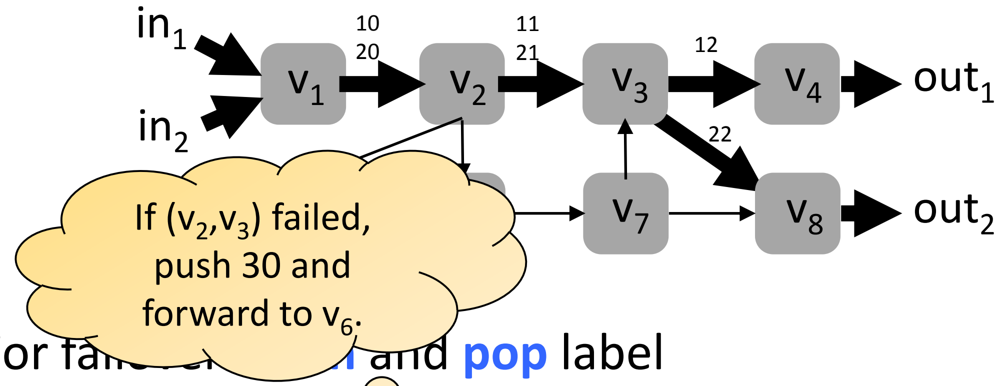
- For failover: **push** and **pop** label



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

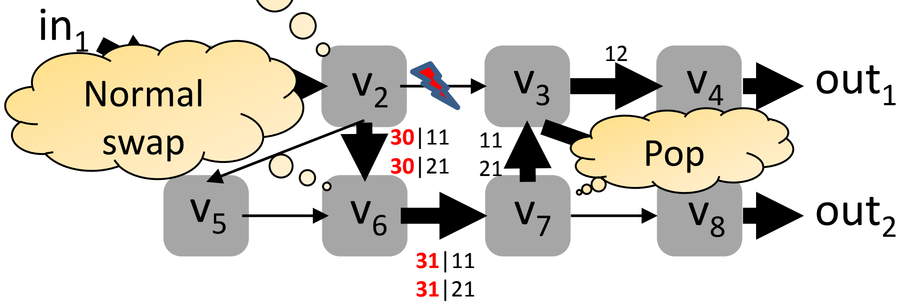
Fast Reroute Around *1 Failure*

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



Default routing of two flows

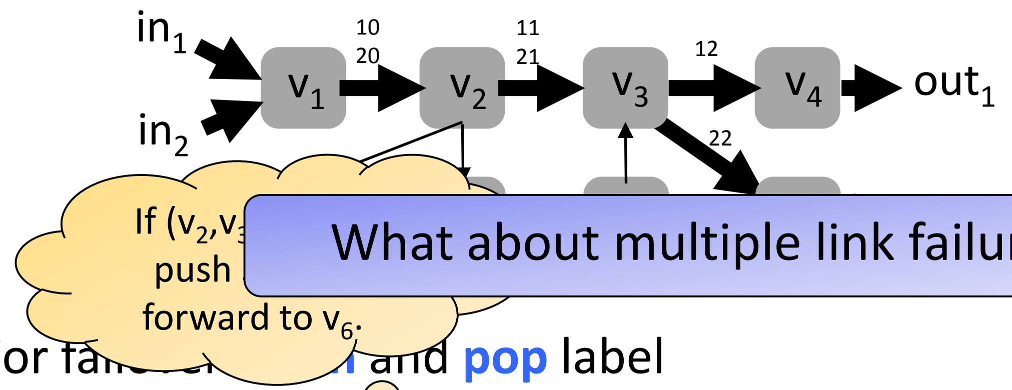
- For fa... and **pop** label



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

Fast Reroute Around *1 Failure*

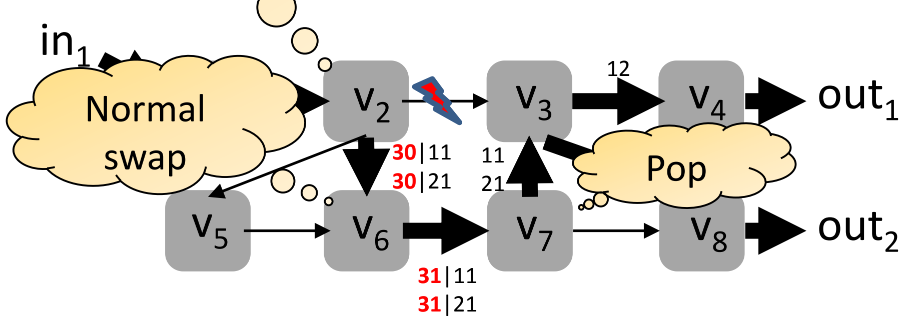
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

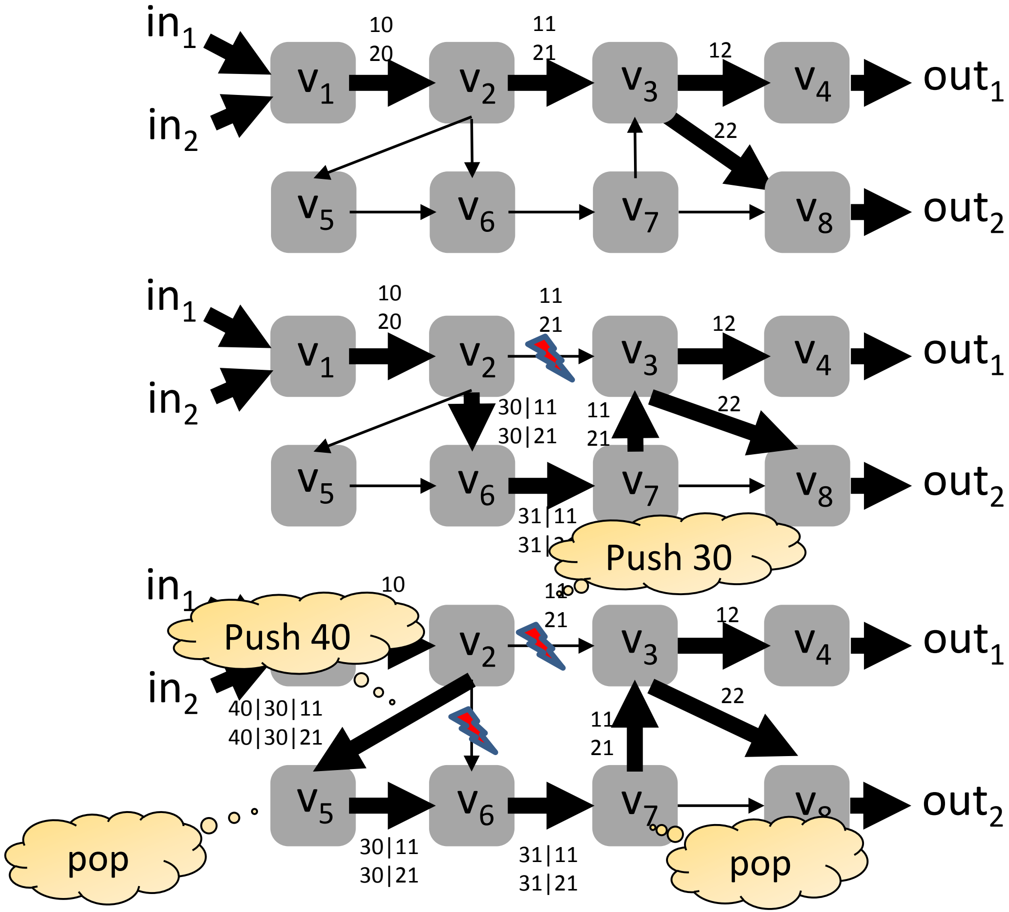
What about multiple link failures?

- For fa and **pop** label



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

2 Failures: Push *Recursively*



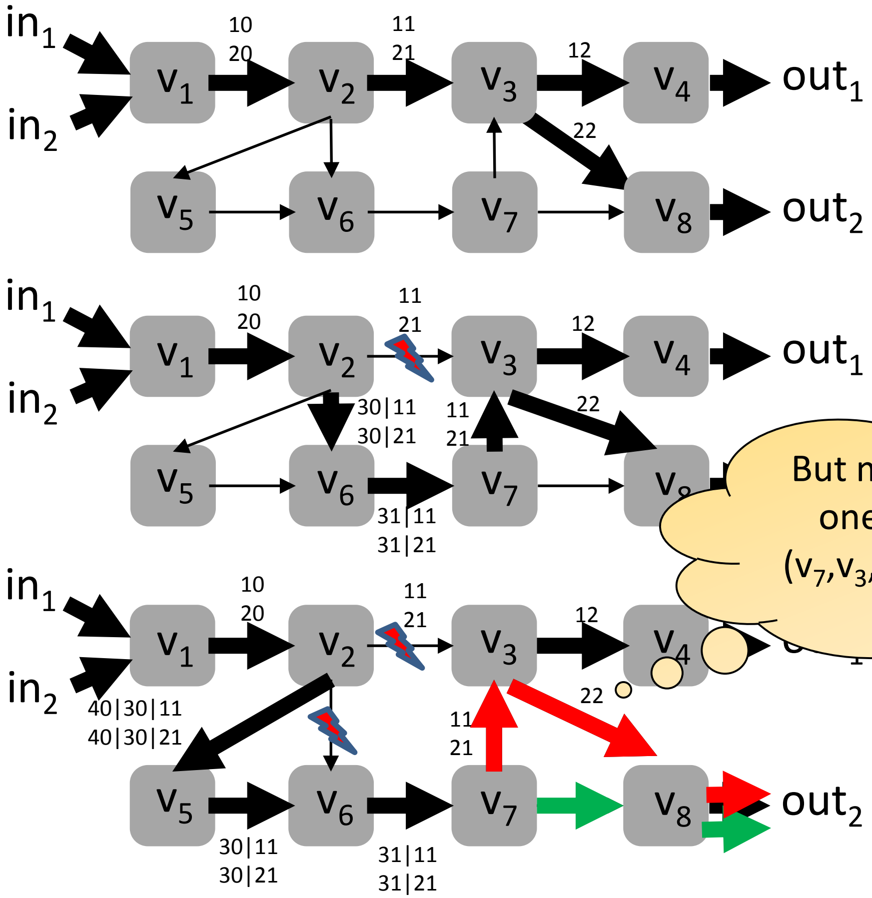
Original Routing

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

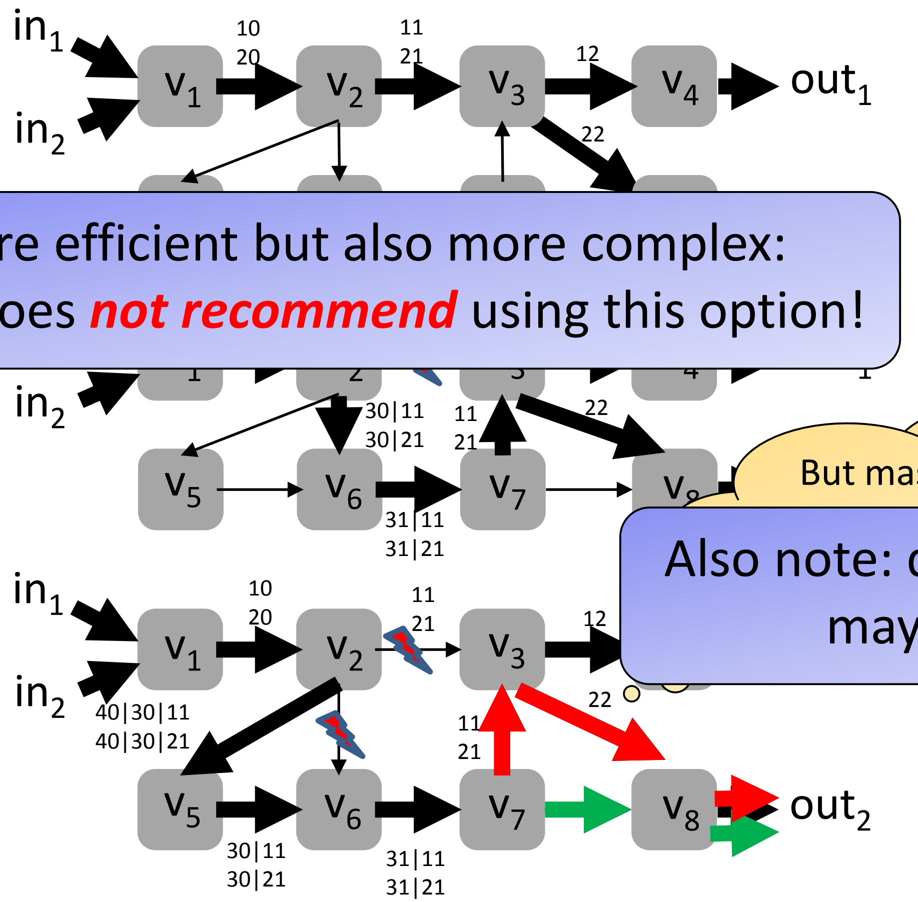
One failure: push 30: ...)

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)	$push($[illegible]
	in_2	$\perp$	(v_1, v_2)	pus[illegible]
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	swa[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](31)
	[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)	$push(40)$

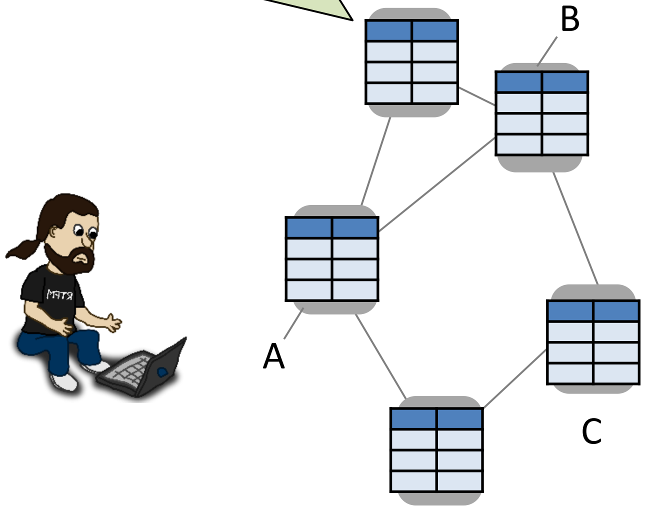
global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

Failover Tables

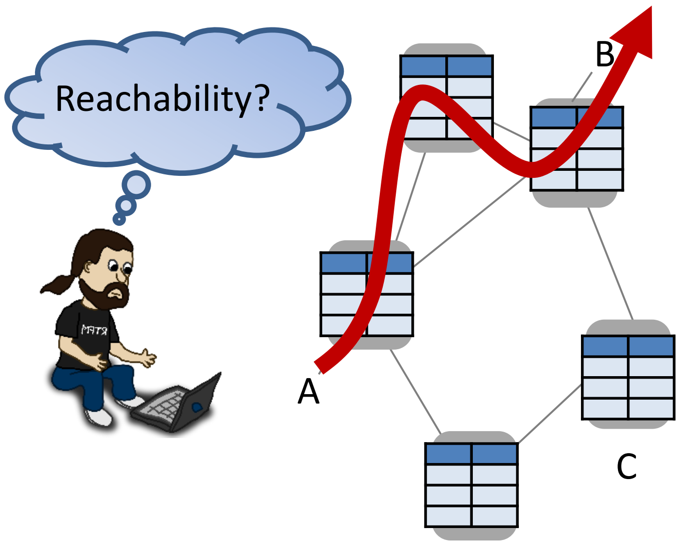
MPLS Tunnels in Today‘s ISP Networks

Responsibilities of a Sysadmin

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



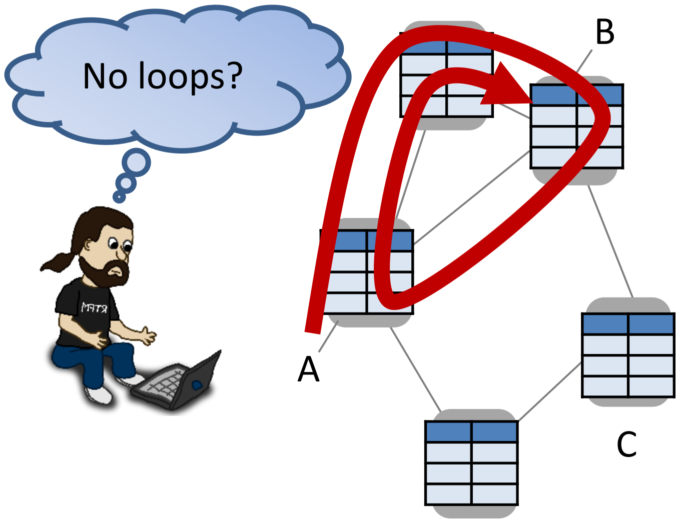
Responsibilities of a Sysadmin



Sysadmin responsible for:

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

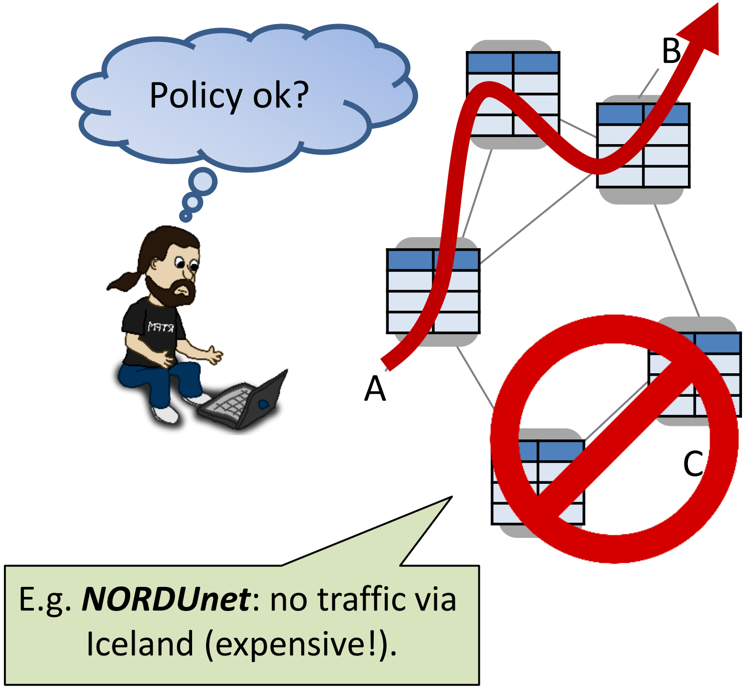
Responsibilities of a Sysadmin



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?

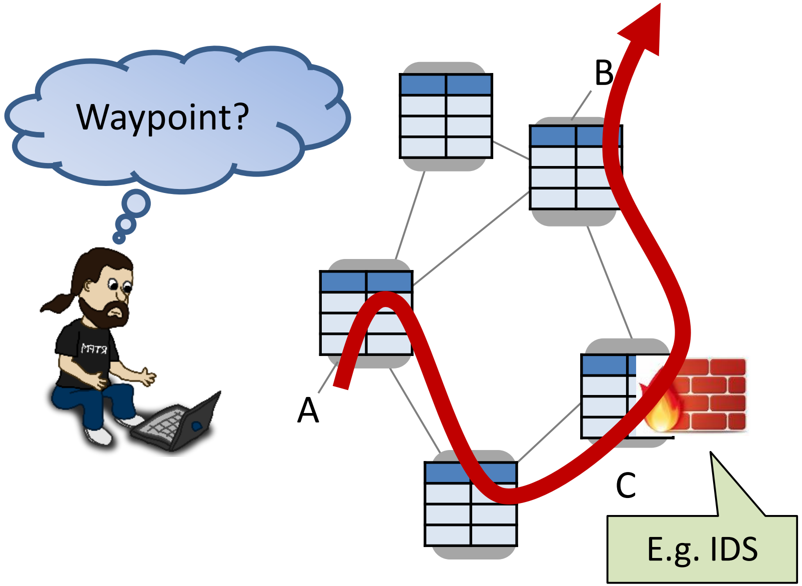
Responsibilities of a Sysadmin



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?

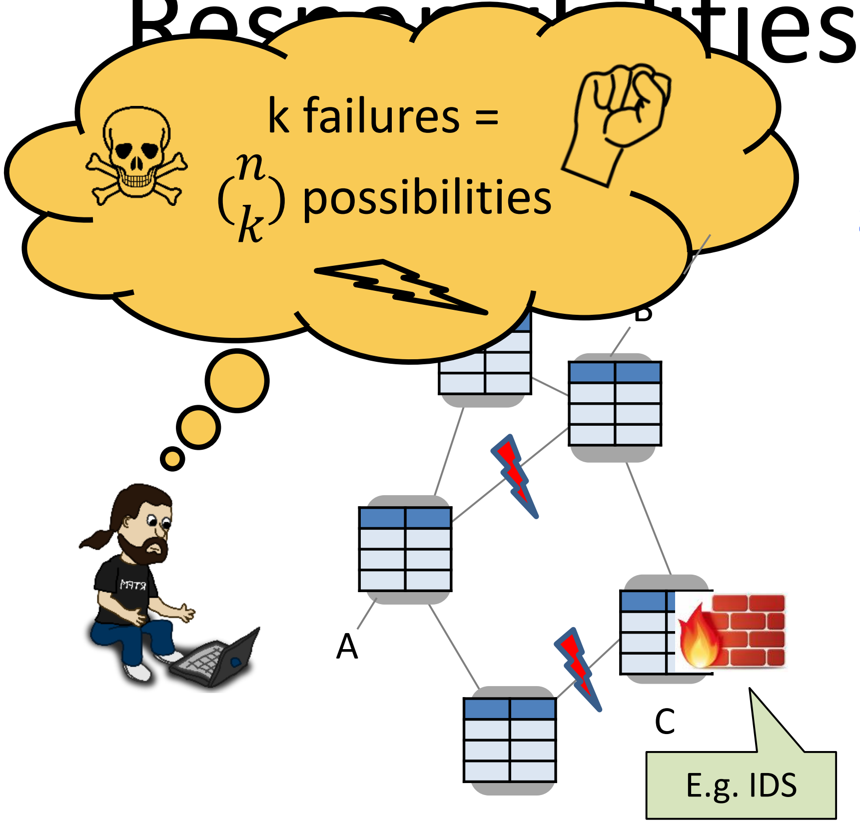
Responsibilities of a Sysadmin



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?!

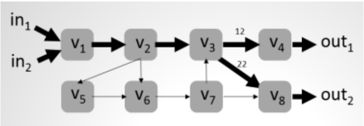
So what formal methods offer here?

A lot!

INFOCOM 2018

Leveraging Automata-Theoretic Approach

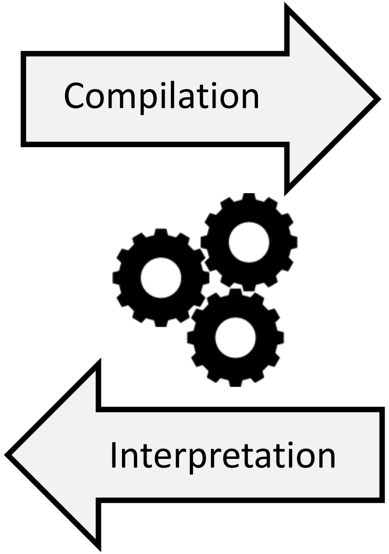
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$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

MPLS **configurations**, Segment Routing etc.



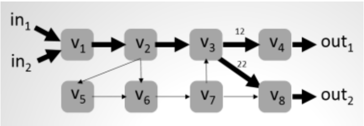
$$\mathrm{p}\mathrm{X} \Longrightarrow \mathrm{q}\mathrm{X}\mathrm{X}$$
$$pX \Longrightarrow qYX$$
$$qY \Longrightarrow rYY$$
$$rY \Longrightarrow r$$
$$rX \Longrightarrow pX$$

Pushdown Automaton and **Prefix Rewriting Systems** Theory

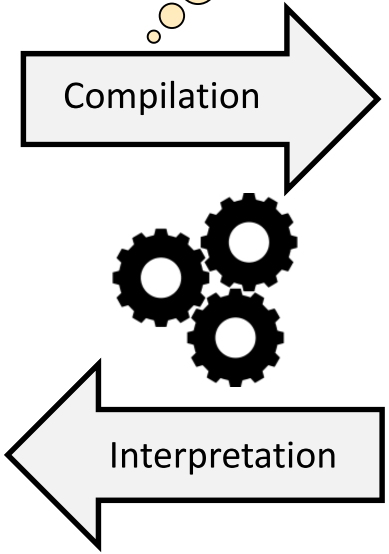
Leveraging Autom... oach

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...

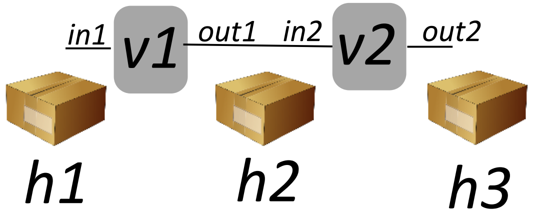
... with new header..

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

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

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

$$\dots$$



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:

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

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!

- 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

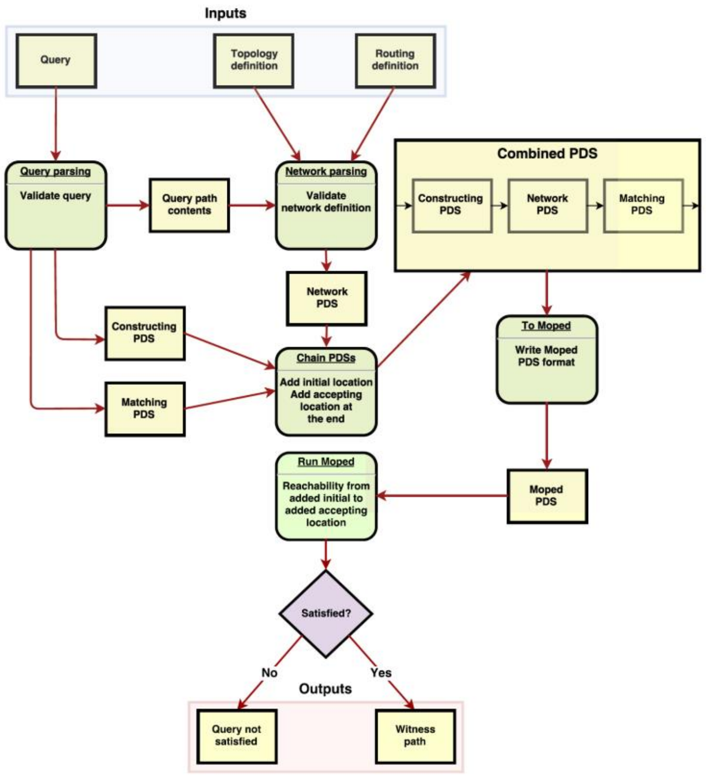
Preliminary Tool and Query Language

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

- In Python 3

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

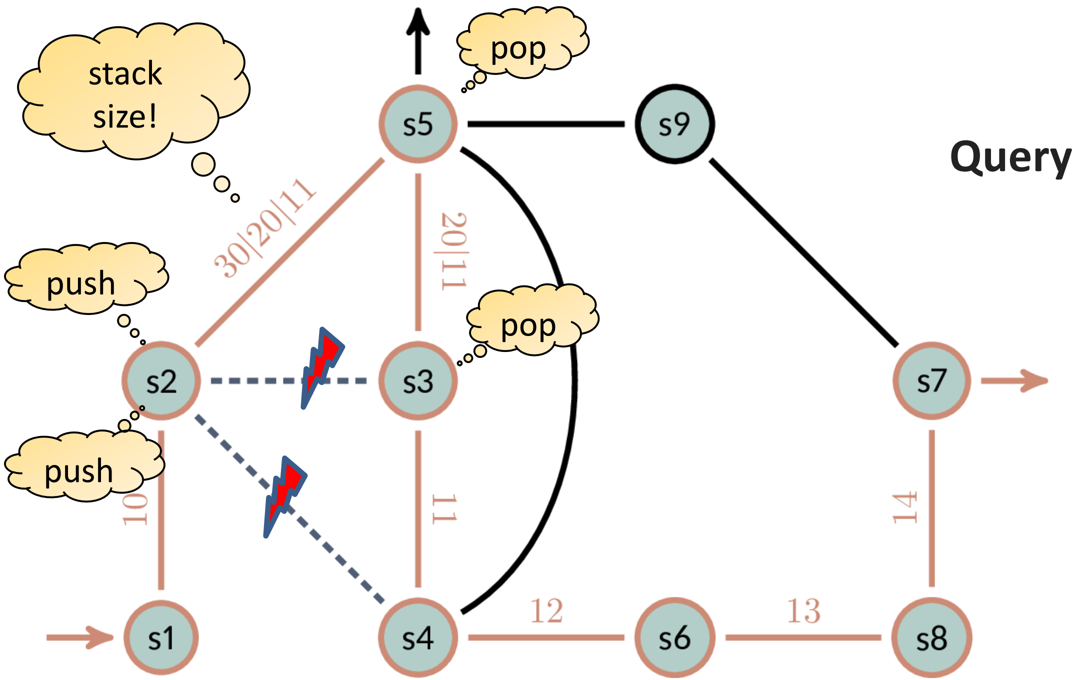
- Using ***Moped*** tool



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?



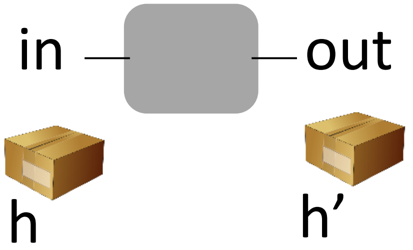
Query: k=2 [] s1 >> s5 >> s7 []

YES!

(Gives witness!)

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

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

VS

(Simplified) MPLS rules:

prefix rewriting

$$in \times L \rightarrow out \times \mathbf{OP}$$

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

Roadmap

- Networks are increasingly ***complex***: a case for **formal methods**?

- Networks are increasingly ***flexible***: a case for **self-adjusting networks**?

The new frontier!

Routing and TE: MPLS, SDN, etc.

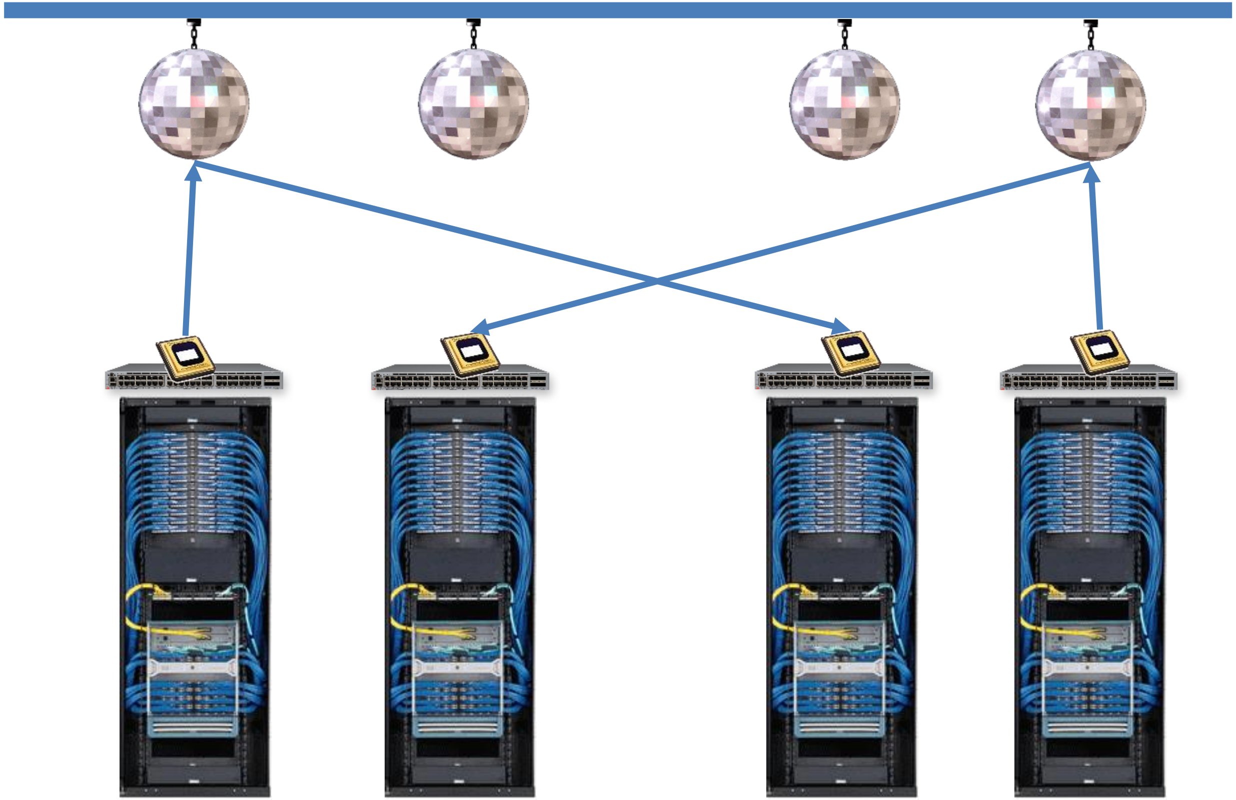
Flexible **placement**

Flexible **migration**

Topology **reconfiguration**

Flexibility of communication networks

t=1



Routing and TE:
MPLS, SDN, etc.

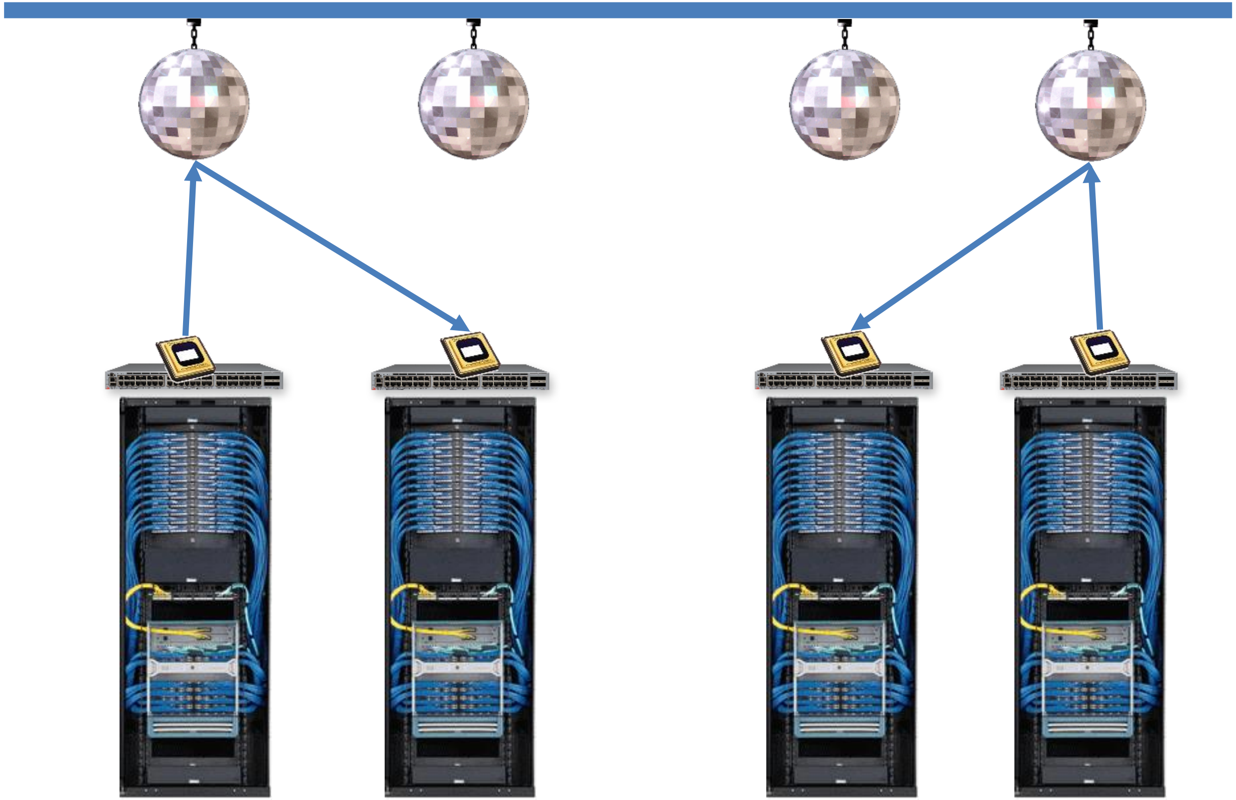
Flexible **placement**

Flexible **migration**

Topology **reconfiguration**

Flexibility of communication networks

t=2



Routing and TE: MPLS, SDN, etc.

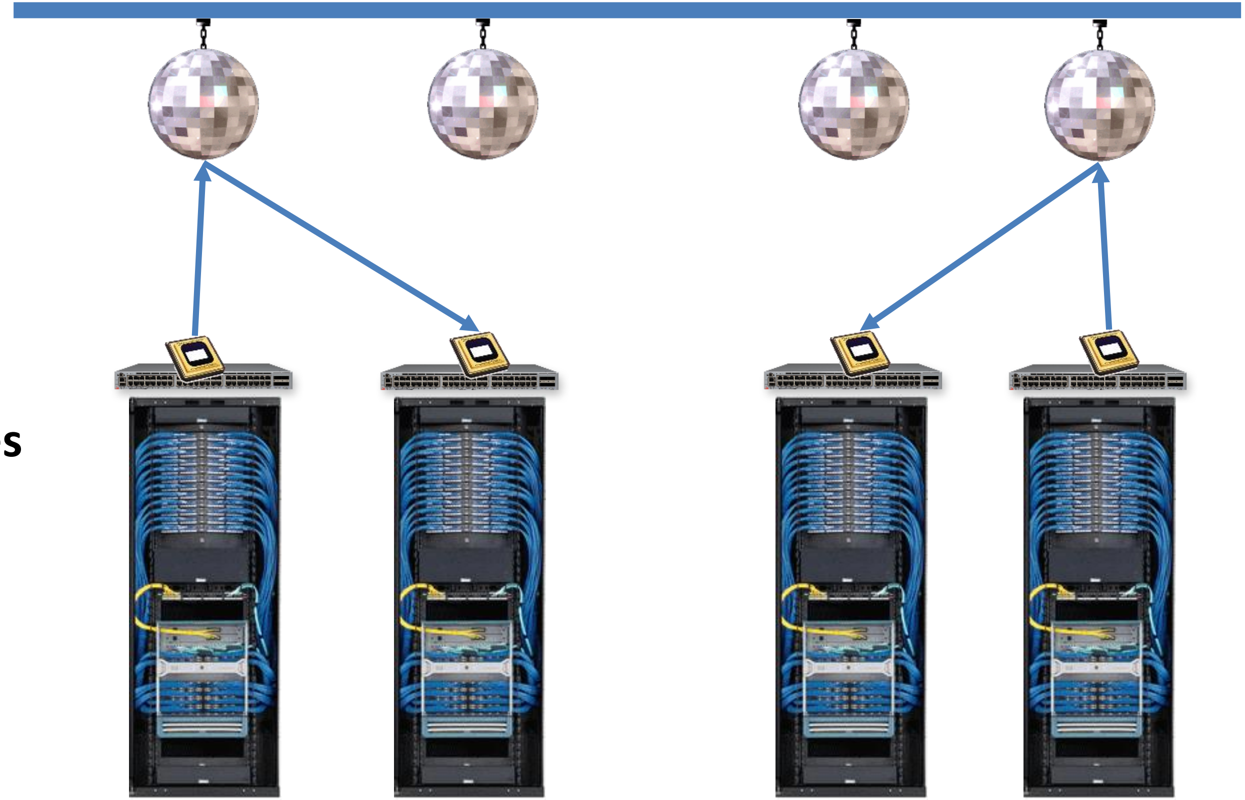
Flexible placement

Flexible migration

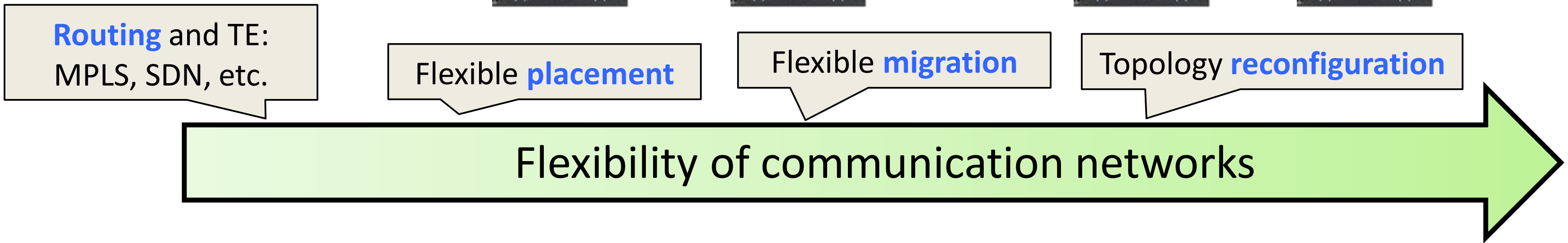
Topology reconfiguration

Flexibility of communication networks

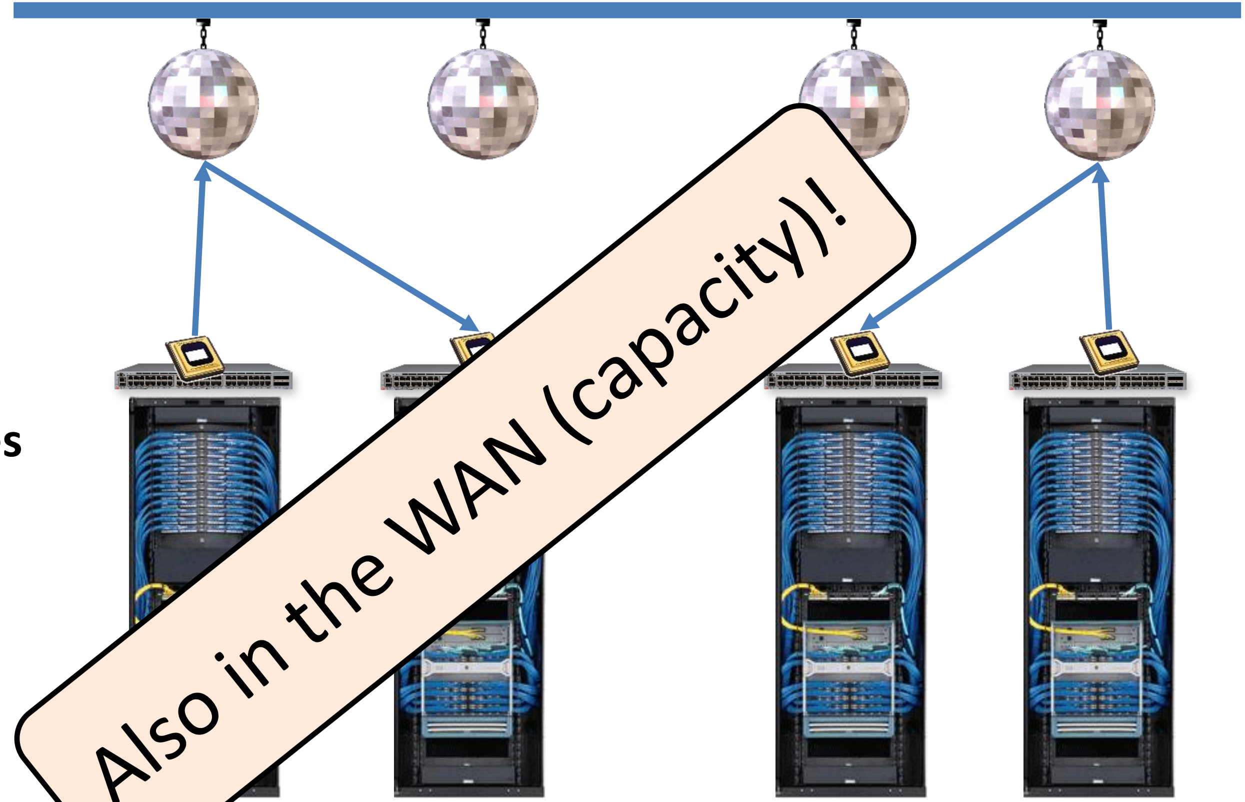
t=2



Less resources and latency



t=2



Also in the WAN (capacity)!

Less resources and latency

Routing and TE: MPLS, SDN, etc.

Flexible **placement**

Flexible **migration**

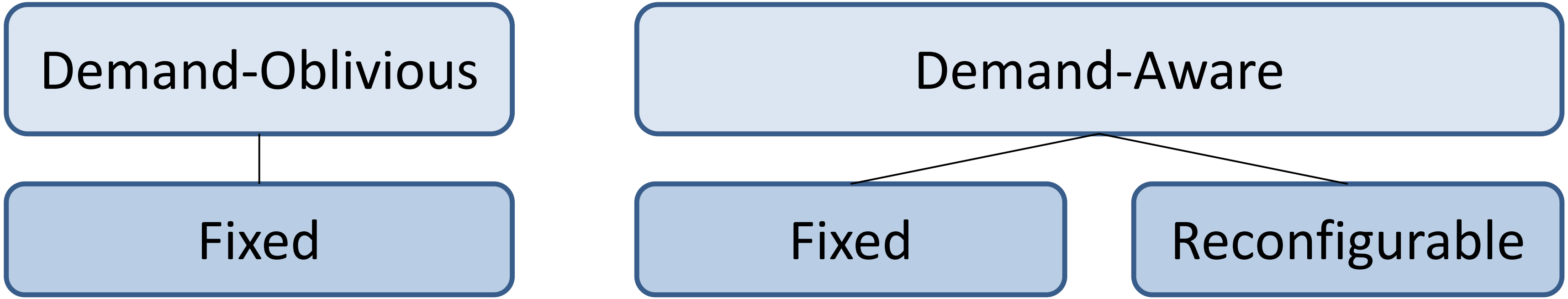
Topology **reconfiguration**

Flexibility of communication networks

What do self-adjusting networks offer?

Toward entropy-proportional routing
DISC 2017 (+ a BA), ANCS 2018, arXiv 2018

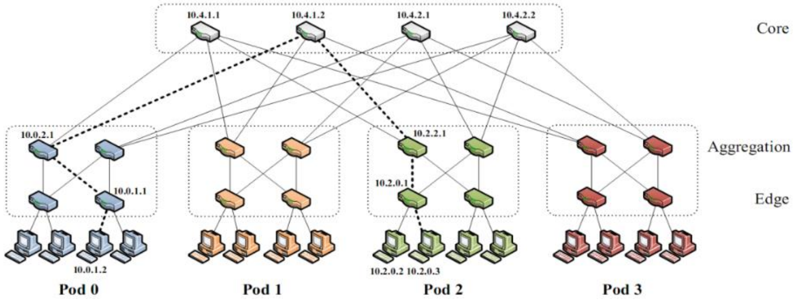
A Brief History of Self-Adjusting Networks

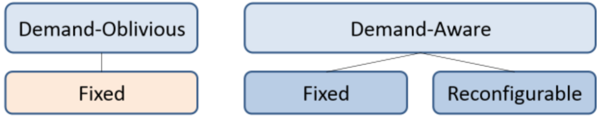


*Focus on **datacenters** but more general…*

Traditional Networks

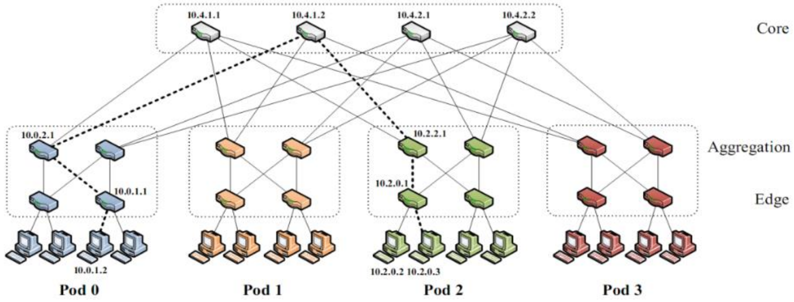
- Lower bounds and hard **trade-offs**, e.g., degree vs diameter
- Usually optimized for the “worst-case” (**all-to-all** communication)
- Example, fat-tree topologies: provide **full bisection bandwidth**

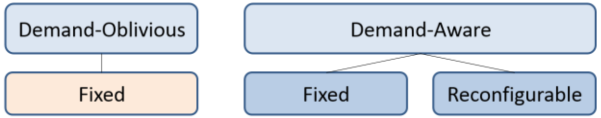




Traditional Networks

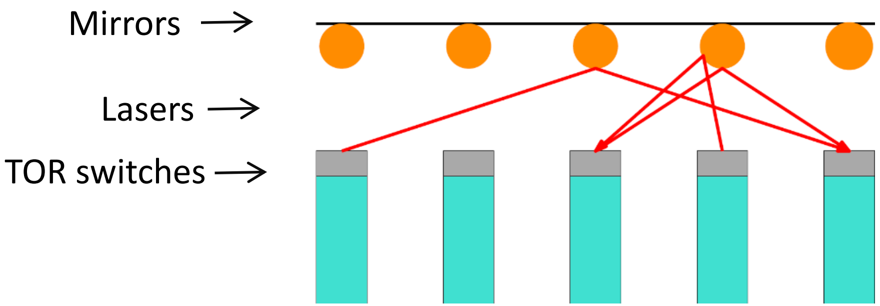
- Lower bounds and hard **trade-offs**, e.g., degree vs diameter
- Usually optimized for the "worst-case" (**all-to-all** communication)
- Example, fat-tree topologies: provide **full bisection bandwidth**

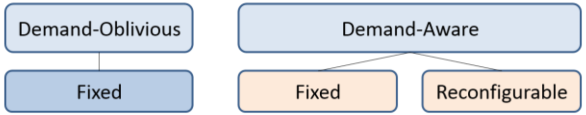




Vision: DANs and SANs

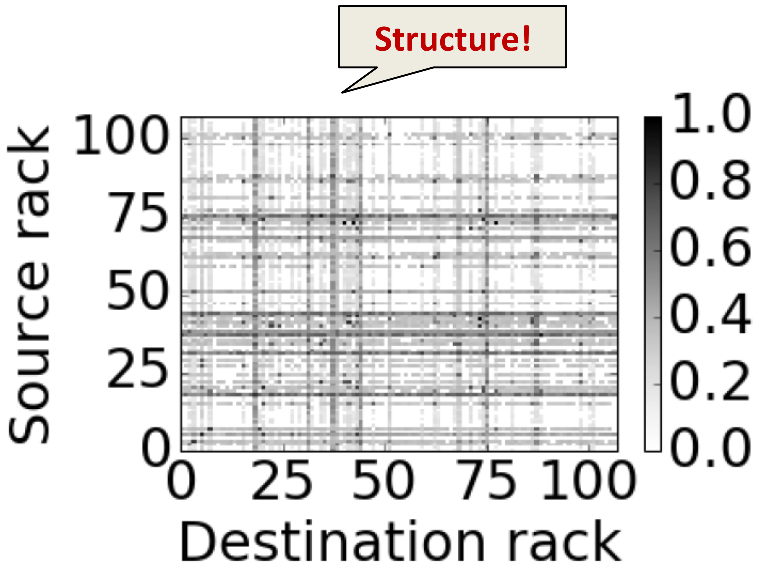
- ***DAN***: Demand-Aware Network
  - Statically optimized **toward the demand**
- ***SAN***: Self-Adjusting Network
  - **Dynamically optimized toward** the (time-varying) demand





Empirical Motivation

- Real traffic pattners are far from random: ***sparse*** structure



Heatmap of rack-to-rack traffic
ProjecToR @ SIGCOMM 2016

Empirical Motivation

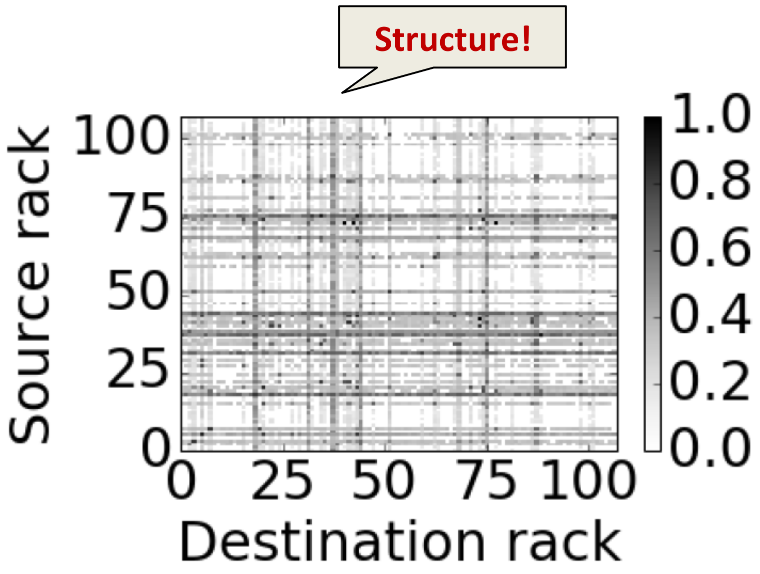
- Real traffic pattners are far from random: ***sparse*** structure

- Little to no communication between certain nodes

  A case for **DAN**s!

- But also ***changes*** over time

  A case for **SAN**s!



Heatmap of rack-to-rack traffic
ProjecToR @ SIGCOMM 2016

Analogous to *Datastructures*: Oblivious…

- Traditional, **fixed** BSTs do not rely on any assumptions on the demand

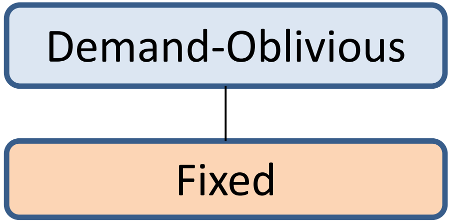
- Optimize for the **worst-case**

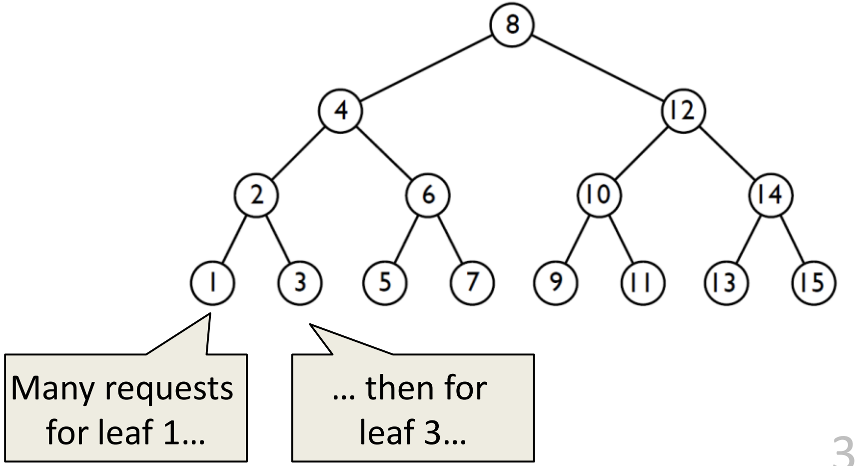
- Example **demand**:

  1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)

  many *many* *many* *many* *many*

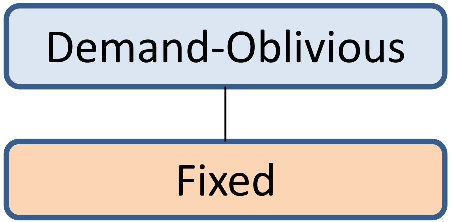
- Items stored at ***O(log n)*** from the root, **uniformly** and **independently** of their frequency





Analogous to *Datastructures*: Oblivious…

- Traditional, **fixed** BSTs do not rely on any assumptions on the demand



- Optimize for the **worst-case**

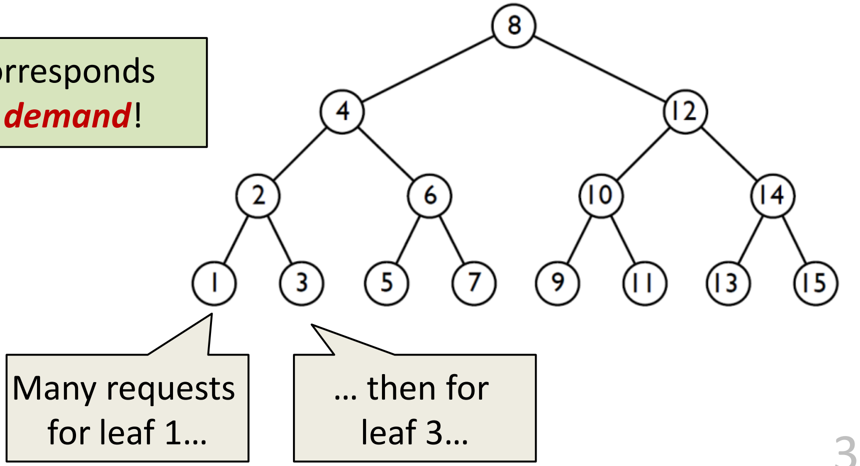
- Example **demand**:

1,…,1,3,…,3,5,…,5,7,…,7

many many many many

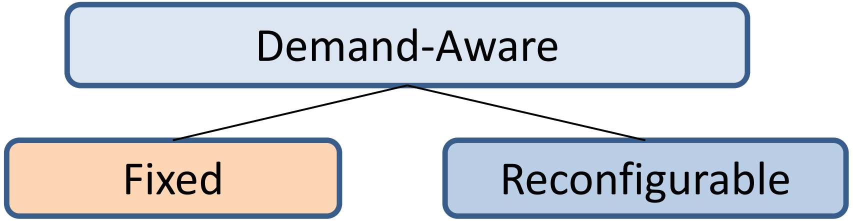
Amortized cost corresponds to ***max entropy of demand***!

- Items stored at ***O(log n)*** from the root, **uniformly** and **independently** of their frequency

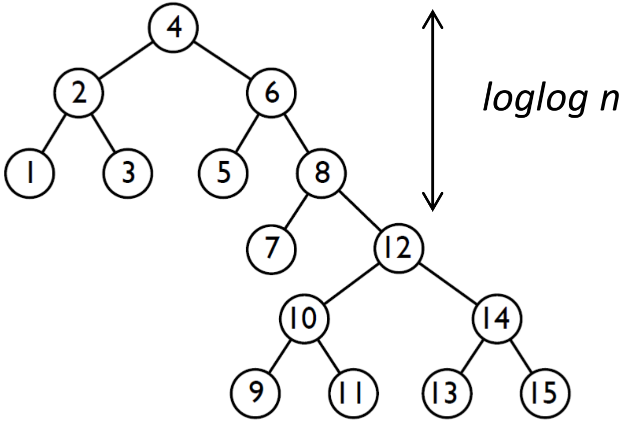


… Demand-Aware …

- **Demand-aware fixed** BSTs can take advantage of ***spatial locality*** of the demand



- Optimize: place frequently accessed elements close to the root
  - Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)



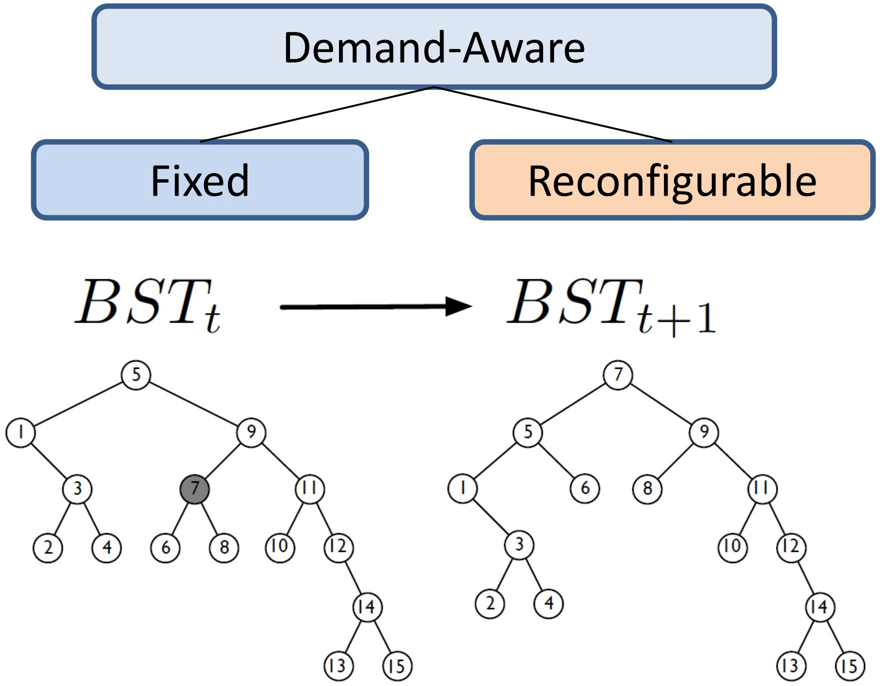
- E.g., **Mehlhorn** trees

Amortized cost corresponds to ***empirical entropy of demand***!

- Amortized cost ***O(loglog n)***

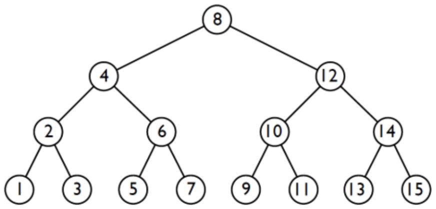
… Self-Adjusting!

- **Demand-aware reconfigurable** BSTs can additionally take advantage of ***temporal locality***
- By moving accessed element to the root: amortized cost is ***constant***, i.e., O(1)
  - Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
- Self-adjusting BSTs e.g., useful for implementing ***caches*** or garbage collection



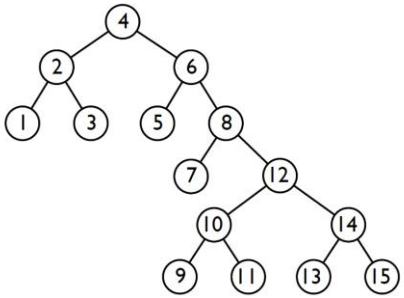
Datastructures

Oblivious



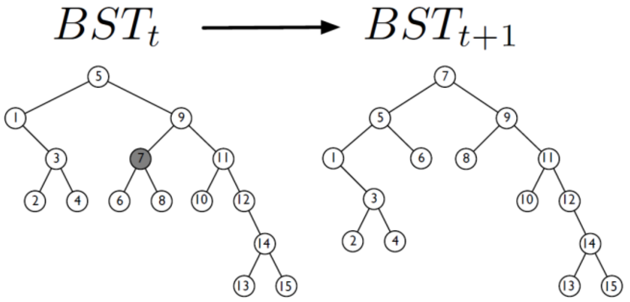
Lookup ***O(log n)***

Demand-Aware



Exploit **spatial locality**:
empirical entropy O(loglog n)

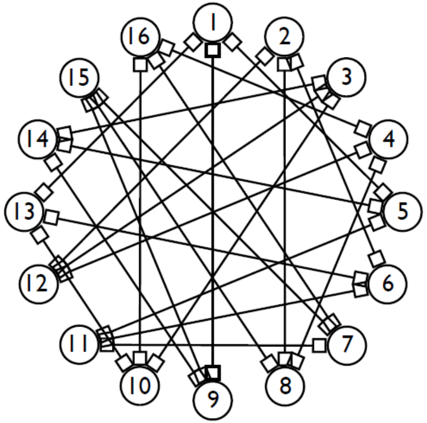
Self-Adjusting



Exploit **temporal locality** as well:
O(1)

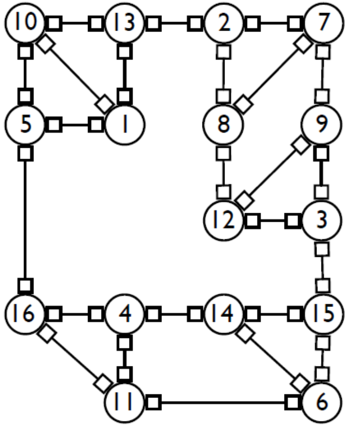
Analogously for Networks

Oblivious



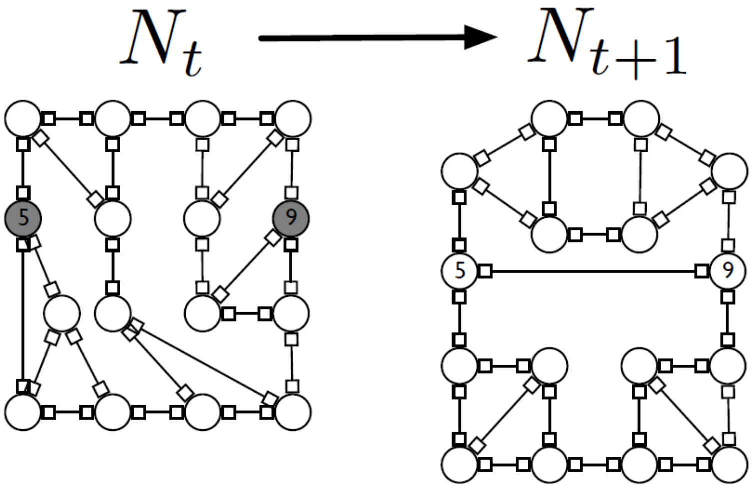
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



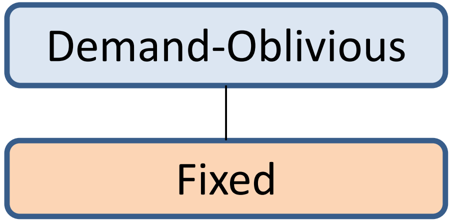
Exploit **spatial locality**: Route lengths depend on ***conditional entropy*** of demand

SAN

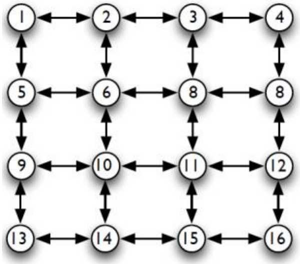


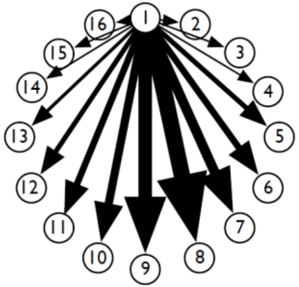
Exploit **temporal locality** as well

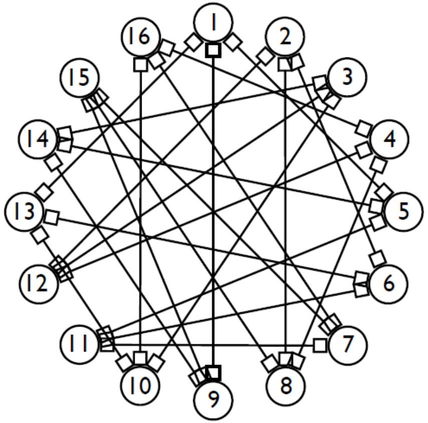
Oblivious Networks…



- Traditional, **fixed** networks (e.g. expander)
- Optimize for the **worst-case**
- Constant degree: communication partners at distance ***O(log n)*** from each other, **uniformly** and **independently** of their communication frequency
- Example **demands**:

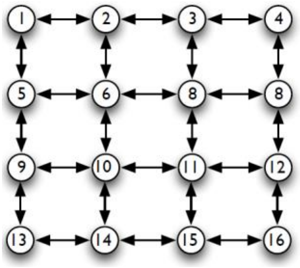


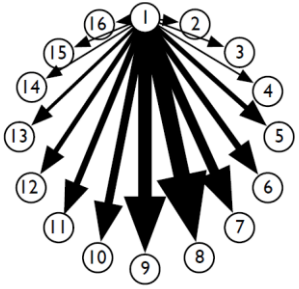




Oblivious Networks...

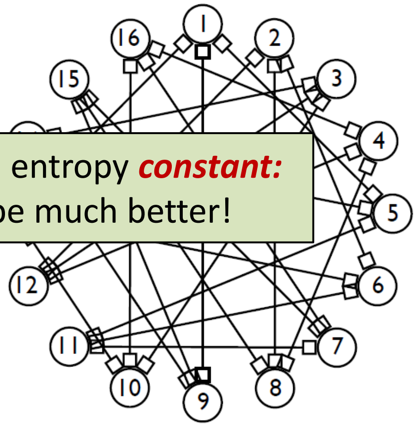
- Traditional, **fixed** networks (e.g. expander)
- Optimize for the **worst-case**
- Constant degree: communication partners at distance ***O(log n)*** from each other, **uniformly** and **independently** of their communication frequency
- Example **demands**:





Demand-Oblivious

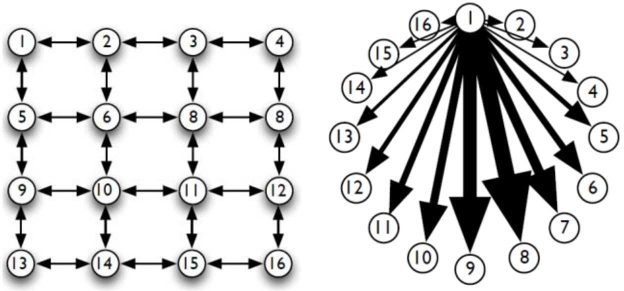
Fixed

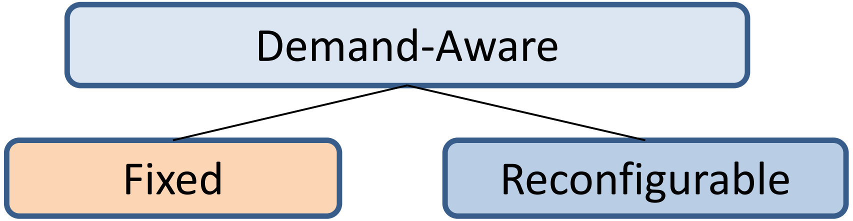


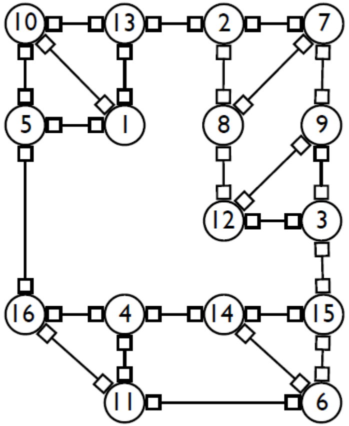
Conditional entropy ***constant:***
DANs would be much better!

… DANs …

- **Demand-aware fixed** networks can take advantage of ***spatial locality***
- Optimize: place frequently communicating nodes close
- ***O(1)*** routes for our demands:



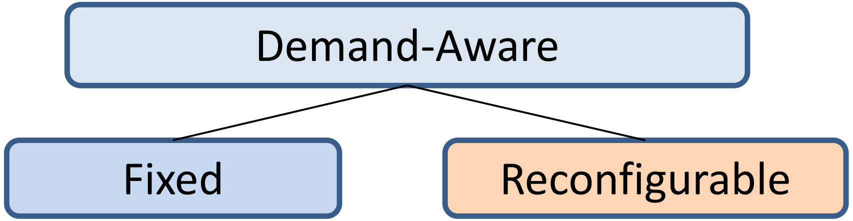


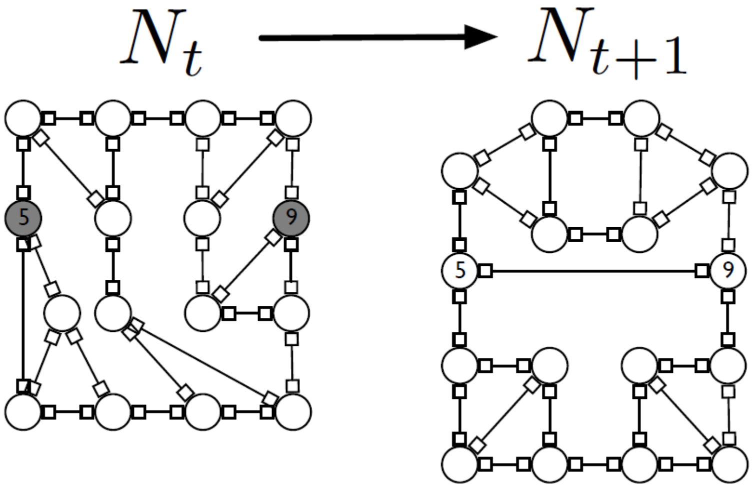


… SANs!

- **Demand-aware reconfigurable** networks can additionally take advantage of ***temporal locality***

- By moving communicating elements close





Diving a Bit Deeper: DAN

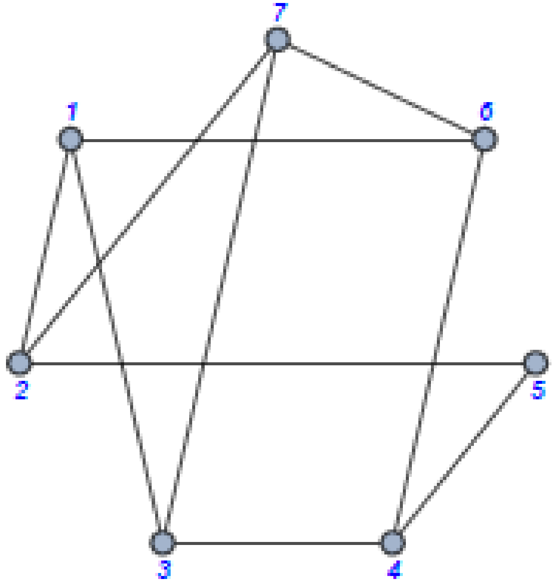
Workload: can be seen as graph as well.

Destinations

Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

design



Demand matrix: joint distribution

DAN (of constant degree)

Diving a Bit Deeper: DAN

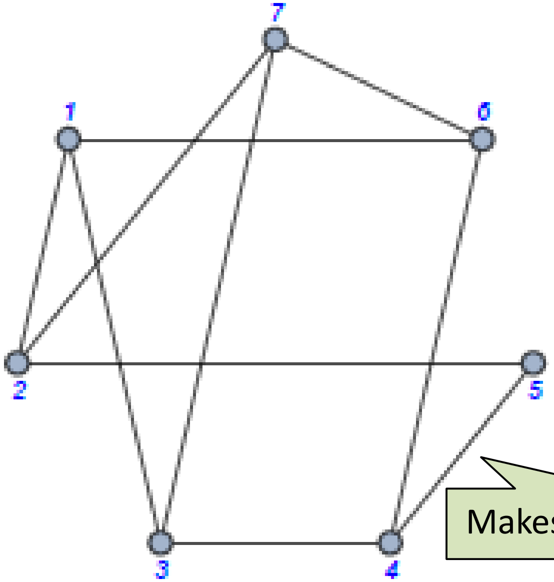
Destinations

Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	2 / [illegible]
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	[illegible]	[illegible]	[illegible] / 13
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Much from 4 to 5.

design



Makes sense to add link!

Demand matrix: joint distribution

DAN (of constant degree)

Diving a Bit Deeper: DAN

Destinations

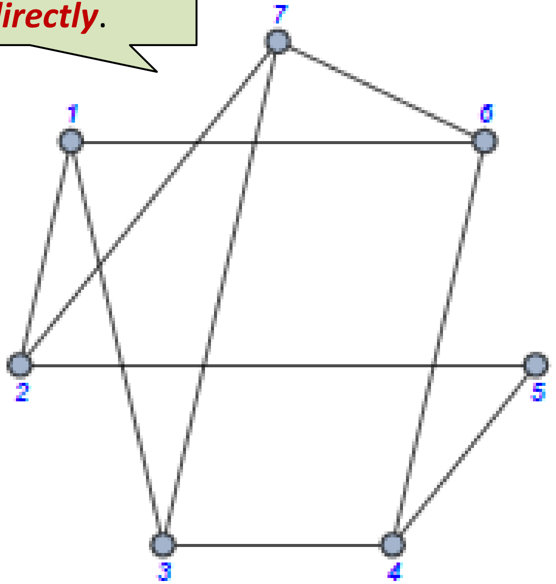
1 communicates to many.

Sources

						6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

design

Bounded degree: route to 7 ***indirectly***.



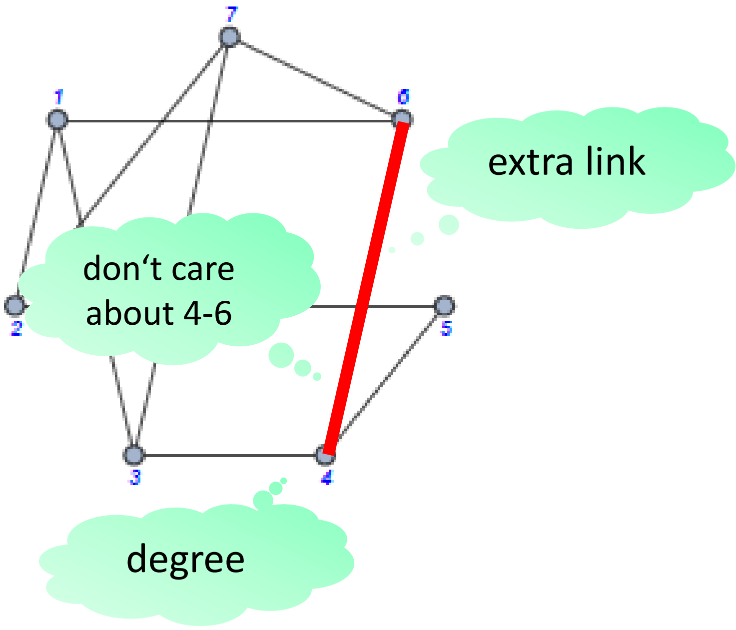
Demand matrix: joint distribution

DAN (of constant degree)

DAN: Relationship to…

Sparse, low-distortion **graph spanners**

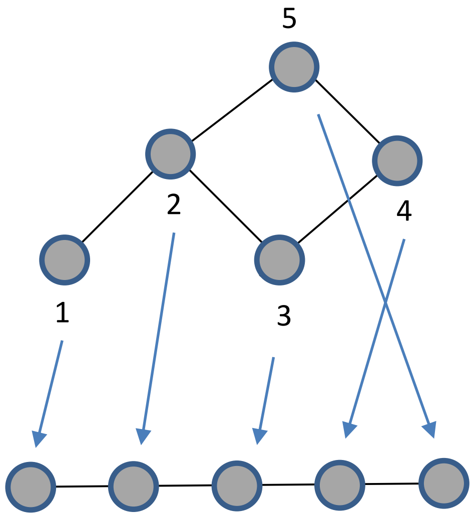
- Similar: keep distances in a „compressed network“ (few edges)
- *But:*
  - We only care about path length **between communicating nodes**, not all node pairs
  - We want **constant degree**
  - Not restricted to subgraph but can have „**additional links**“ (like geometric spanners)



DAN: Relationship to…

Minimum Linear **Arrangement** (MLA)

- MLA: map guest graph to line (host graph) so that sum of distances is minimal
- DAN similar: if degree bound = 2, DAN is line or ring (or sets of lines/rings)
- *But* unlike “**graph embedding problems**“
  - The host graph is also subject to optimization
  - Does this render the problem simpler or harder?



Diving a Bit Deeper: DAN

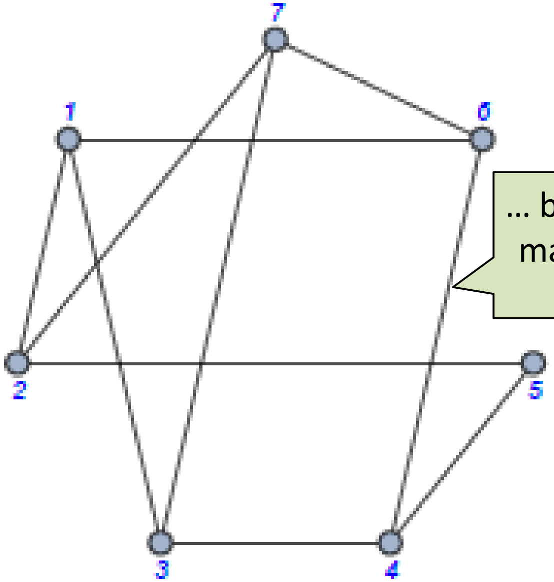
Destinations

Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	[illegible]	[illegible]
6	$\frac{2}{65}$	0	0	0	0	[illegible]	[illegible]
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

4 and 6 don't communicate…

design



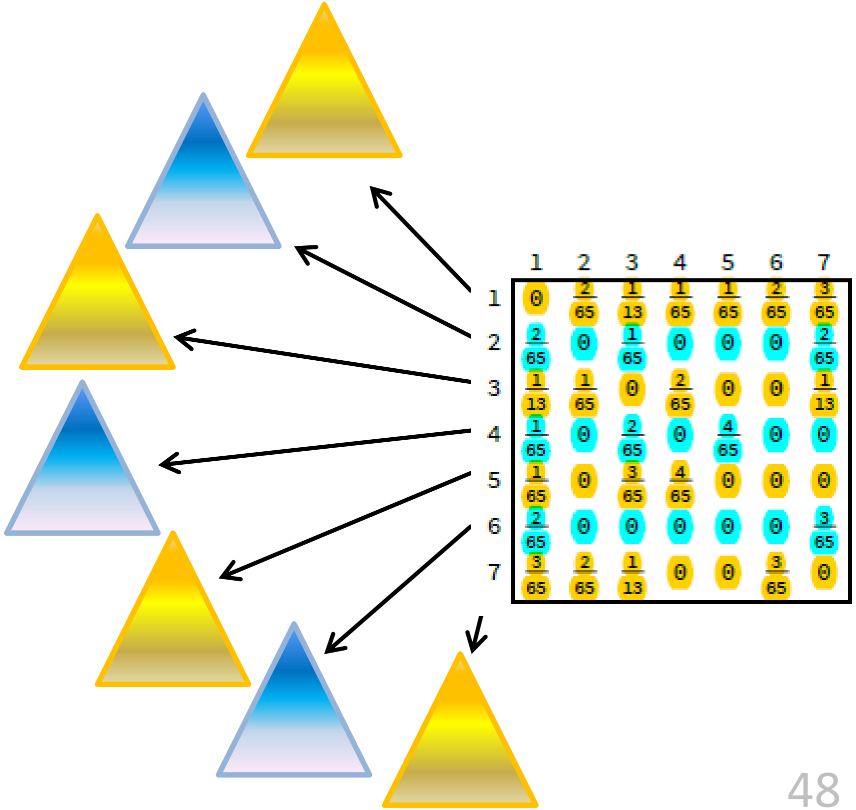
… but "extra" link still makes sense: ***not a subgraph***.

Demand matrix: joint distribution

DAN (of constant degree)

Lower Bound: Idea

- **Proof idea** (EPL=$\Omega(H_\Delta(Y|X))$):
- Build ***optimal*** Δ-ary tree for each source i: entropy lower bound known on EPL known for binary trees (***Mehlhorn*** 1975 for BST but proof does not need search property)
- Consider ***union*** of all trees
- Violates ***degree restriction*** but valid lower bound



$$
\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0
\end{array}
$$

Lower Bound: Idea

Do this in **both dimensions**:

EPL ≥ $\Omega(\max\{H_\Delta(Y|X), H_\Delta(X|Y)\})$

$\Omega(H_\Delta(X|Y))$

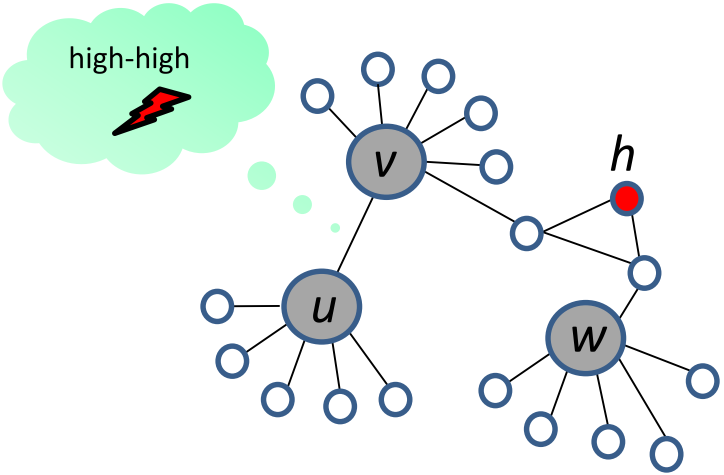
	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

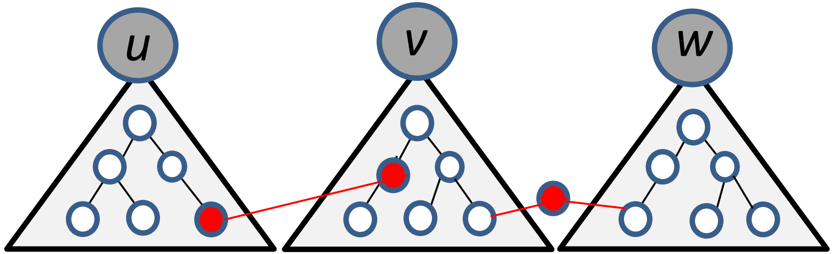
$\Omega(H_\Delta(Y|X))$

$\mathcal{D}$

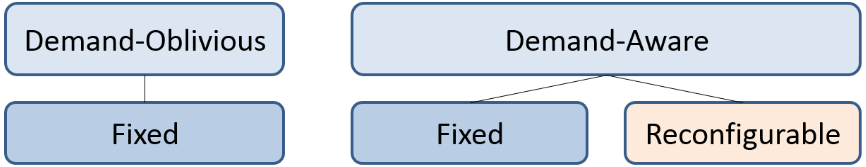
(Tight) Upper Bounds: Algorithm Idea

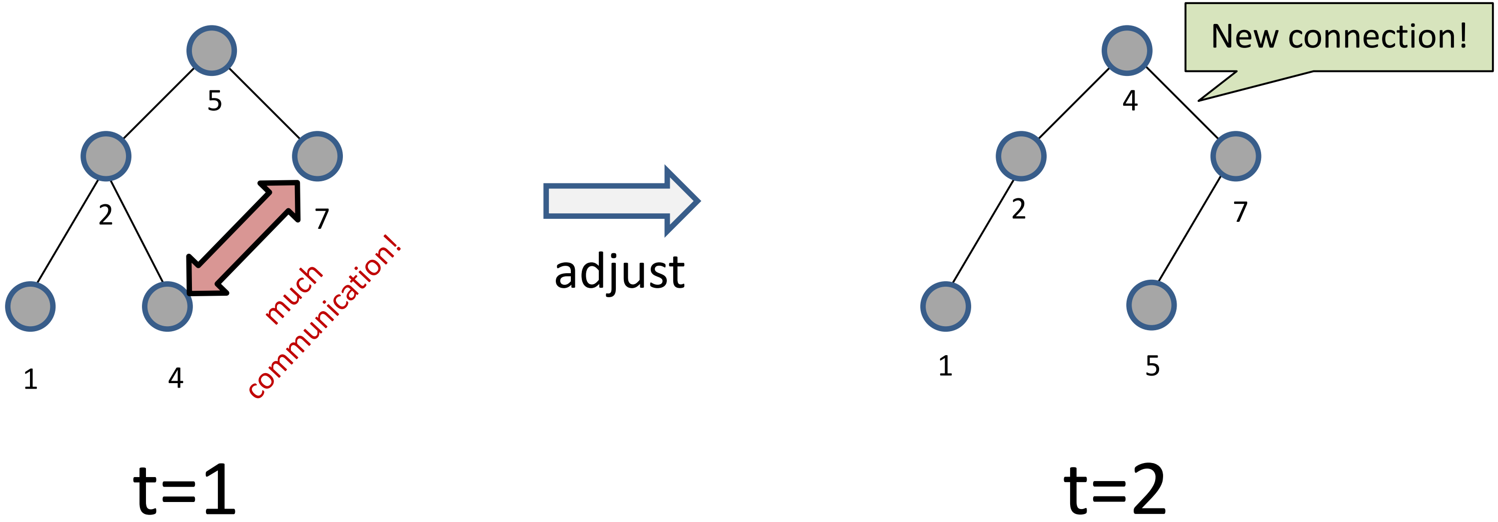
- Idea: construct **per-node optimal tree**
  - BST (e.g., Mehlhorn)
  - Huffman tree
  - Splay tree (!)
- Take **union** of trees but reduce degree
  - E.g., in sparse distribution: leverage **helper** nodes between two “large” (i.e., high-degree) nodes





Example: Self-Adjusting Network (SANs) *Trees*



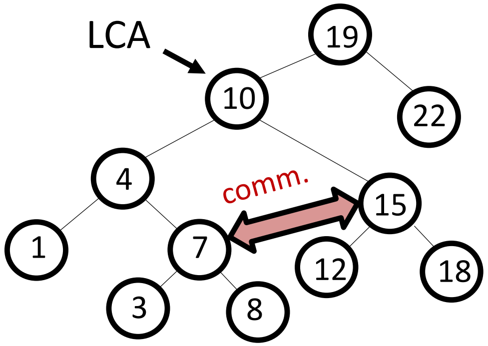


Challenges: How to **minimize reconfigurations**?
How to keep network **locally** routable?

SplayNet: Towards Locally Self-Adjusting Networks. **TON** 2016.

SAN Idea 1: SplayNet

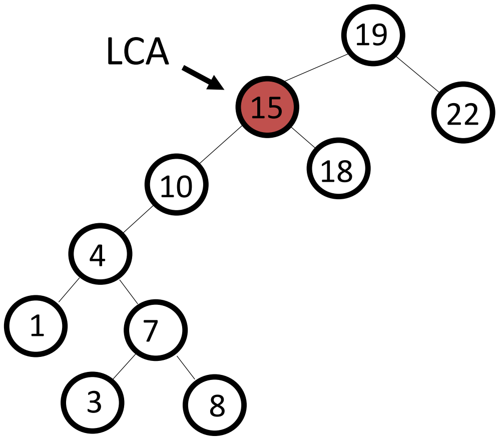
- Idea: Binary Search Tree (**BST**) *network*
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to **Least Common Ancestor (LCA)**



SplayNet

SAN Idea 1: SplayNet

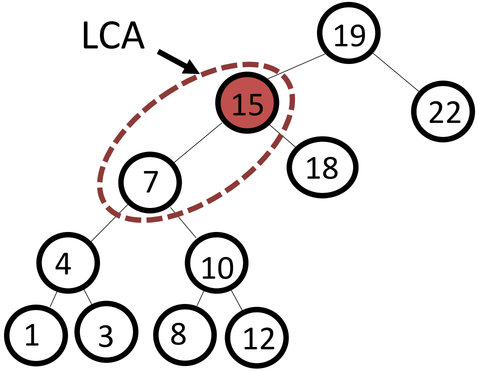
- Idea: Binary Search Tree (**BST**) *network*
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to **Least Common Ancestor (LCA)**



SplayNet

SAN Idea 1: SplayNet

- Idea: Binary Search Tree (**BST**) *network*
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to **Least Common Ancestor (LCA)**



SplayNet

SplayNet: Properties

Property 1: Optimal static network can be computed in polynomial-time (dynamic programming)

– **Unlike unordered tree?**

1. Define: flow out of interval I

$W_I(v)=\sum_{u \in I'} w(u,v) + w(v,u)$

Decouple cost to ouside: distance to root of T_I only

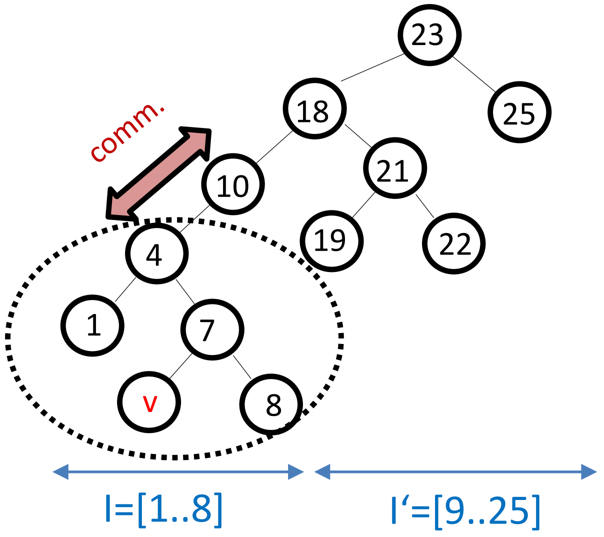
2. Cost of a given tree T_I on I:

$\mathrm{Cost}(T_I, W_I)=[\sum_{u,v \in I}(d(u,v)+1)w(u,v)\ ] + D_I * W_I$

(D_I distances of nodes in I from root of T_I)

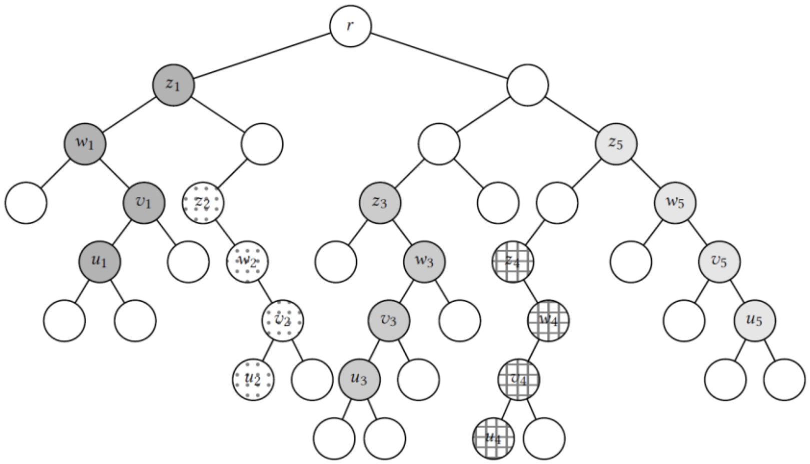
3. Dynamic program over intervals

Choose optimal root and add dist to root



SplayNet: Properties

Property 2: Provides amortized cost and amortized throughput guarantees



Rotations can happen concurrently: independent clusters

Splay tree: requests one after another

	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓	-	-	-	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓	-	-	-	-	-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-

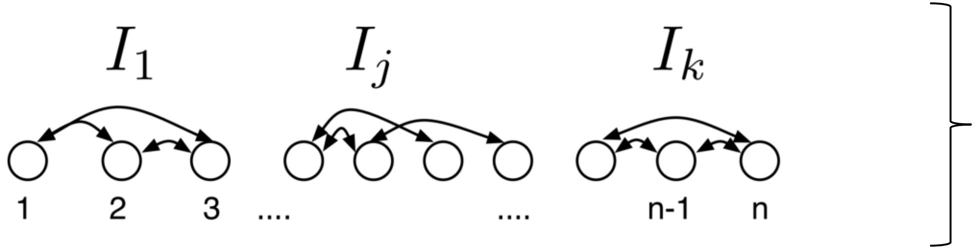
SplayNet: concurrent

	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

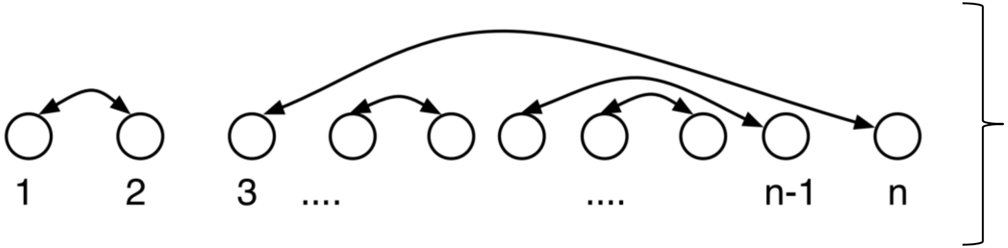
Analysis more challenging: potential function sum no longer telescopic. One request can "push-down" another.

SplayNet: Properties

Property 3: Converges to optimal network under specific demands



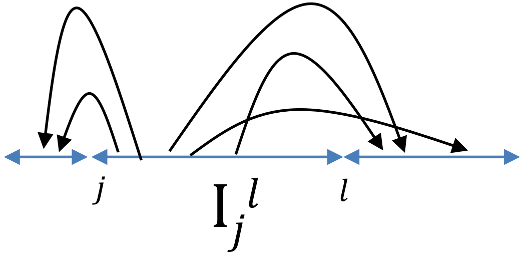
Cluster scenario: SplayNet will converge to state where path between cluster nodes only includes cluster nodes



Non-crossing matching scenario: SplayNet will converge to state where all communication pairs are adjacent

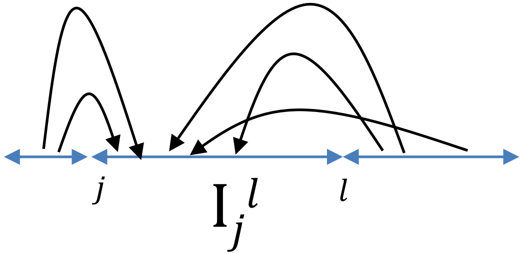
SplayNet: Improved Lower Bounds

Interval Cuts Bound



$\mathrm{cut}_{\mathrm{out}}(I_j^{\,l})$

$\mathrm{cut}_{\mathrm{in}}(I_j^{\,l})$



$$\mathrm{Cost} = \Omega(\max_i \min_{j,l} H(\mathrm{cut}_{\mathrm{in}}(I_j^{\,l})))$$

$$\mathrm{Cost} = \Omega(\max_i \min_{j,l} H(\mathrm{cut}_{\mathrm{out}}(I_j^{\,l})))$$

Edge Expansion Bound

- Let cut W(S) be weight of edges in cut (S,S') for a given S
- Define a distribution w_S (u) according to the weights to all possible nodes v:

$$w_S(u) = \sum_{\substack{(u,v)\in E(S,\bar{S}) \\ u\in S}} w(u,v)/W(S)$$

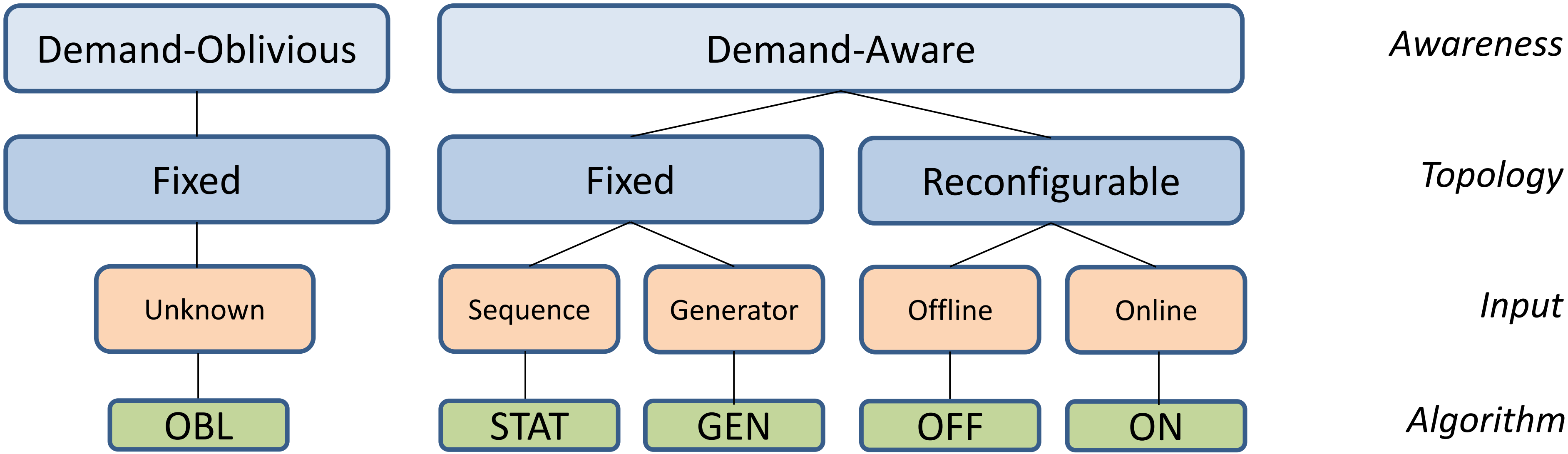
- Define entropy of cut and src(S),dst(S) distributions accordingly: :

$$\varphi_H(S) = W(S)\left(H(\mathrm{src}(S)) + H(\mathrm{dst}(S))\right)$$

- Conductance entropy is lower bound:

$$\Omega(\phi_H(\mathcal{R}(\sigma)))$$

Uncharted Space



Can compare to static
or dynamic baseline!

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **ArXiv** 2018.

Roadmap

- Networks are increasingly ***complex***: a case for **formal methods**?
- Networks are increasingly ***flexible***: a case for **self-adjusting networks**?

Thank you!

Further Reading

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.
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.
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.
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.

See also references on slides!