Algorithms for Flexible Networks: Opportunities and Challenges

Stefan Schmid (Uni Vienna)

A Great Time to Be a Networking Researcher!

Rhone and Arve Rivers, Switzerland

Credits: George Varghese.

Flexibilities: Along 3 Dimensions

Passau, Germany

Inn, Donau, Ilz

Flexibilities: Along 3 Dimensions

Passau, Germany

Inn, Donau, Ilz

Flexibilities: Along 3 Dimensions

Enabler:
SDN

Enabler:
Virtualization

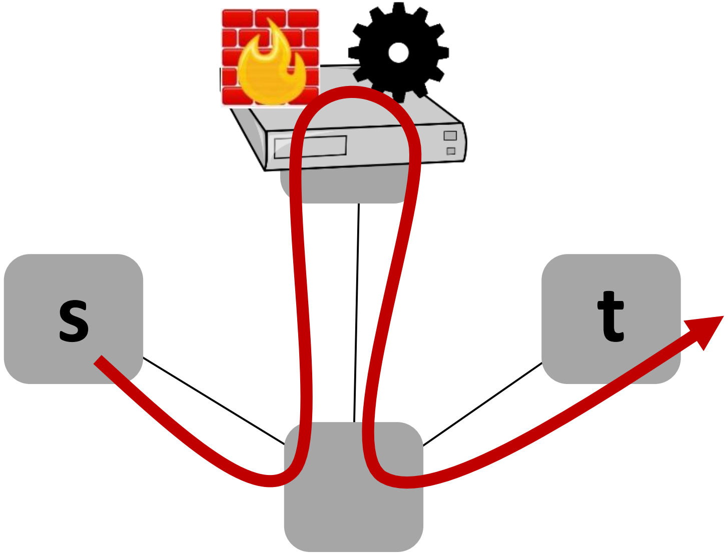
Enabler:
Optics

u, Germany

Inn, Donau, Ilz

Opportunity: Flexible Routing

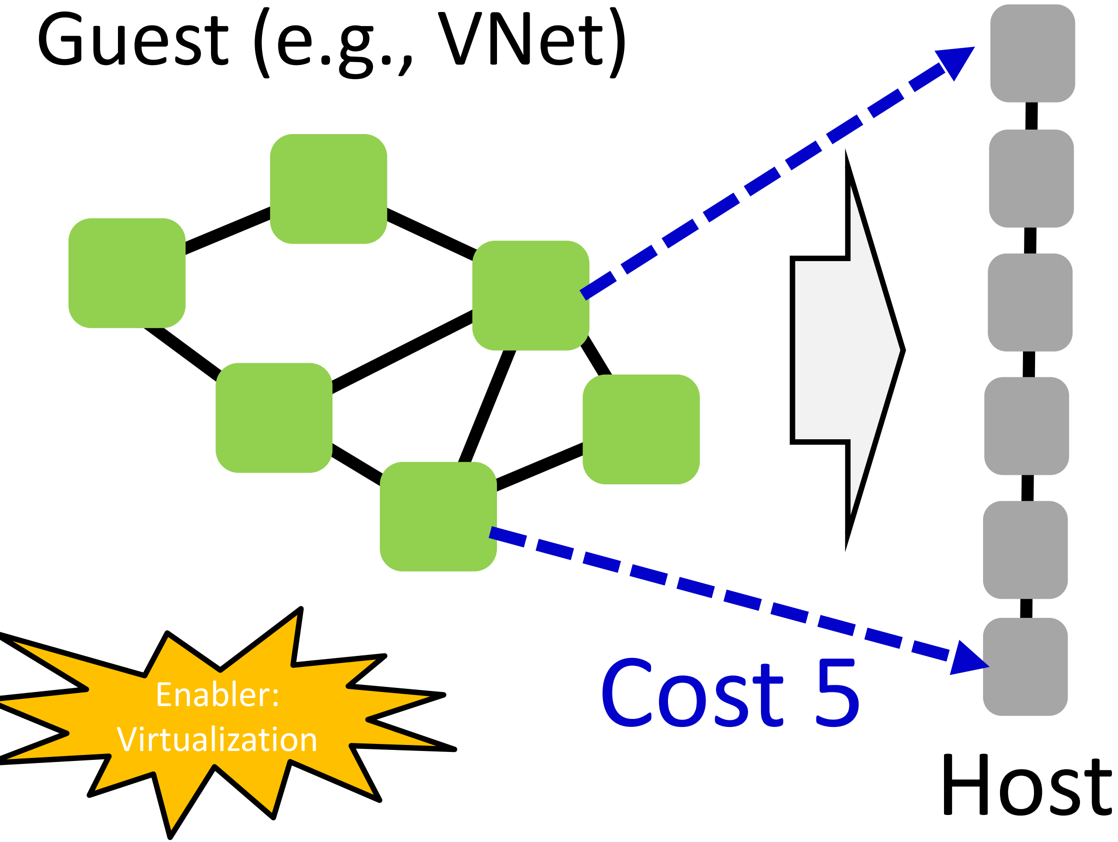
- **Direct control** over paths
- **Generalized** match-action
- Composing **innovative services**
- Not even simple paths: **walks**!



Enabler: SDN

Charting the Algorithmic Complexity of Waypoint Routing. Amiri et al. ACM SIGCOMM CCR, 2018.

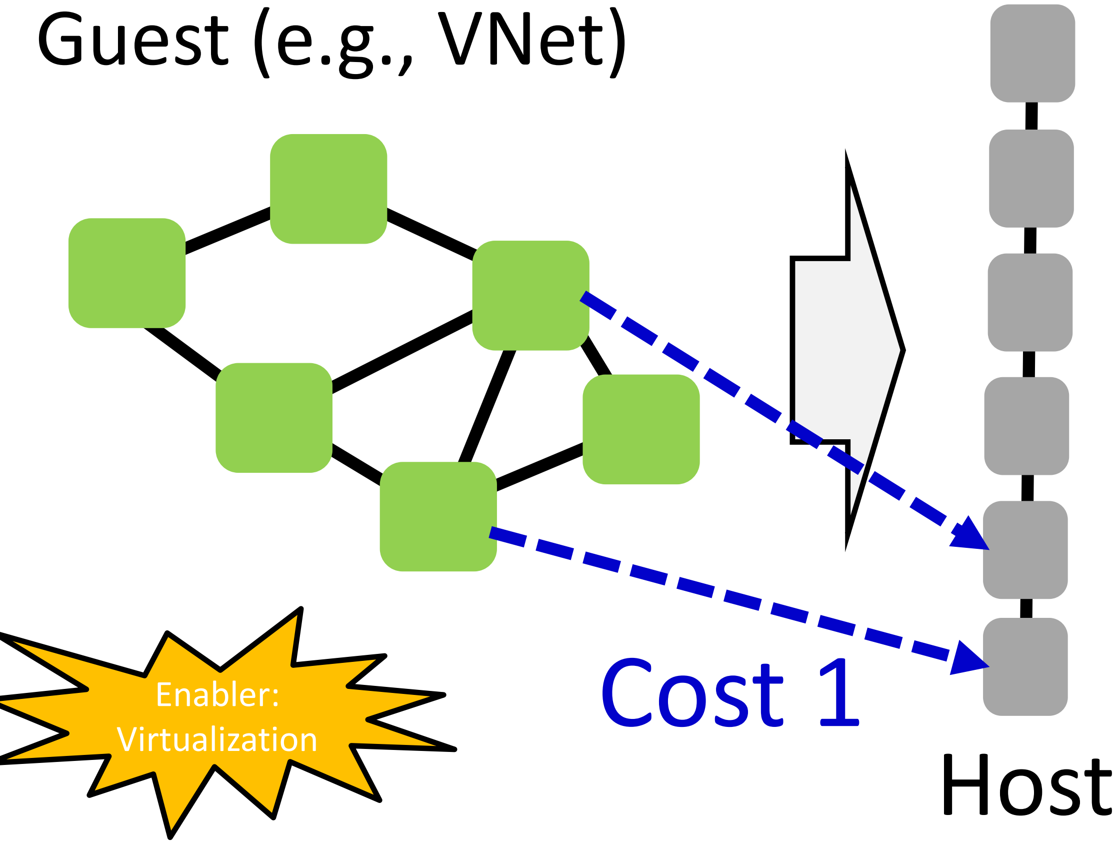
Opportunity: Flexible Embedding



- Improved **resource allocation**
- Minimize communication paths: lower **latency**, load, etc.

Charting the Complexity Landscape of Virtual Network Embeddings. Rost et al. IFIP Networking, 2018.

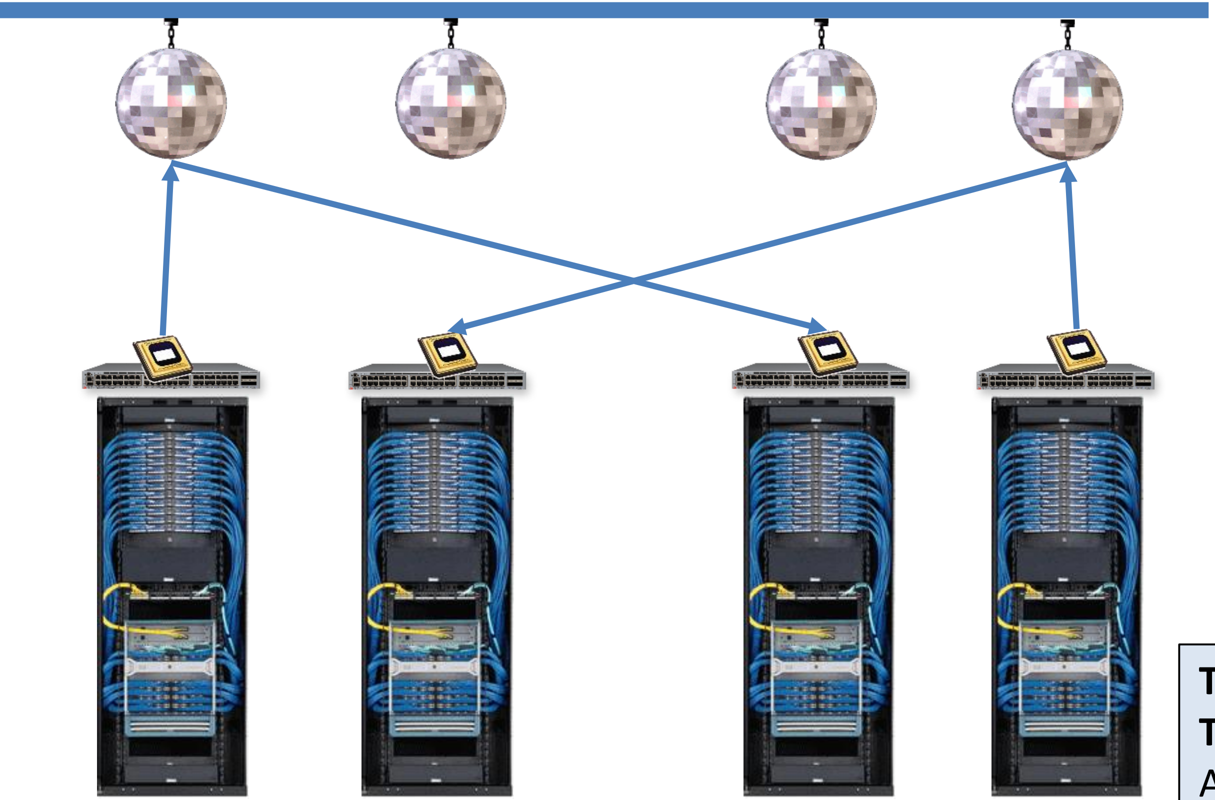
Opportunity: Flexible Embedding



- Improved **resource allocation**
- Minimize communication paths: lower **latency**, load, etc.

Charting the Complexity Landscape of Virtual Network Embeddings. Rost et al. IFIP Networking, 2018.

Opportunity: Flexible Topology Programming

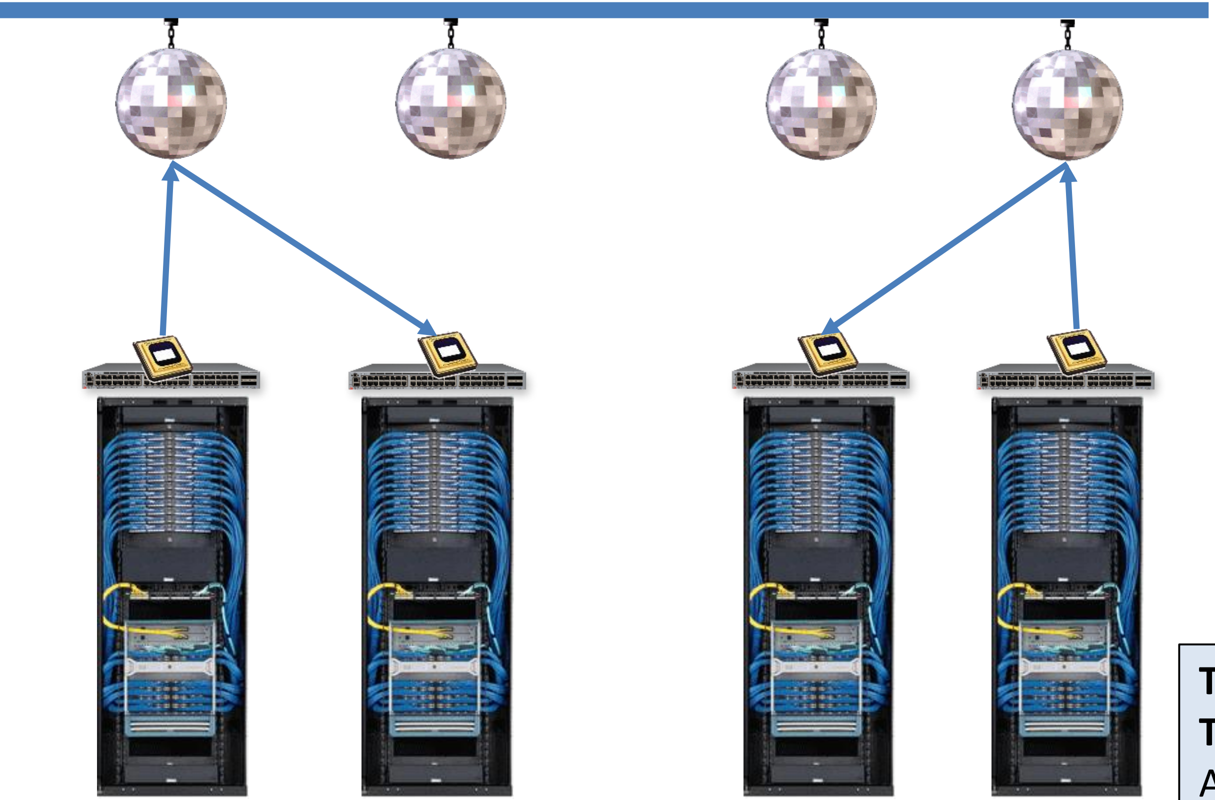


- **Reconfigure** networks towards needs

Enabler: Free-space optics

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

Opportunity: Flexible 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.

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)

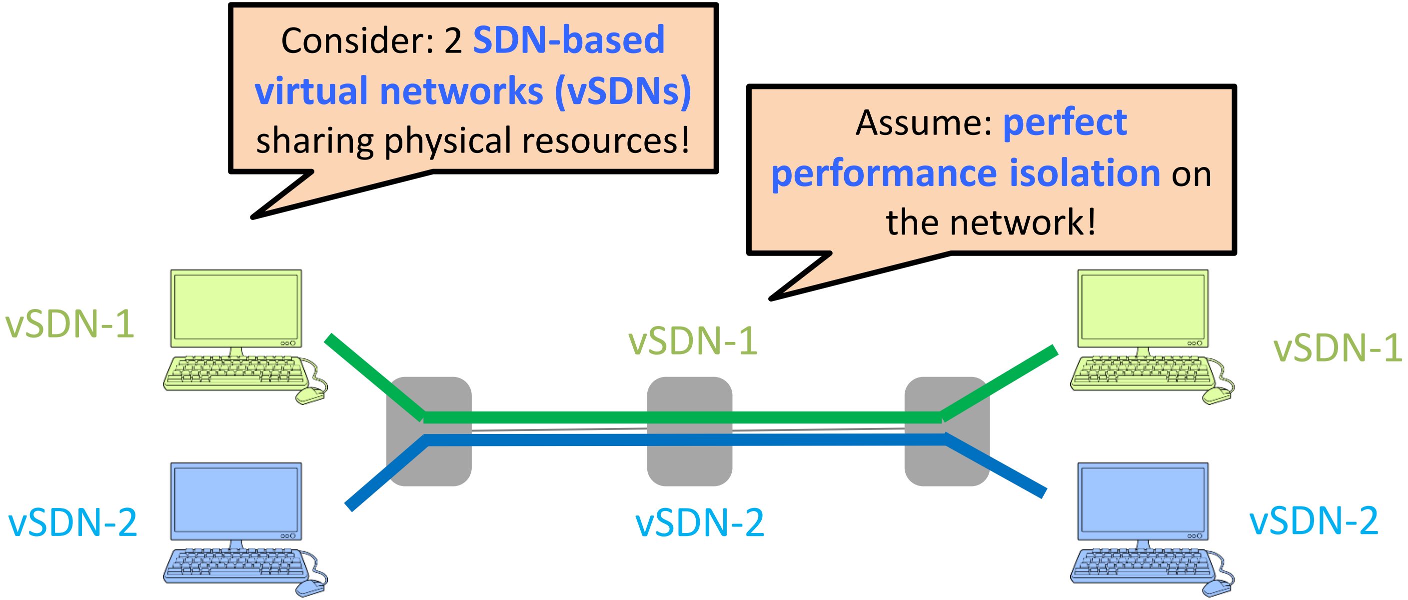
Another Challenge: Model vs Reality ☺

You: **I invented a great new algorithm to route and embed service chains at low resource cost and providing minimal bandwidth guarantees!**

Boss: **So can I promise our customers a predictable performance?**

You: **hmm…**

Predictable Performance in SDNs?

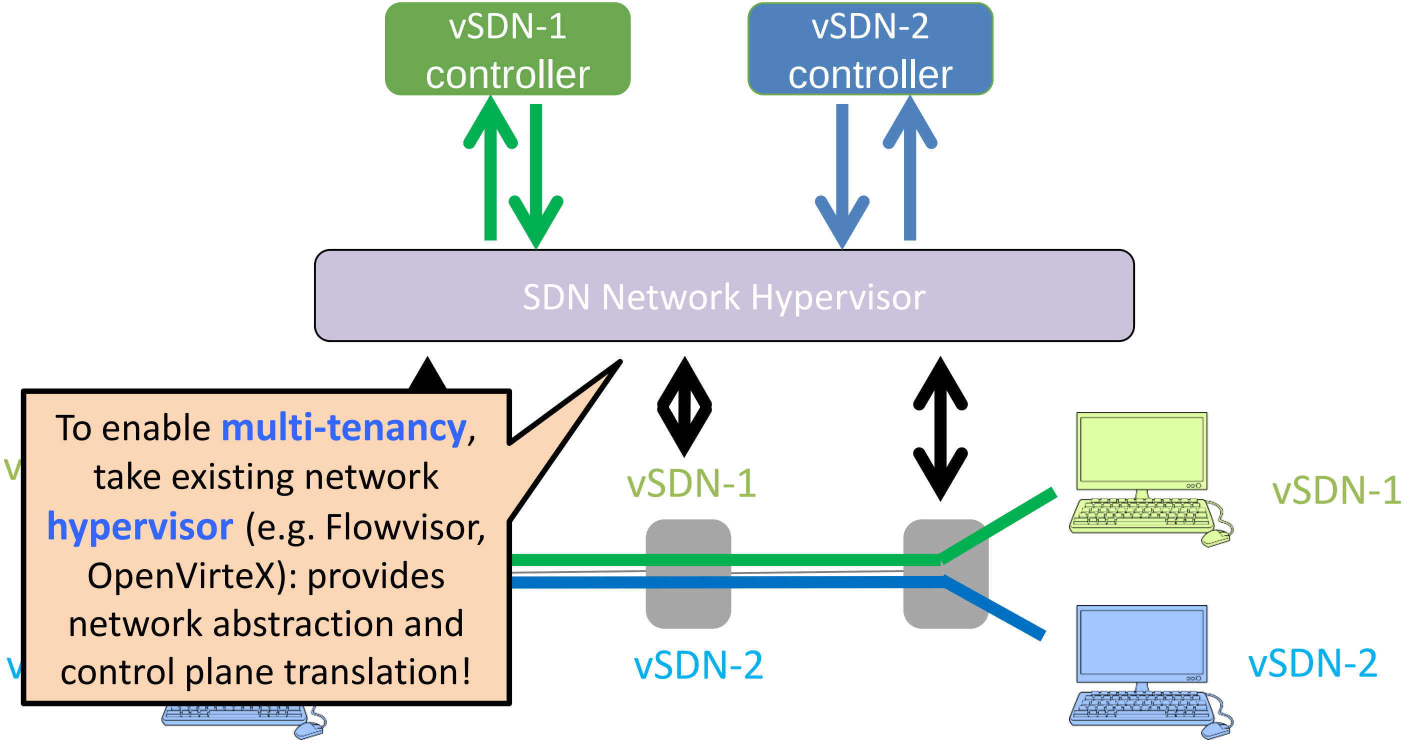


Consider: 2 **SDN-based virtual networks (vSDNs)** sharing physical resources!

Assume: **perfect performance isolation** on the network!

An Experiment: 2 vSDNs with bw guarantee!

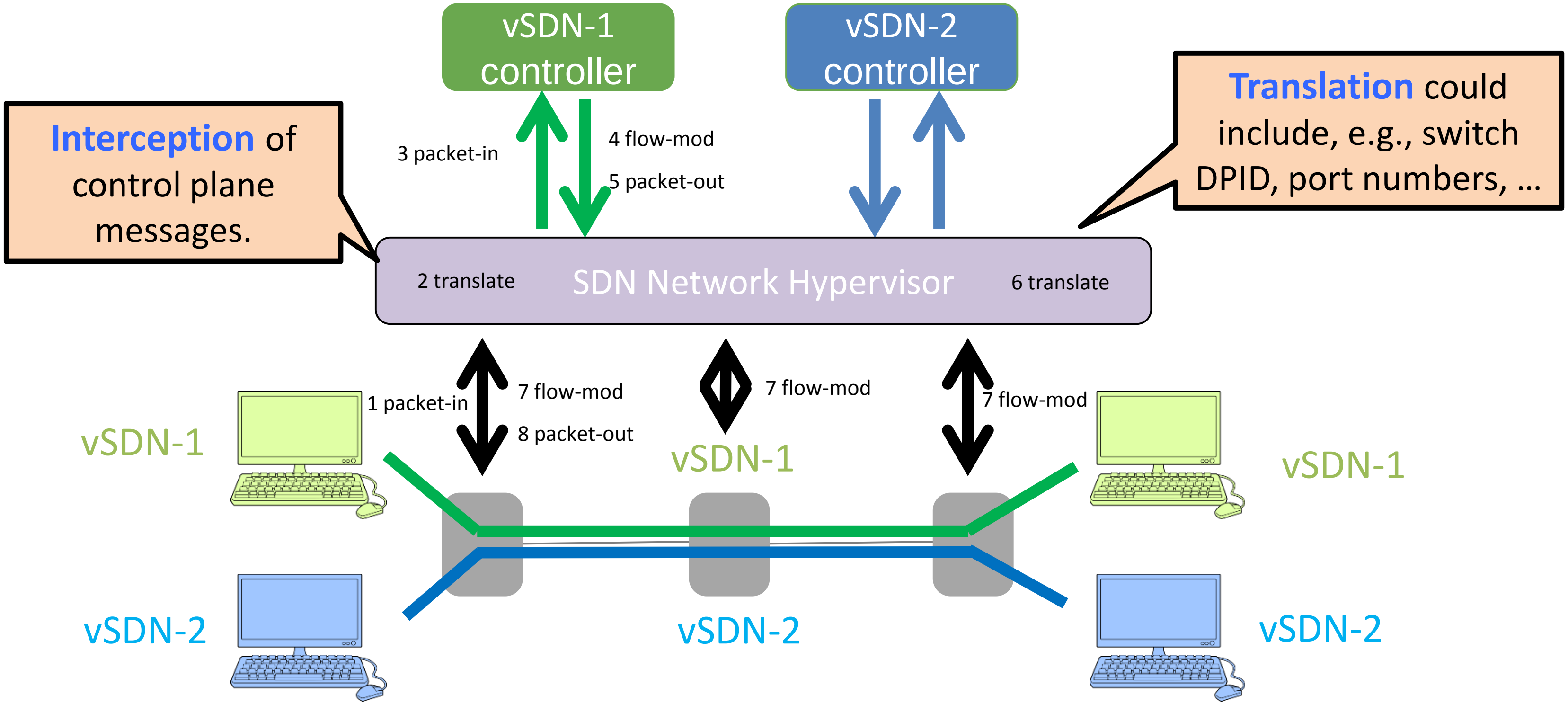
Predictable Performance in SDNs?



To enable **multi-tenancy**, take existing network **hypervisor** (e.g. Flowvisor, OpenVirteX): provides network abstraction and control plane translation!

An Experiment: 2 vSDNs with bw guarantee!

Predictable Performance in SDNs?



Interception of control plane messages.

Translation could include, e.g., switch DPID, port numbers, …

vSDN-1 controller

vSDN-2 controller

3 packet-in

4 flow-mod

5 packet-out

2 translate

SDN Network Hypervisor

6 translate

1 packet-in

7 flow-mod

8 packet-out

7 flow-mod

7 flow-mod

vSDN-1

vSDN-1

vSDN-1

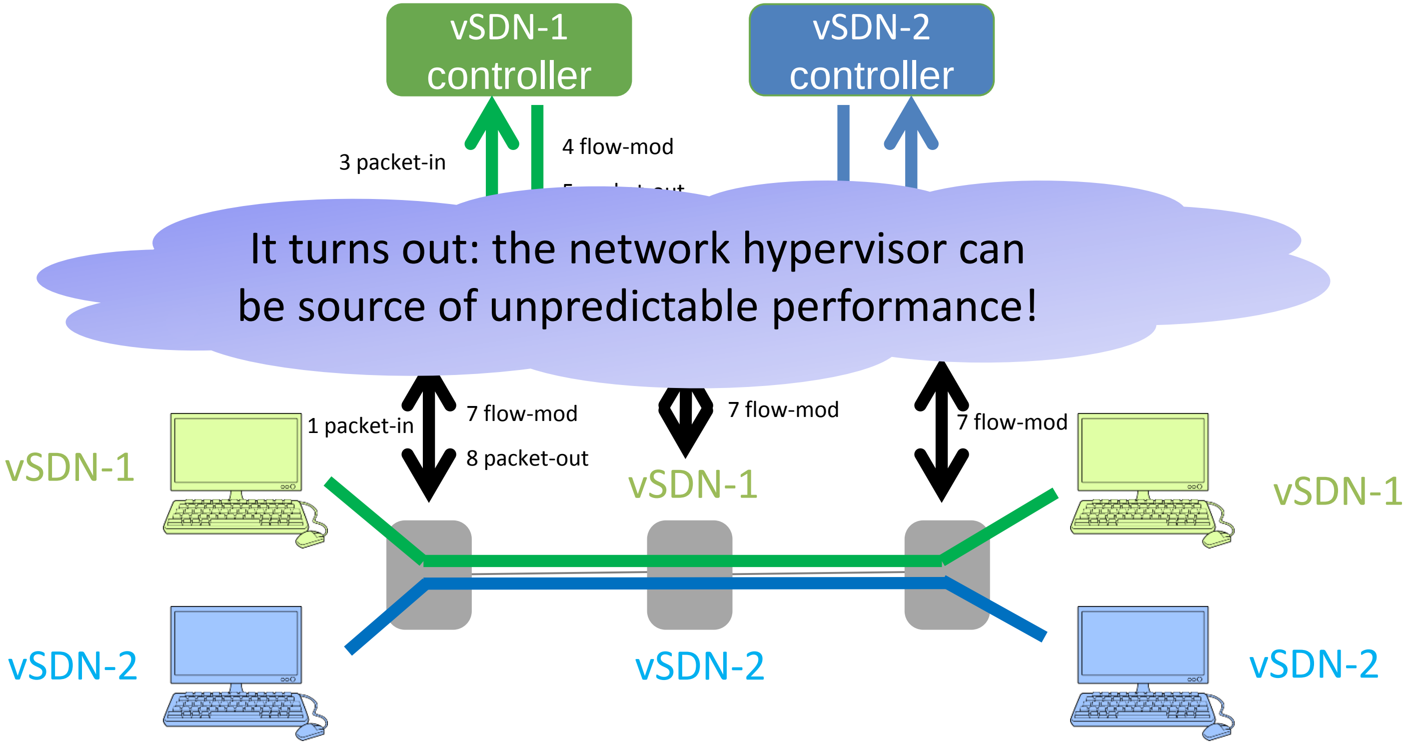
vSDN-2

vSDN-2

vSDN-2

An Experiment: 2 vSDNs with bw guarantee!

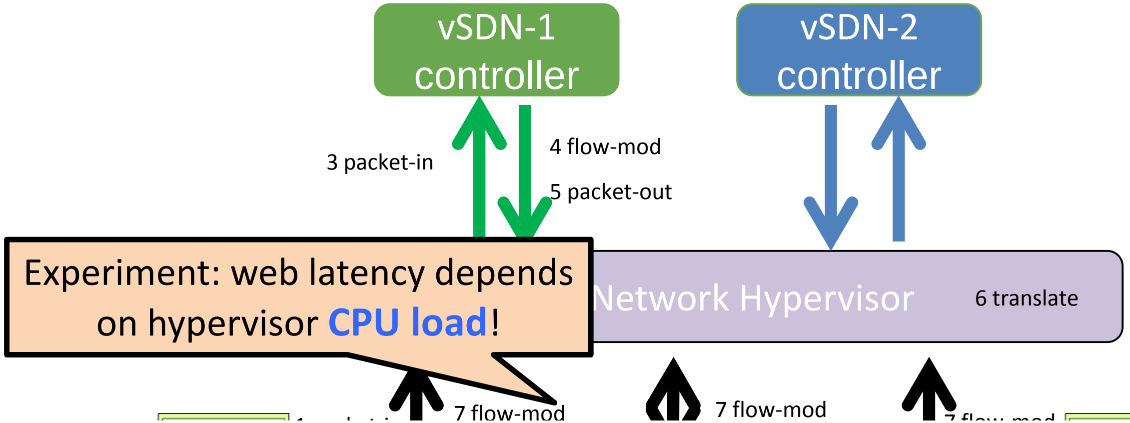
Predictable Performance in SDNs?



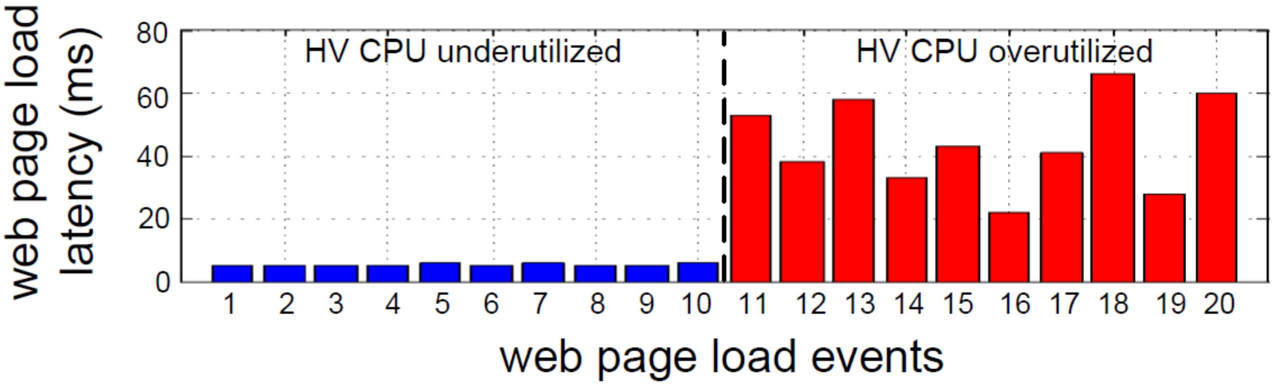
It turns out: the network hypervisor can be source of unpredictable performance!

An Experiment: 2 vSDNs with bw guarantee!

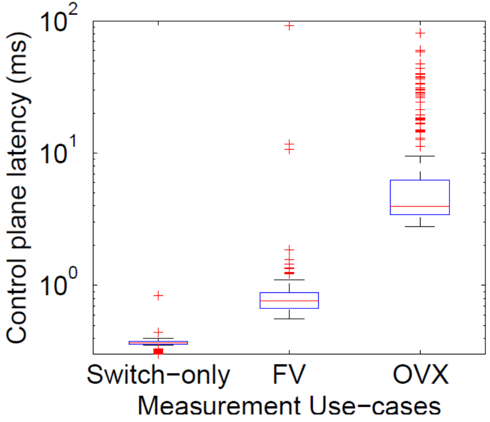
Predictable Performance in SDNs?



Experiment: web latency depends on hypervisor **CPU load**!



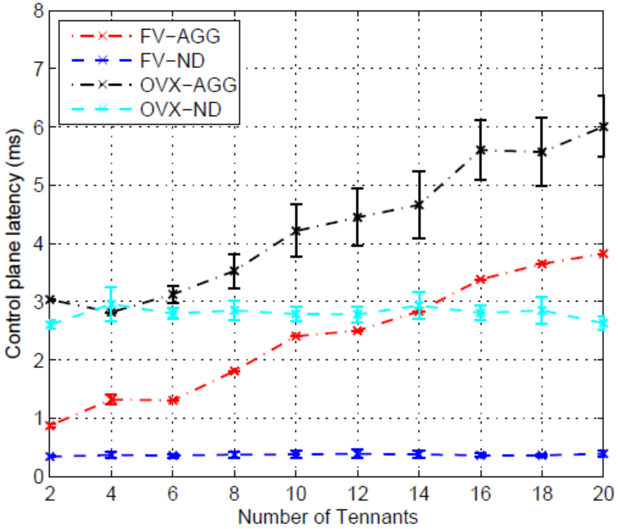
Predictable Performance in SDNs?



Performance also depends on hypervisor type…

(multithreaded or not, which version of Nagle's algorithm, etc.)

… number of tenants…



On The Impact of the Network Hypervisor on Virtual Network Performance. Blenk et al. *IFIP Networking*, 2019.

First Conclusions

- Exploiting network flexibilities is non-trivial, especially if **fine-grained** and **fast** reactions are desired
- Also modelling such networked systems is challenging: details of **interference**, **demand**, etc. will only be available at runtime

Optimal operation of flexible networks ***too complex for humans***.

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

Self-observing, self-adjusting, self-repairing, self-driving, …

It’s about automation!

Roadmap

- Opportunities of self-* networks
  - Example 1: Demand-aware, self-adjusting networks
  - Example 2: Self-repairing networks
- Challenges of desinging self-* networks

Roadmap

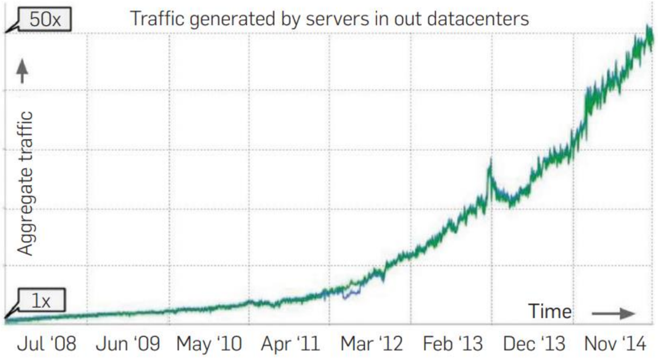
- Opportunities of self-* networks
  - **Example 1: Demand-aware, self-adjusting networks**
  - Example 2: Self-repairing networks
- Challenges of desinging self-* networks

Why Demand-Aware…?

Case study: data-center networks

Explosive Growth of Demand…

Batch processing, web services, **distributed ML**, …: ***data-centric applications*** are distributed and interconnecting network is ***critical***

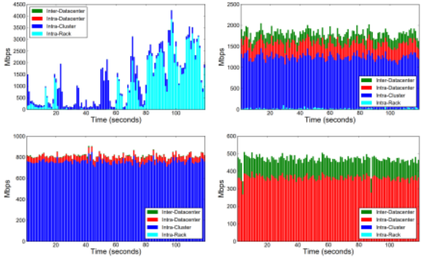


Source: Jupiter Rising. SIGCOMM 2015.

Aggregate server traffic in **Google's datacenter fleet**

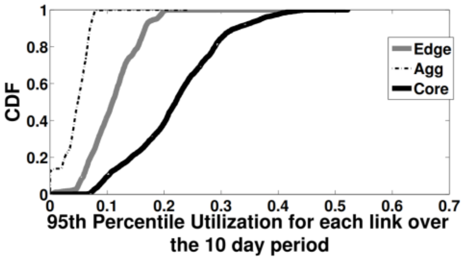
… But Much Structure!

Facebook

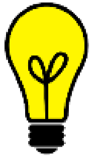


Inside the Social Network's (Datacenter) Network @ SIGCOMM 2015

Benson et al.

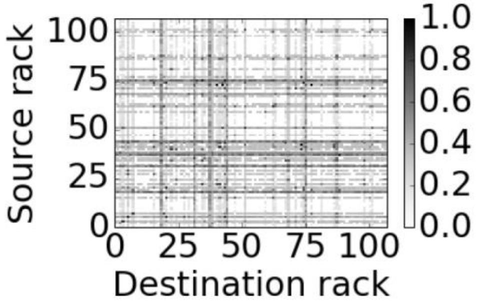


Understanding Data Center Traffic Characteristics @ WREN 2009



Spatial (***sparse!***) and temporal **locality**

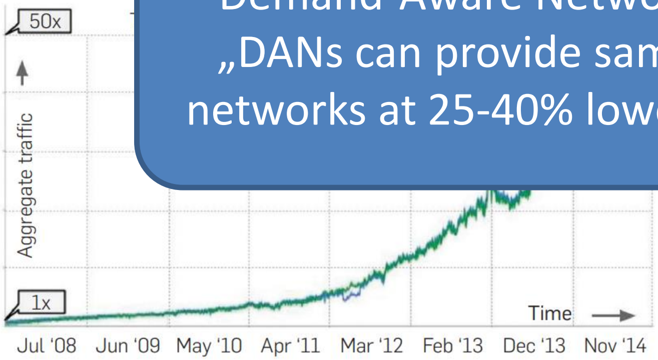
Microsoft



ProjecToR @ SIGCOMM 2016

Explosive Growth of Demand…

Batch processing, web services, **distributed ML**, …: ***data-centric applications*** are distributed and interconnecting network is ***critical***

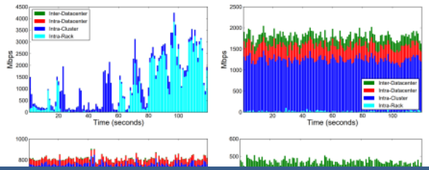


Source: Jupiter Rising. SIGCOMM 2015.

Aggregate server traffic in **Google's datacenter fleet**

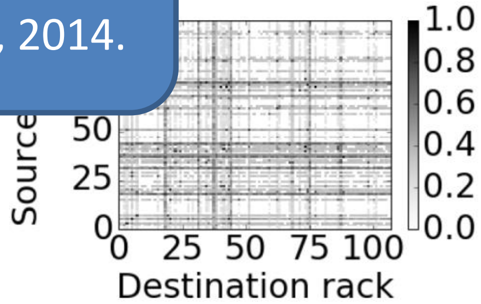
… But Much Structure!

Facebook

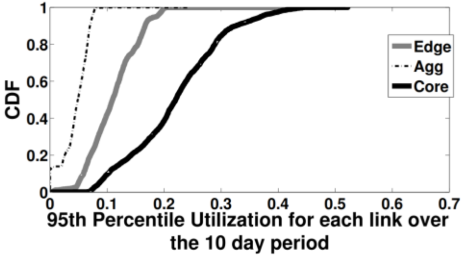


ial (***sparse!***) and
oral **locality**

Microsoft



ProjecToR @ SIGCOMM 2016

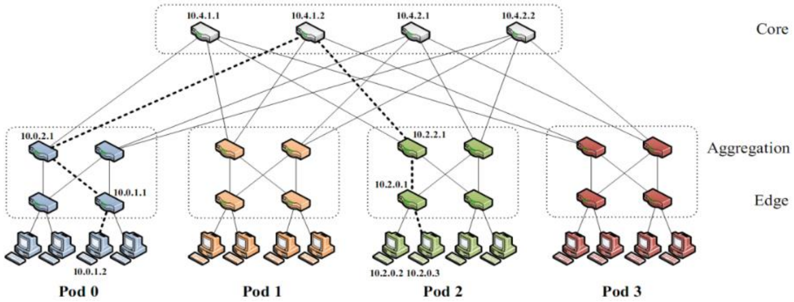


Understanding Data Center Traffic Characteristics @ WREN 2009

Demand-Aware Networks (DANs) can exploit this structure: „DANs can provide same performance as demand-oblivious networks at 25-40% lower costs.“ Firefly, SIGCOMM CCR, 2014.

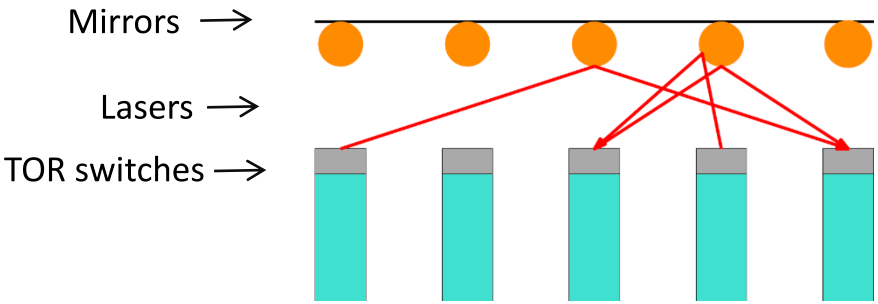
Example: Demand-Aware Topology

Traditional datacenter network



- Usually optimized ***for the "worst-case"*** (**all-to-all** communication)
- Example, fat-tree topologies: provide **full bisection bandwidth**

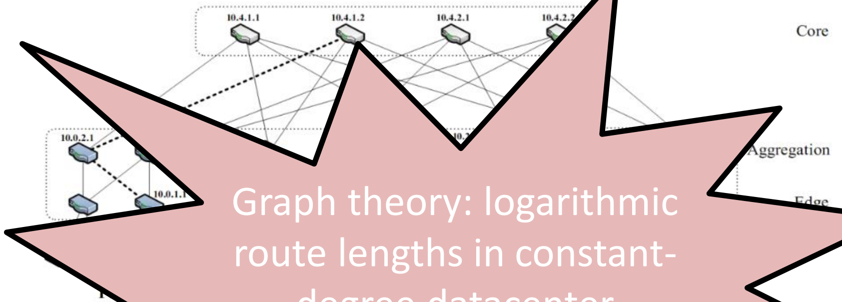
Reconfiguable datacenter network



- Optimized ***toward the workload*** it serves (e.g., **route length**)
- Statically or ***even dynamically***

Example: Demand-Aware Topology

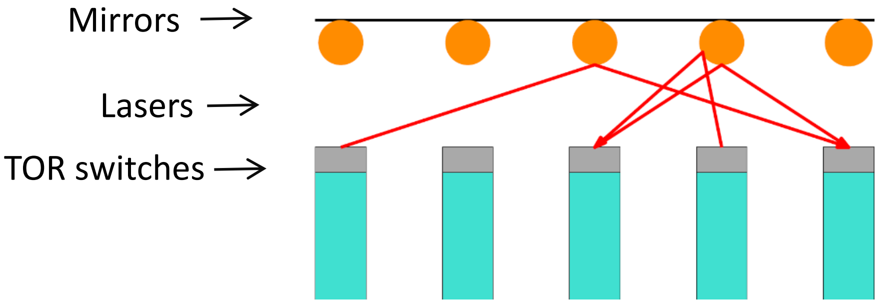
Traditional datacenter network



Graph theory: logarithmic route lengths in constant-degree datacenter networks.

- Usual ... *e* ... *st-case* (all-t... co...nicatio...
- Example, fa... tree topologies: provide **full bisection bandwidth**

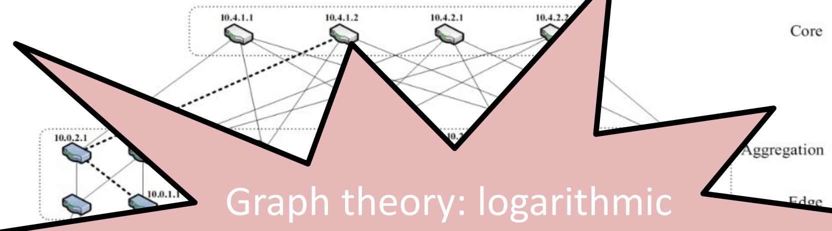
Reconfiguable datacenter network



- Optimized ***toward the workload*** it serves (e.g., **route length**)
- Statically or ***even dynamically***

Example: Demand-Aware Topology

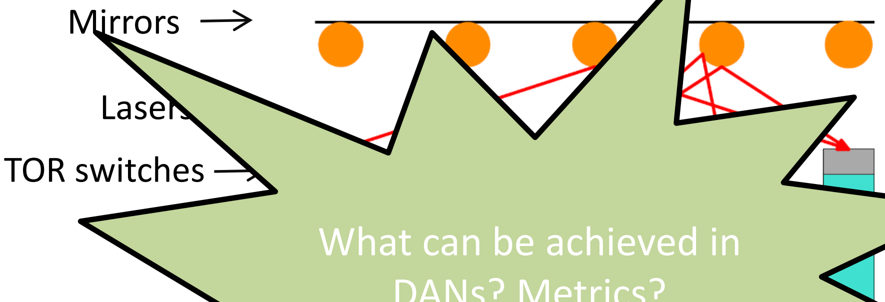
Traditional datacenter network



Graph theory: logarithmic route lengths in constant-degree datacenter networks.

- Usual … *e* … *st-case* (all-t… co… nicatio…
- Example, fa… tree topologies: provide **full bisection bandwidth**

Reconfiguable datacenter network



What can be achieved in DANs? Metrics?

- Opti… ed … *ar* … *work* … *d* it serves (e.g … **oute le … th**)
- Statically or ***even dynamically***

DAN Design: New Types of Problems

Input: Workload

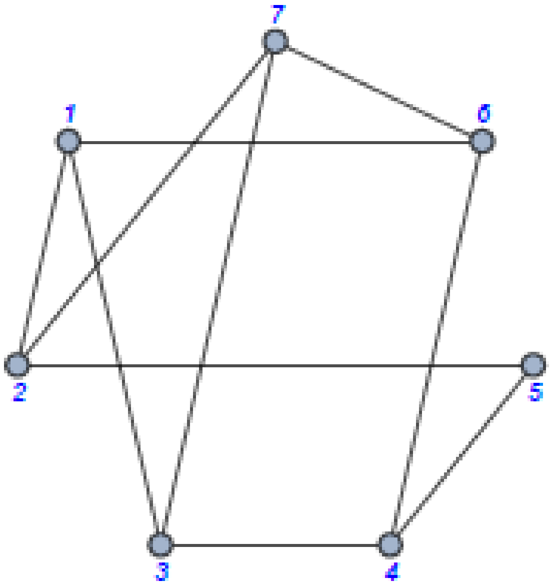
Destinations

Sources

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

design

Output: DAN



Demand matrix: joint distribution

… of ***constant degree*** (**scalability**)

DAN Design: New Types of Problems

Input: Workload

Destinations

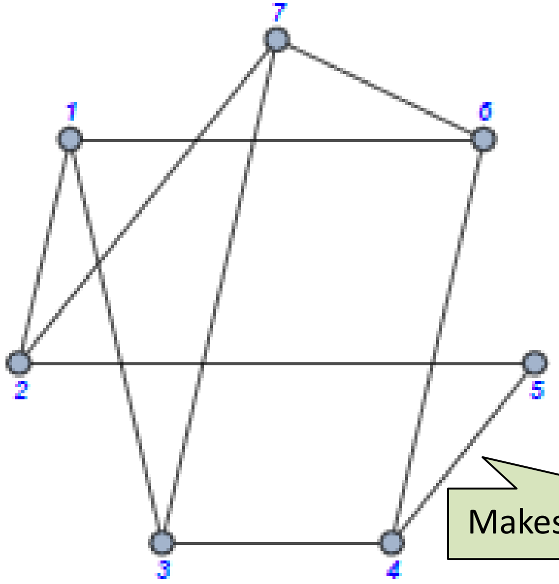
Sources

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

Much from 4 to 5.

design

Output: DAN



Makes sense to add link!

Demand matrix: joint distribution

… of ***constant degree*** (**scalability**)

DAN Design: New Types of Problems

Input: Workload

Destinations

1 communicates to many.

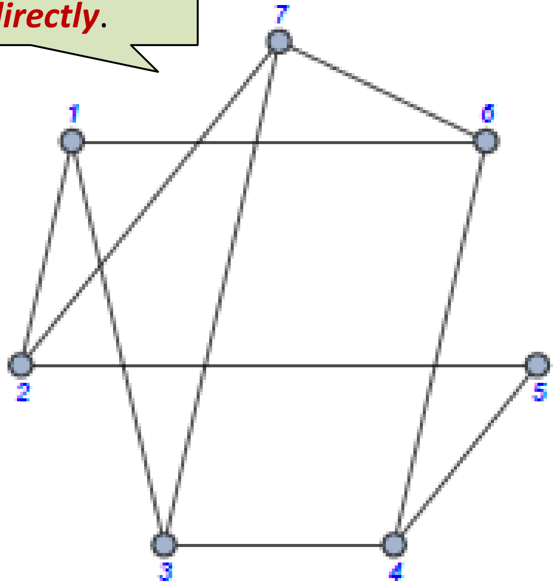
Sources

						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

Output: DAN

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



Demand matrix: joint distribution

… of ***constant degree*** (**scalability**)

DAN Design: New Types of Problems

Input: Workload

Destinations

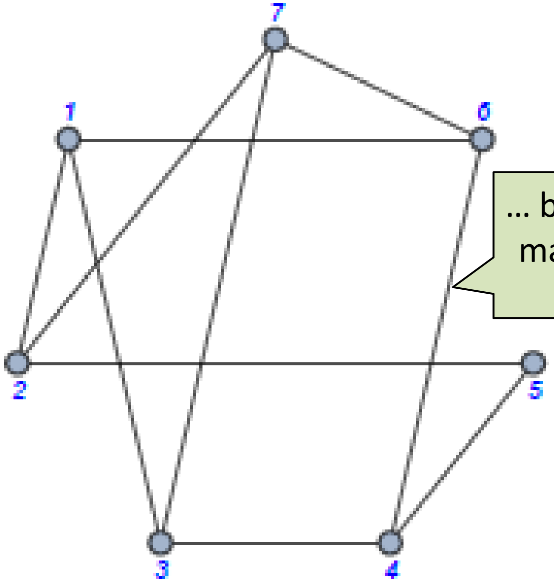
Sources

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

4 and 6 don’t communicate…

design

Output: DAN



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

Demand matrix: joint distribution

… of ***constant degree*** (**scalability**)

More Formally: DAN Design Problem

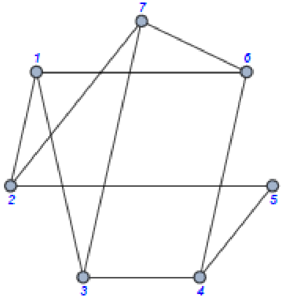
Input:

$\mathcal{D}[\mathbf{p(i,j)}]$: joint **distribution**, **Δ**

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

Output:

N: DAN



Bounded degree
Δ=3

Objective:

Expected Path Length (EPL):
Demand-weighted route length

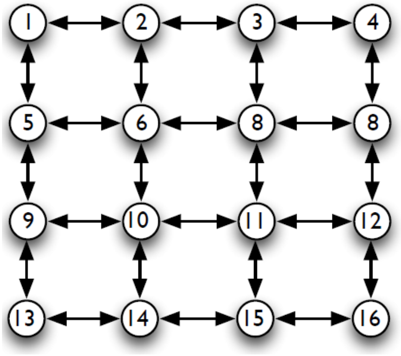
$$\mathrm{EPL}(\mathcal{D},\mathrm{N}) = \sum_{(\mathrm{u},\mathrm{v})\in\mathcal{D}} \mathrm{p}(\mathrm{u},\mathrm{v}) \cdot \mathrm{d}_{\mathrm{N}}(\mathrm{u},\mathrm{v})$$

Path length ***on DAN N***.

Frequency

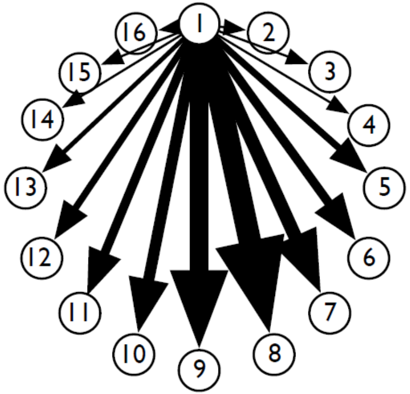
Sometimes, DANs can be much better!

Example 1: **low-degree demand**



- Already low degree: degree-4 DAN can serve this ***at cost 1*** .

Example 2: high-degree but **skewed demand**



- If sufficiently skewed: constant-degree DAN can serve it at cost ***O(1)***

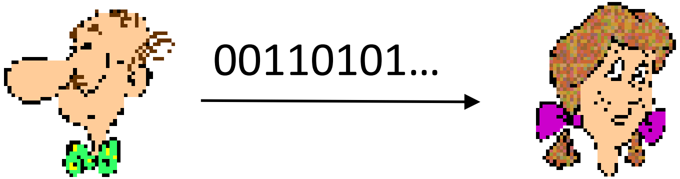
So on what does it depend?

So on what does it depend?

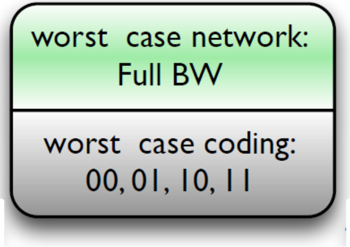
We argue (but still don‘t know!): on the **“entropy” of the demand**!

An Analogy to Coding

„Coming to Wroclaw?“



if demand ***arbitrary*** and ***unknown***



worst case network:
Full BW

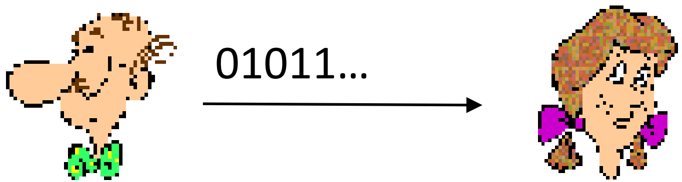
worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

An Analogy to Coding

„Coming to Wroclaw?“



if demand ***arbitrary*** and ***unknown***

worst case network:
Full BW

log diameter

worst case coding:
00, 01, 10, 11

log # bits / symbol

DAN!

if demand ***known*** and ***fixed***

entropy?

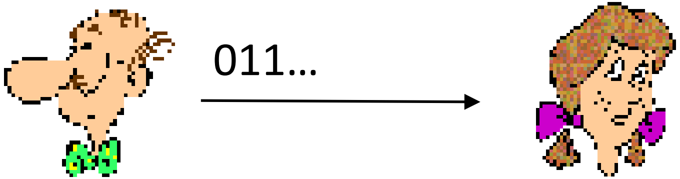
static
Demand-Aware Nets

entropy / symbol

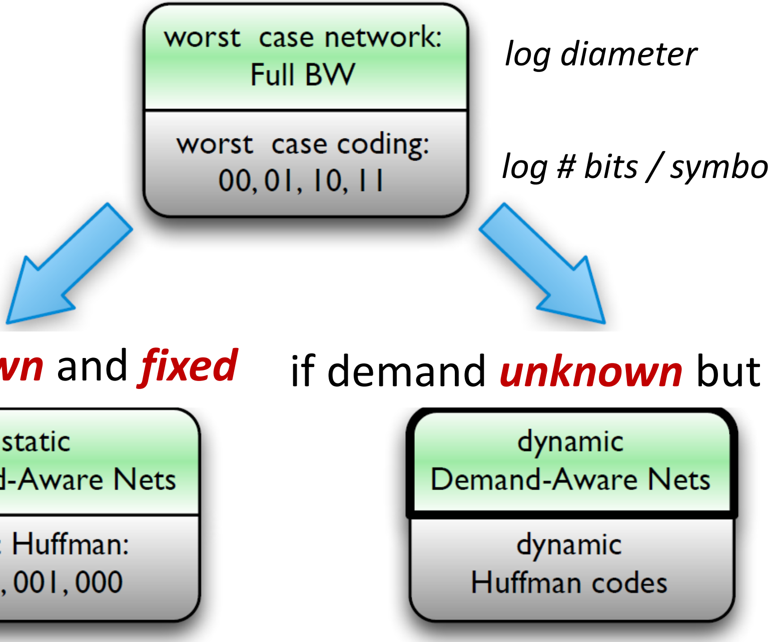
static Huffman:
1, 01, 001, 000

An Analogy to Coding

„Coming to Wroclaw?“



if demand ***arbitrary*** and ***unknown***



worst case network:
Full BW

worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!

DAN!

SAN!

if demand ***known*** and ***fixed***

if demand ***unknown*** but ***reconfigurable***

entropy?

entropy / symbol

static
Demand-Aware Nets

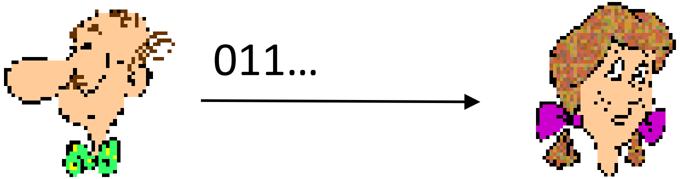
static Huffman:
1, 01, 001, 000

dynamic
Demand-Aware Nets

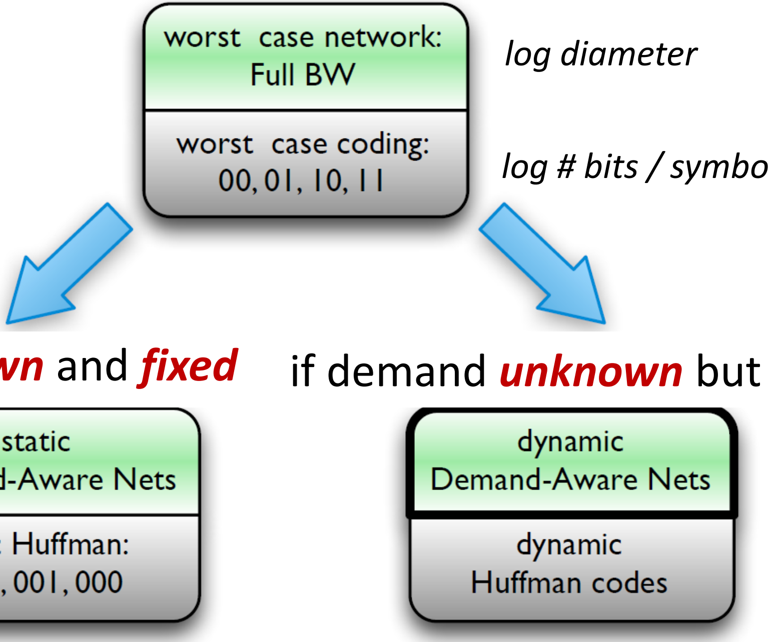
dynamic
Huffman codes

An Analogy to Coding

„Coming to Wroclaw?“



if demand ***arbitrary*** and ***unknown***



worst case network:
Full BW

worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!

DAN!

SAN!

if demand ***known*** and ***fixed***

if demand ***unknown*** but ***reconfigurable***

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

dynamic
Demand-Aware Nets

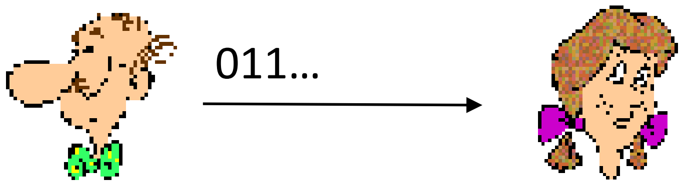
dynamic
Huffman codes

Can exploit
spatial locality!

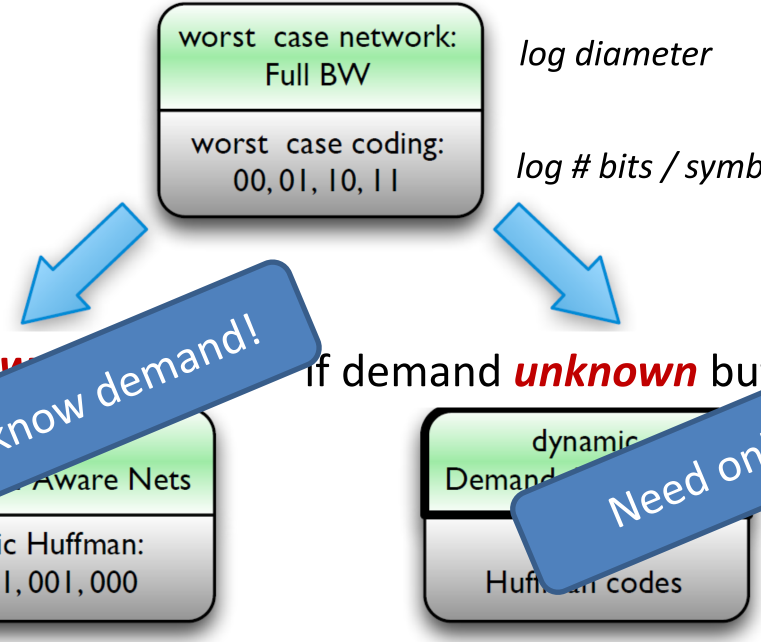
Additionally exploit
temporal locality!

An Analogy to Coding

„Coming to Wroclaw?“



if demand ***arbitrary*** and ***unknown***



worst case network:
Full BW

log diameter

worst case coding:
00, 01, 10, 11

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!

DAN!

SAN!

if demand ***know***

if demand ***unknown*** but ***r***

Can exploit
spati

Aware Nets

static Huffman:
1, 01, 001, 000

dynamic
Demand

Huf codes

„Cheating“: need to know demand!

Need online algorithms!

Additionally exploit
temporal locality!

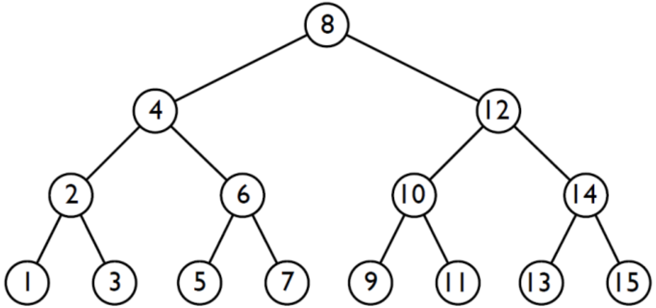
Analogous to *Datastructures*: Oblivious…

- 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

Corresponds to ***max possible demand***!



Many requests for leaf 1…

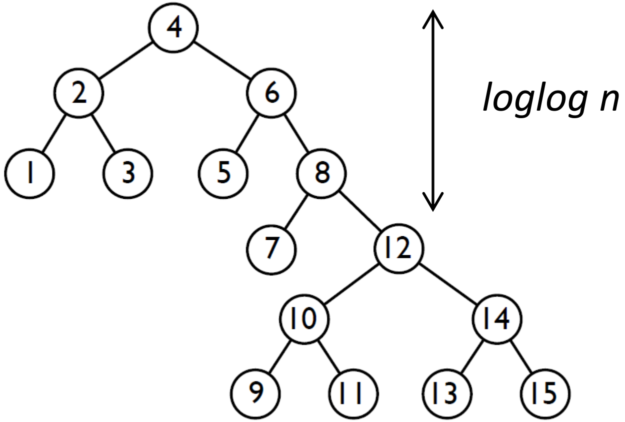
… then for leaf 3…

… Demand-Aware …

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

- E.g.: place frequently accessed elements close to the root

- E.g., **Knuth/Mehlhorn/Tarjan** trees



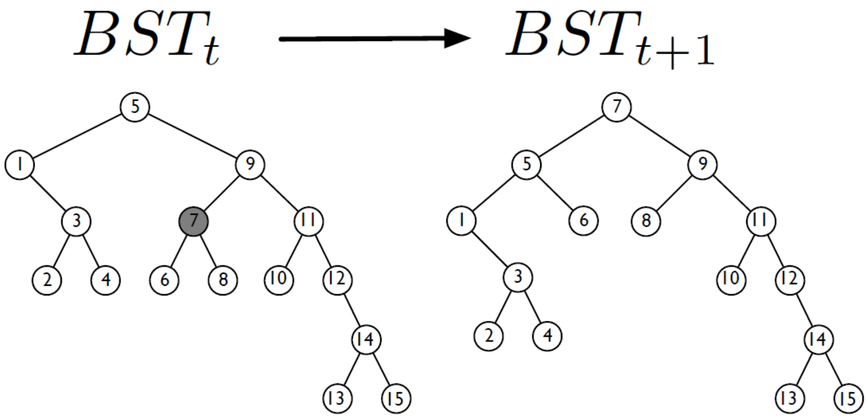
- Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
  - Amortized cost ***O(loglog n)***

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

… Self-Adjusting!

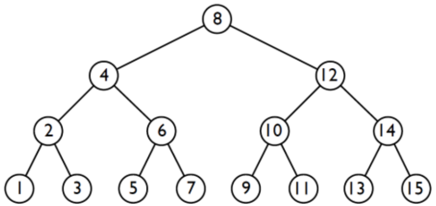
- **Demand-aware reconfigurable** BSTs can additionally take advantage of ***temporal locality***

- 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)



Datastructures

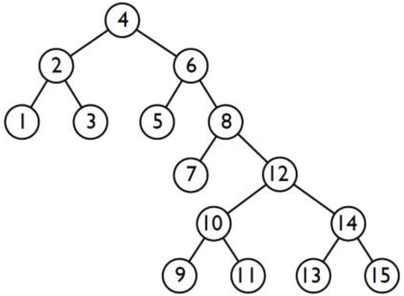
Oblivious



Lookup

O(log n)

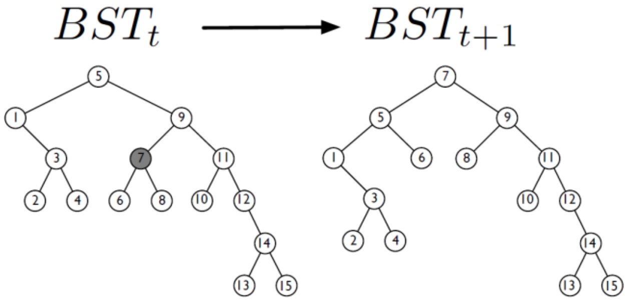
Demand-Aware



Exploit **spatial locality**:

empirical entropy O(loglog n)

Self-Adjusting

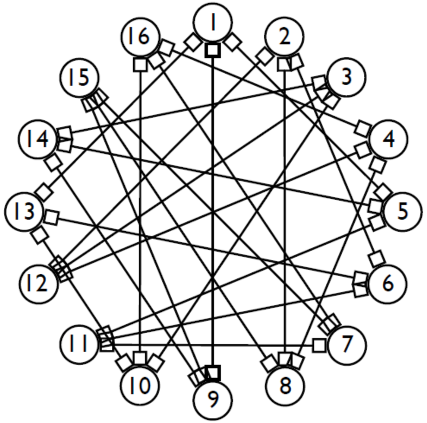


Exploit **temporal locality** as well:

O(1)

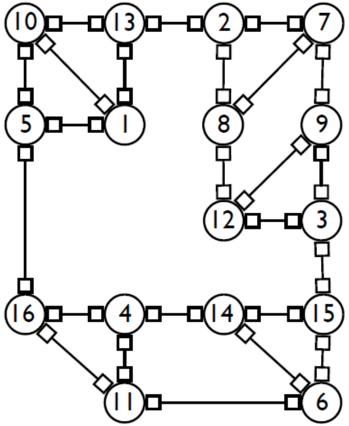
Analogously for Networks

Oblivious



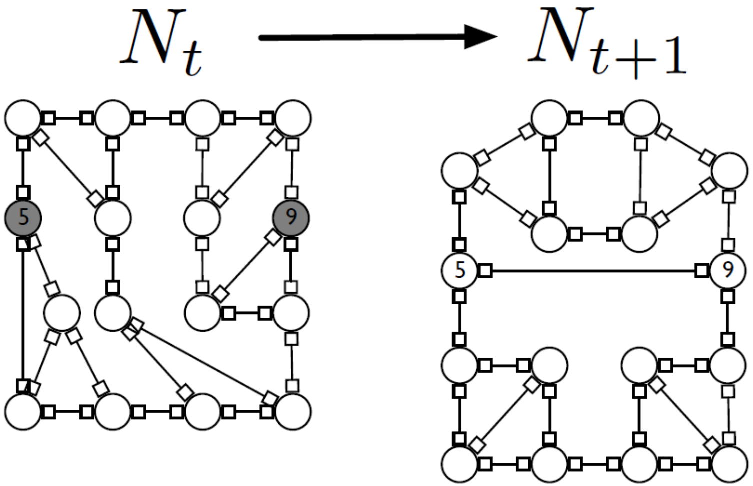
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**

SAN



Exploit **temporal locality** as well

Avin, S.: Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **SIGCOMM CCR** 2018.

Intuition: Entropy Lower Bound

Lower Bound Idea:
Leverage Coding or Datastructure

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

- DAN just for a ***single (source) node 1***: cannot do better than Δ-ary **Huffman tree** for its destinations

- How good can this tree be?

Entropy lower bound on EPL known for binary trees, e.g. ***Mehlhorn*** 1975 for BST

Lower Bound Idea:
Leverage Coding or Datastructure

An optimal “**ego-tree**“ for this source!

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

- DAN just for a ***single (source) node 1***: cannot do better than Δ-ary **Huffman tree** for its destinations

- How good can this tree be?

Entropy lower bound on EPL known for binary trees, e.g. ***Mehlhorn*** 1975 for BST

So: Entropy of the *Entire* Demand

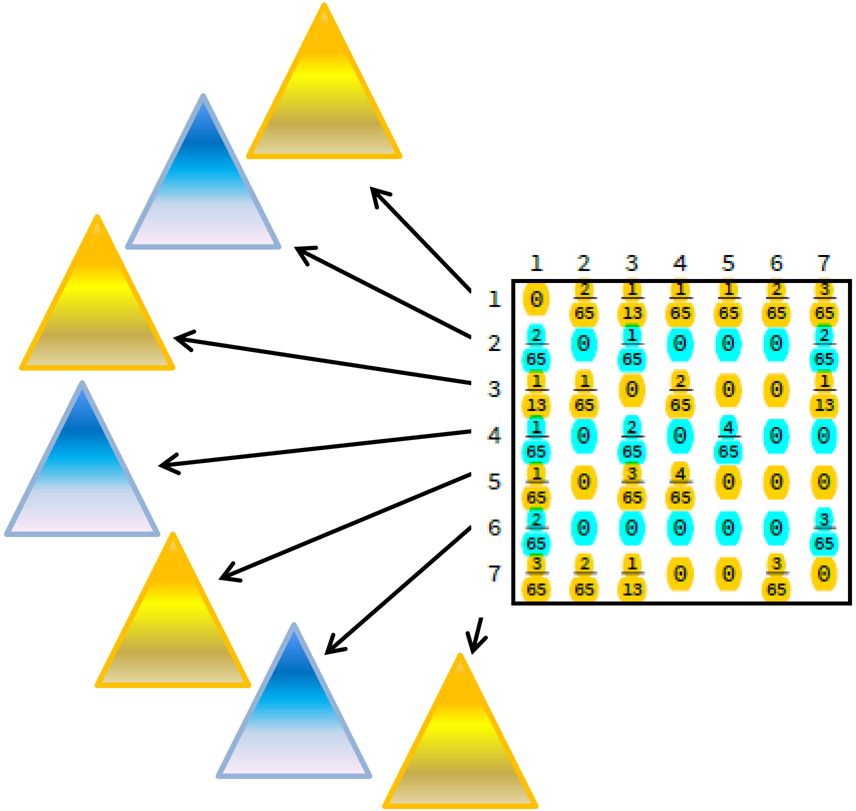
sources

destinations

- **Proof idea** (EPL=Ω(H_Δ(Y|X))):

entropy

- Compute **ego-tree** for each source node
- Take ***union*** of all **ego-trees**
- Violates ***degree restriction*** but valid lower bound



	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

Entropy of the *Entire* Demand: Sources *and* Destinations

Do this in **both dimensions**:

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

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

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

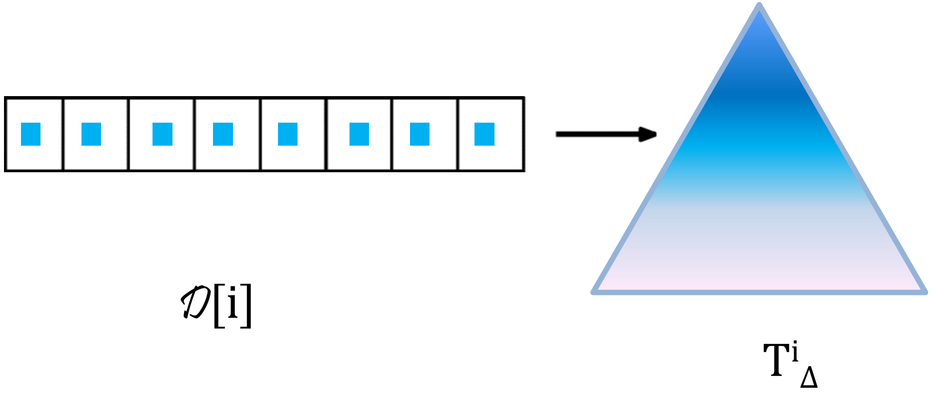
$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13}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

$\mathcal{D}$

Intuition: Reaching Entropy Limit in Datacenters

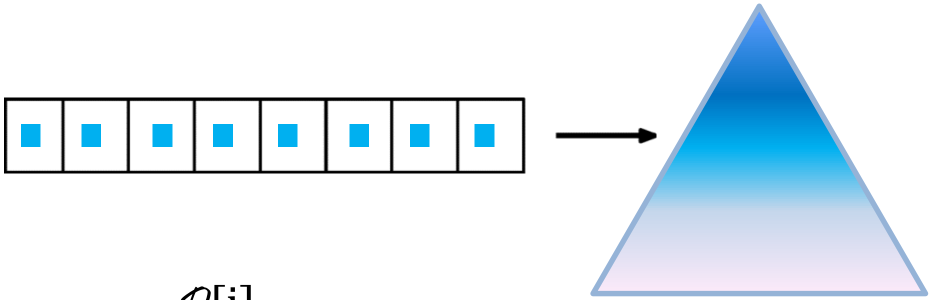
Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



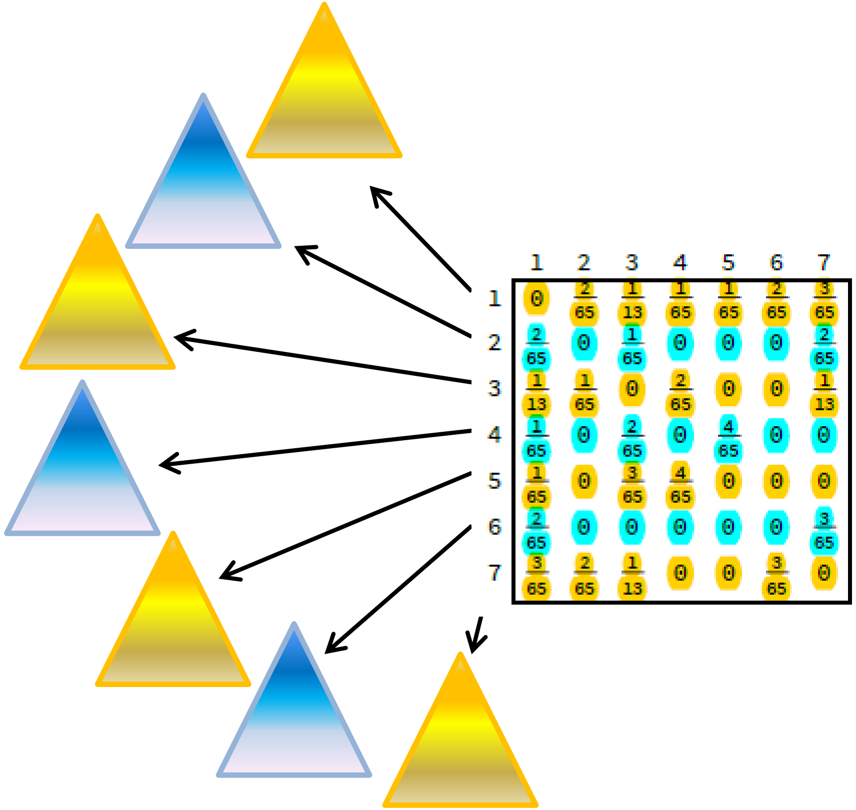
Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



$\mathcal{D}[i]$

T^i_Δ

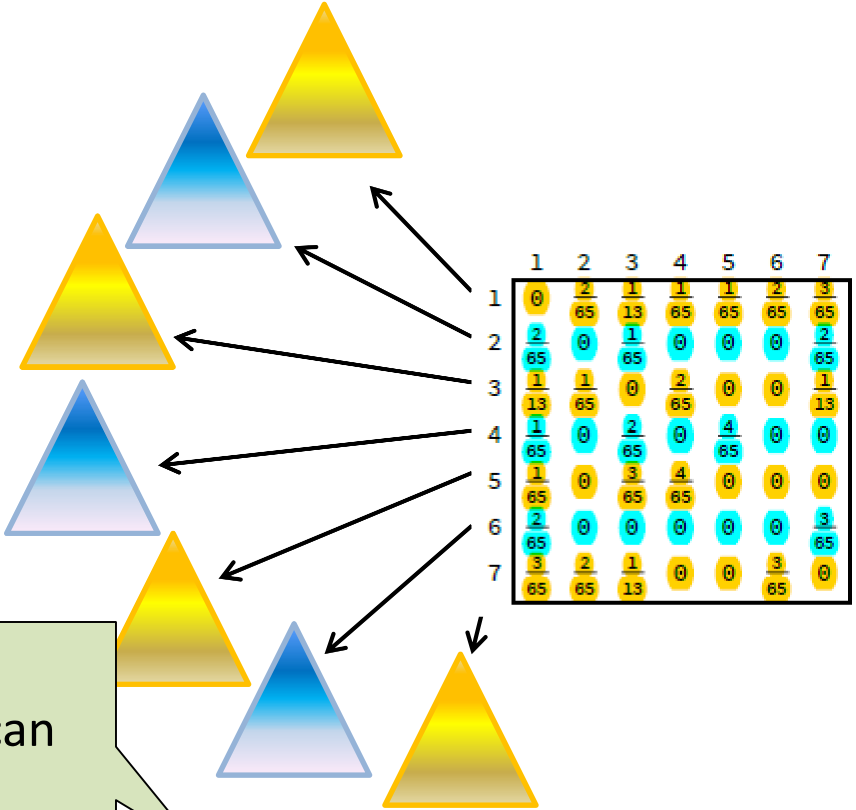


$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13}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

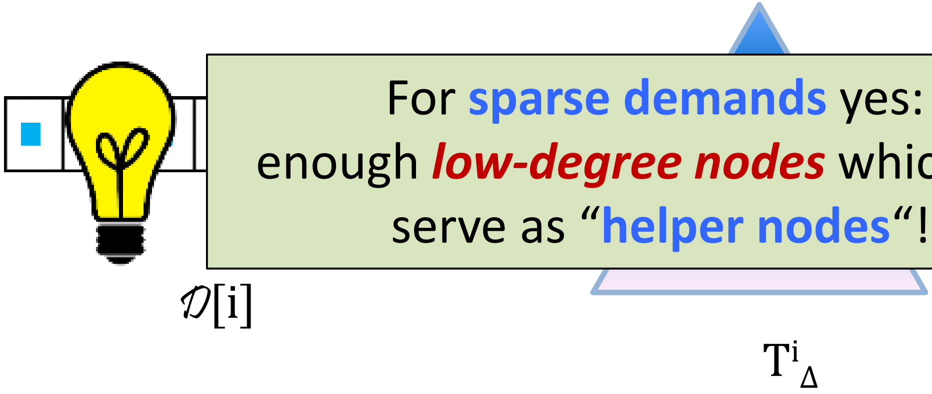
Can we merge the trees ***without distortion*** and ***keep degree low***?

Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



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

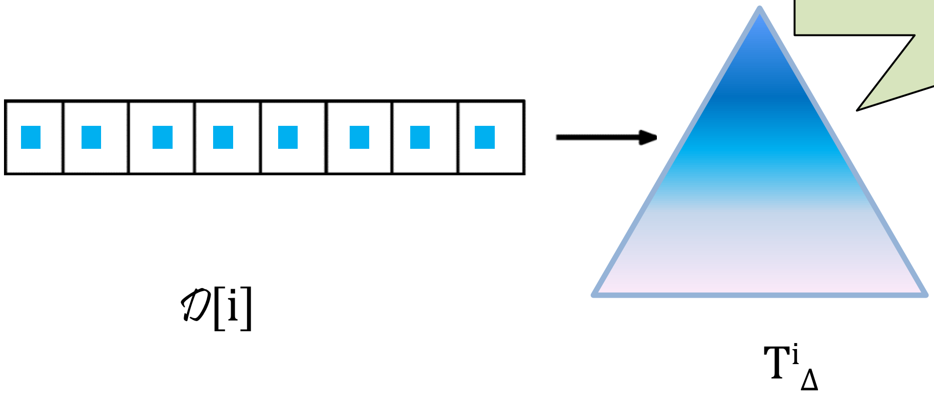


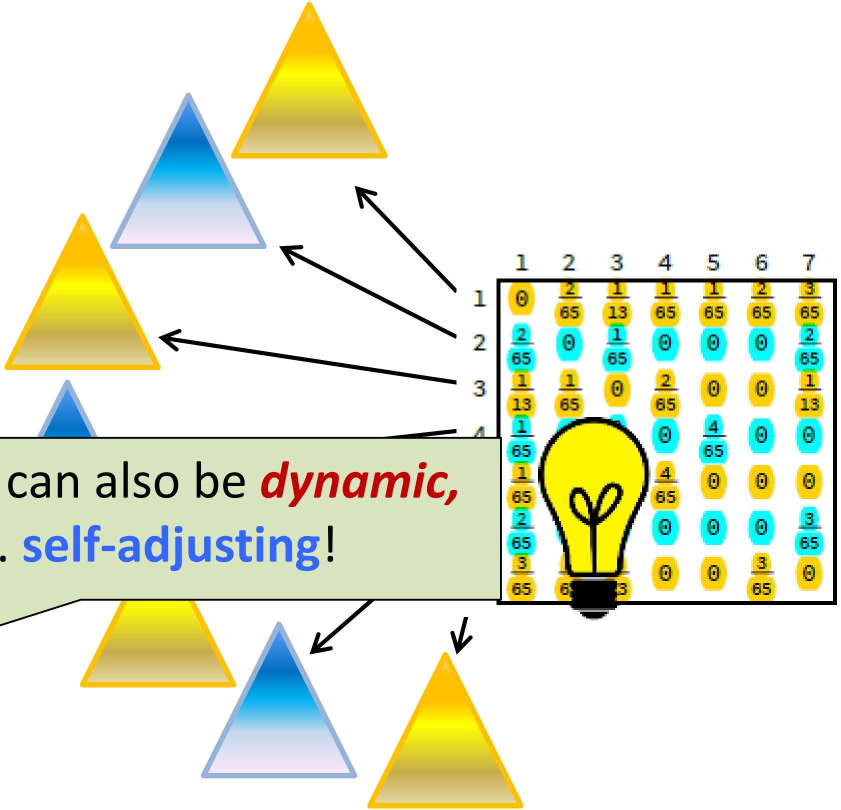
For **sparse demands** yes:
enough ***low-degree nodes*** which can
serve as "**helper nodes**"!

Can we merge the trees ***without distortion*** and ***keep degree low***?

Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)





	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	$\frac{1}{65}$	[illegible]	[illegible]	0	$\frac{4}{65}$	0	0
	$\frac{1}{65}$	[illegible]	[illegible]	$\frac{4}{65}$	0	0	0
	$\frac{2}{65}$	[illegible]	[illegible]	0	0	0	$\frac{3}{65}$
	$\frac{3}{65}$	[illegible]	[illegible]	0	0	$\frac{3}{65}$	0

Ego-tree can also be ***dynamic,*** i.e. **self-adjusting**!

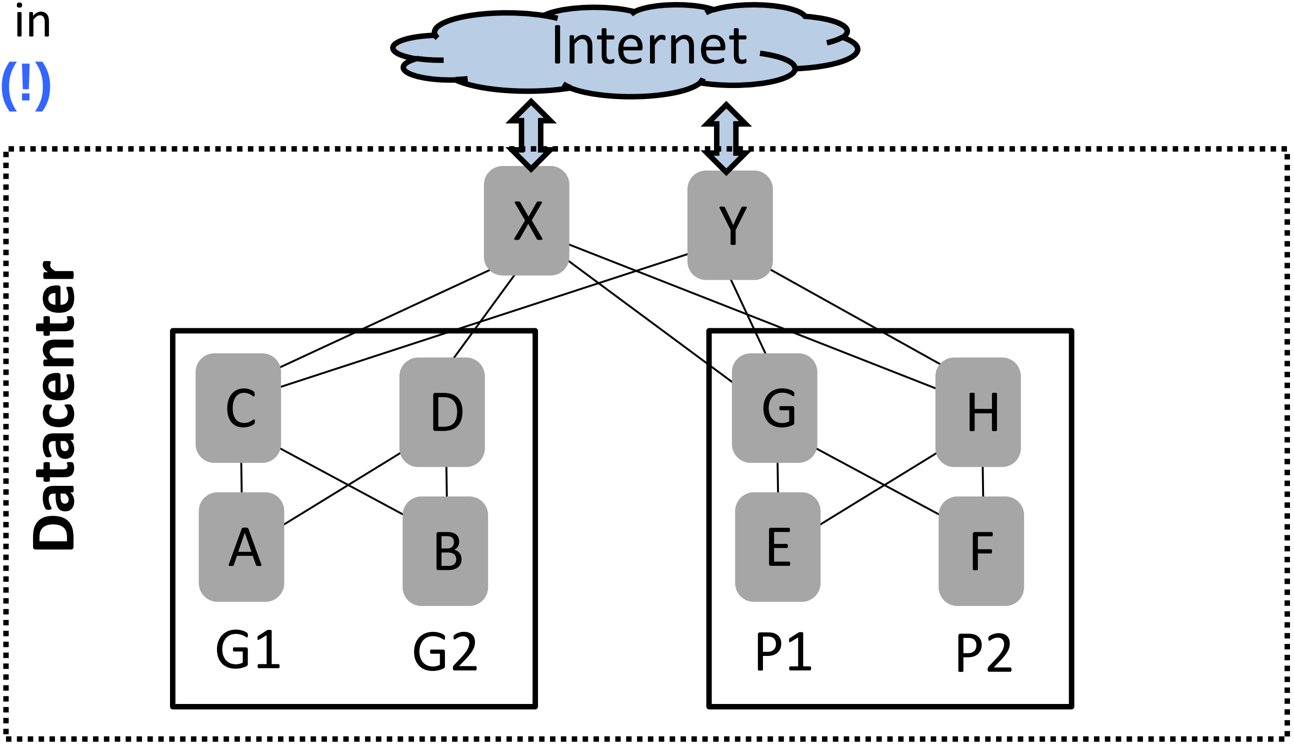
Can we merge the trees ***without distortion*** and ***keep degree low***?

Roadmap

- Opportunities of self-* networks
  - Example 1: Demand-aware, self-adjusting networks ✓
  - **Example 2: Self-repairing networks**
- Challenges of desinging self-* networks

Reasoning About Failures is Hard

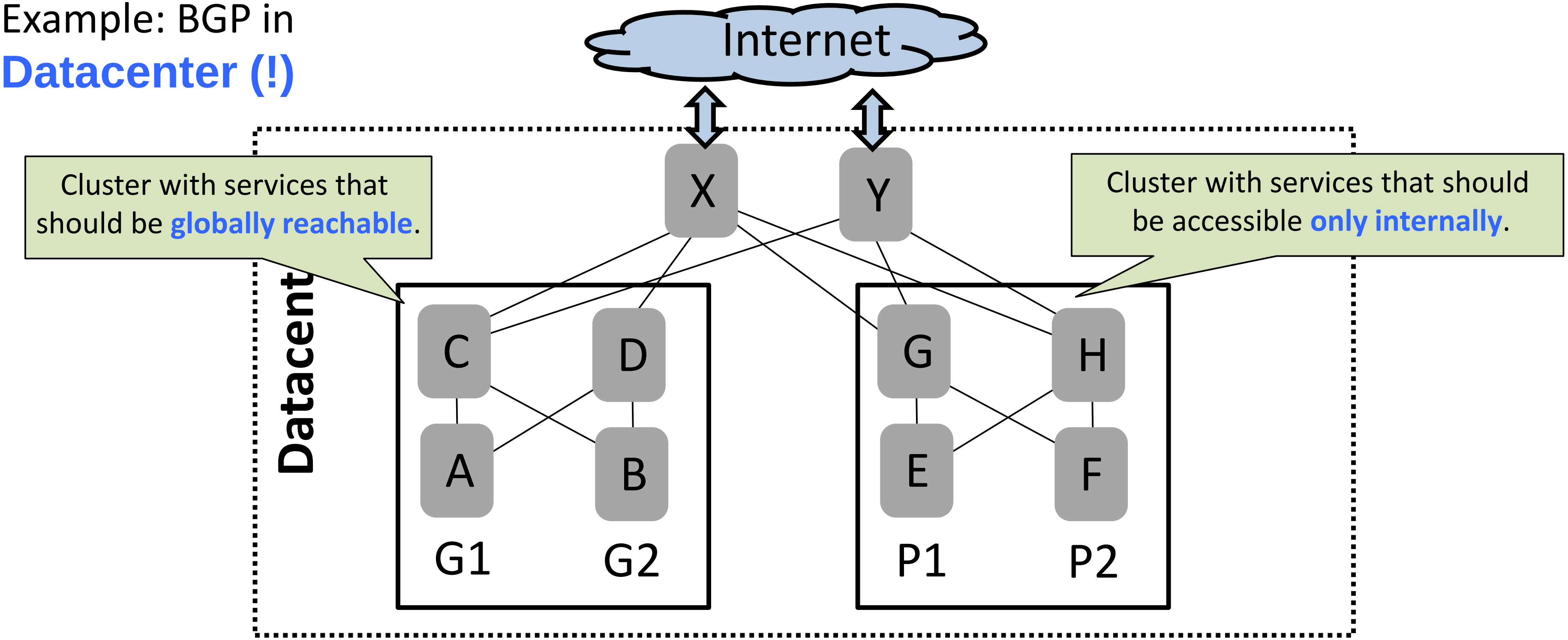
Example: BGP in **Datacenter (!)**



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

Reasoning About Failures is Hard

Example: BGP in
Datacenter (!)



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

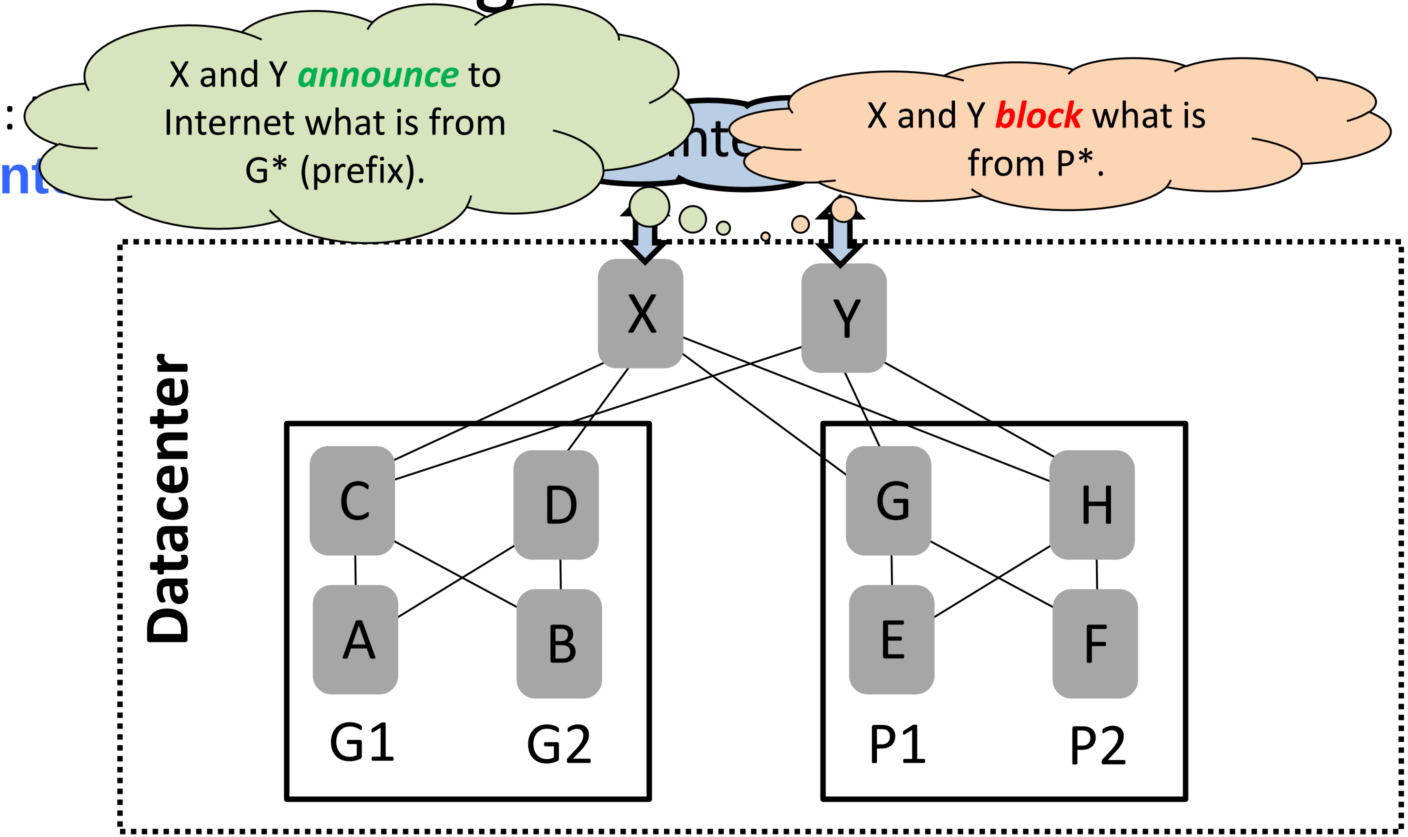
Cluster with services that should be accessible **only internally**.

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

Reasoning About Failures is Hard

Example:

Datacent



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

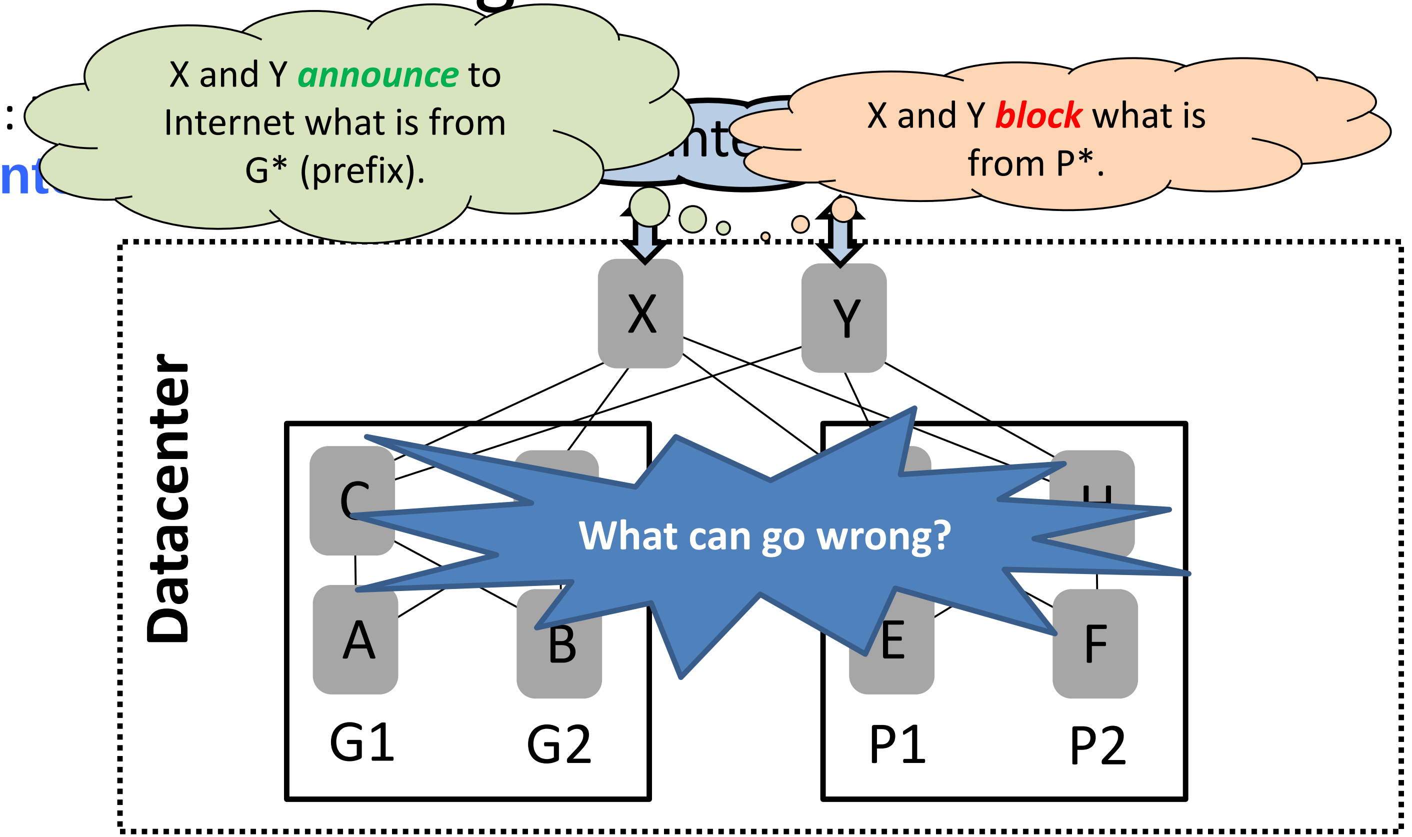
X and Y ***block*** what is from P*.

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

Reasoning About Failures is Hard

Example:

Datacent



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

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

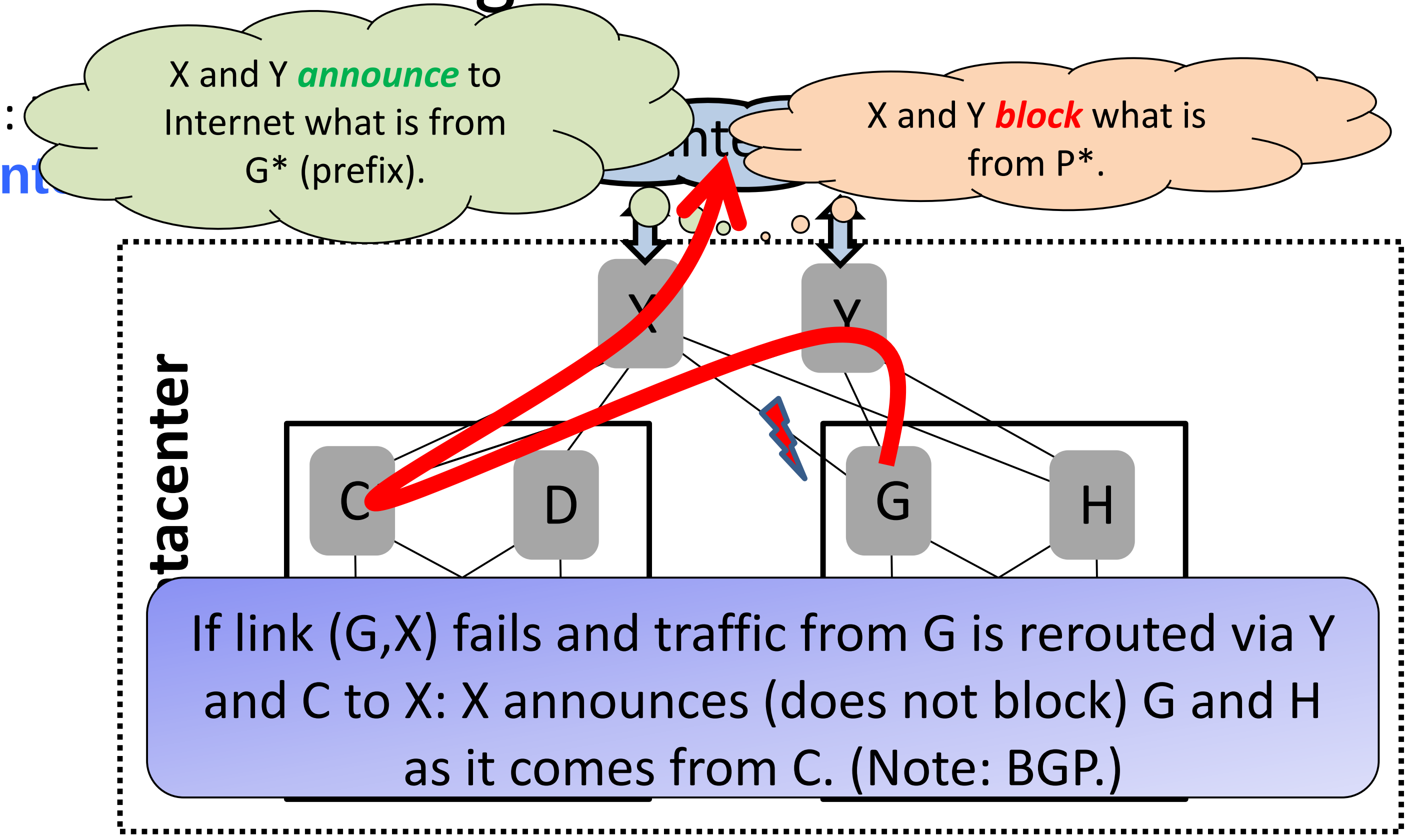
What can go wrong?

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

Reasoning About Failures is Hard

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.)

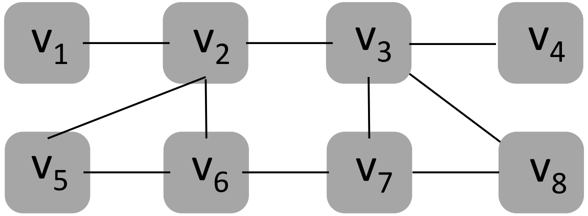
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Managing Complex Networks is Hard for Humans

Another Case for Automation!

Example: Self-Repairing MPLS Networks

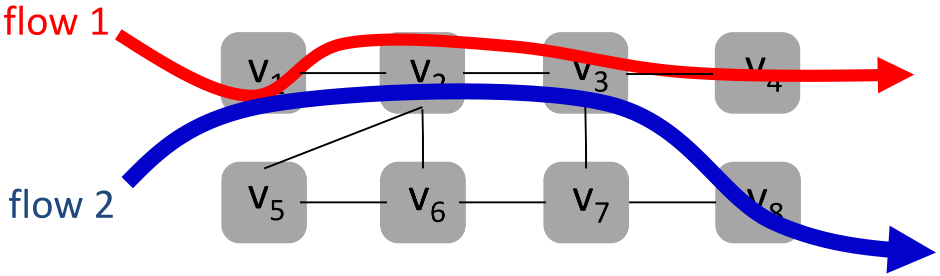
- MPLS: forwarding based on **top label** of label **stack**



Default routing of
two flows

Example: Self-Repairing MPLS Networks

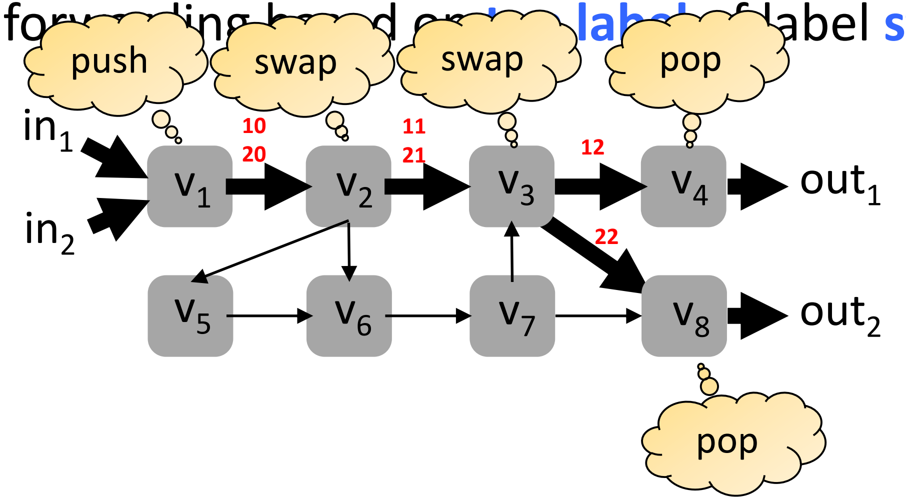
- MPLS: forwarding based on **top label** of label **stack**



Default routing of
two flows

Example: Self-Repairing MPLS Networks

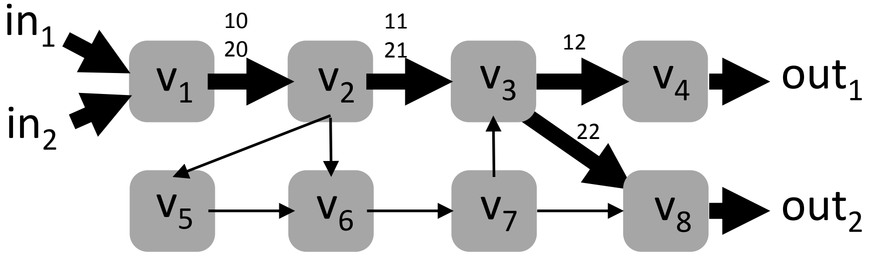
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

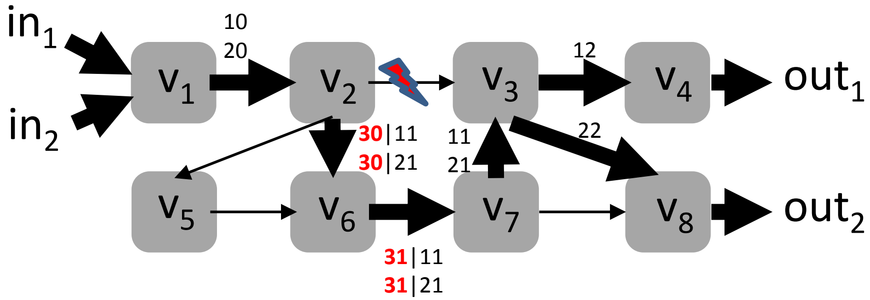
Fast Reroute Around *1 Failure*

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



Default routing of two flows

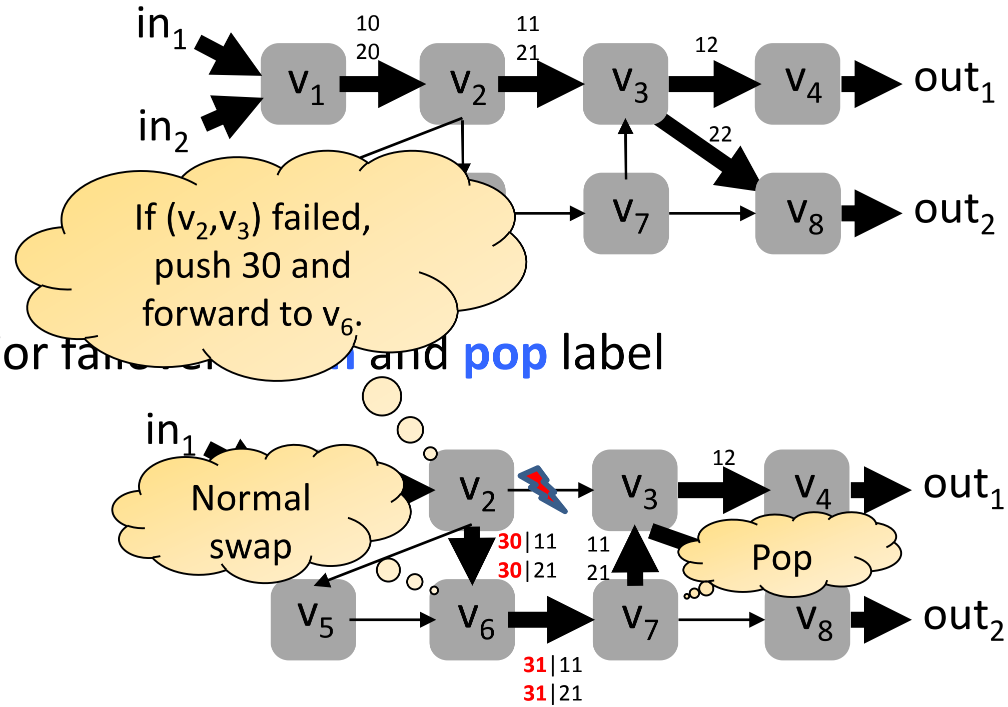
- For failover: **push** and **pop** label



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

Fast Reroute Around *1 Failure*

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



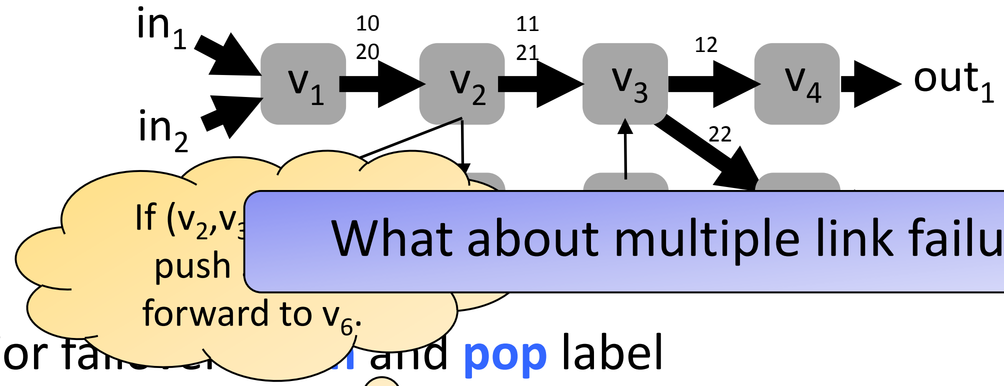
Default routing of
two flows

- For fa and **pop** label

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

Fast Reroute Around *1 Failure*

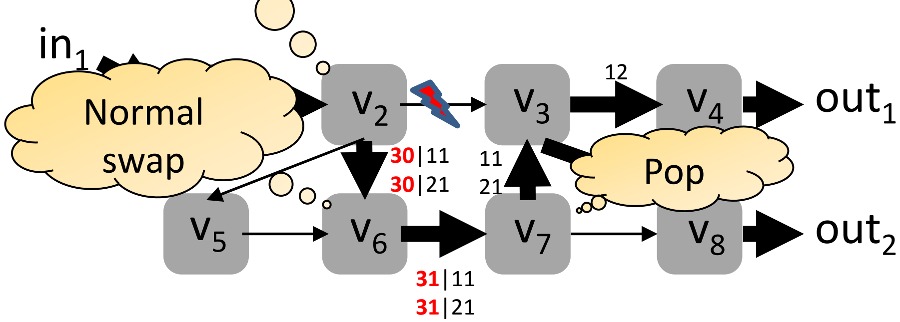
- MPLS: forwarding based on **top label** of label **stack**



Default routing of two flows

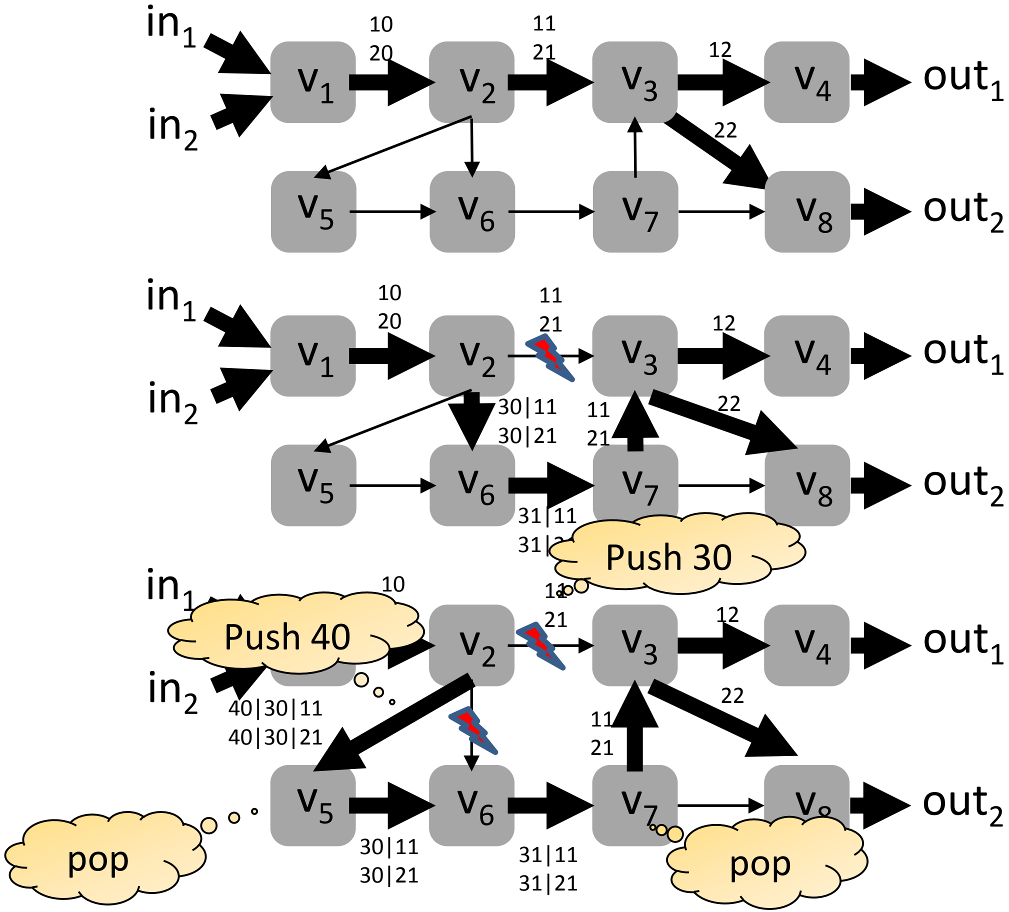
What about multiple link failures?

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



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

2 Failures: Push *Recursively*



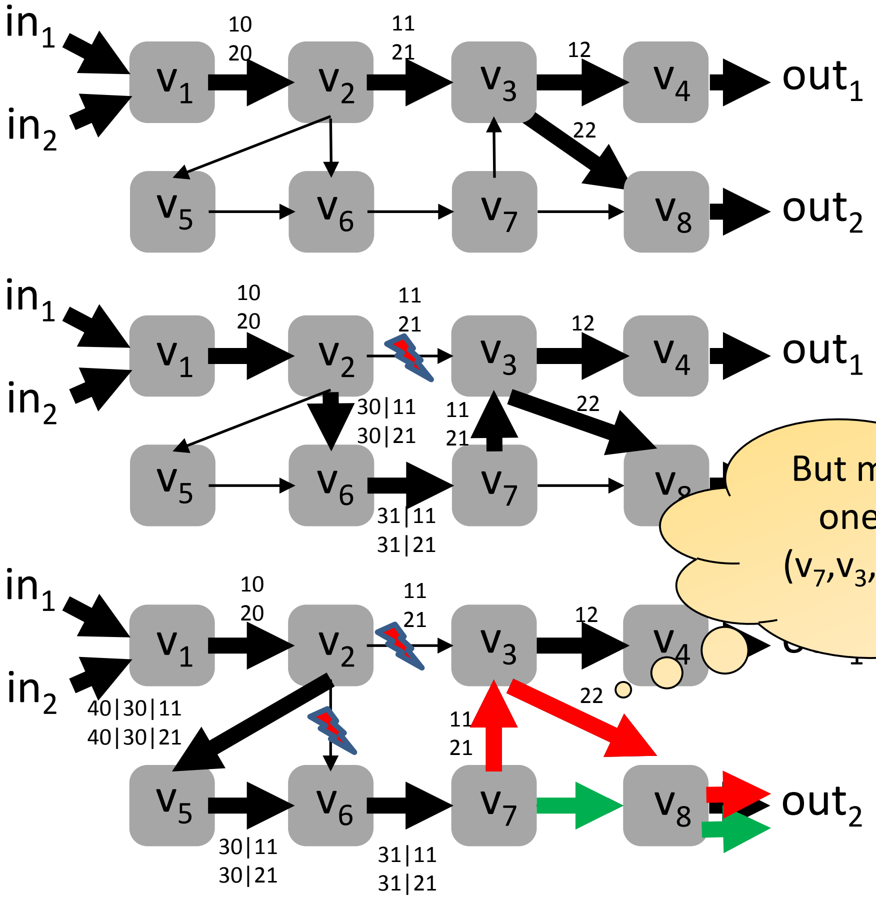
Original Routing

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:

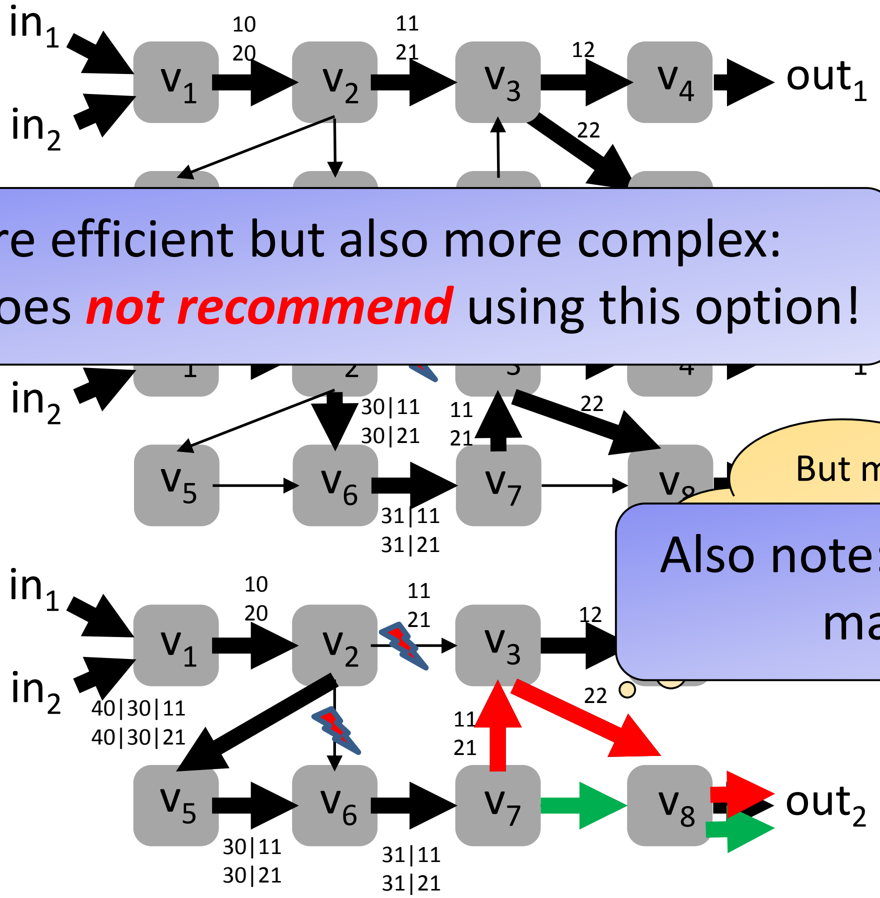
)

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:

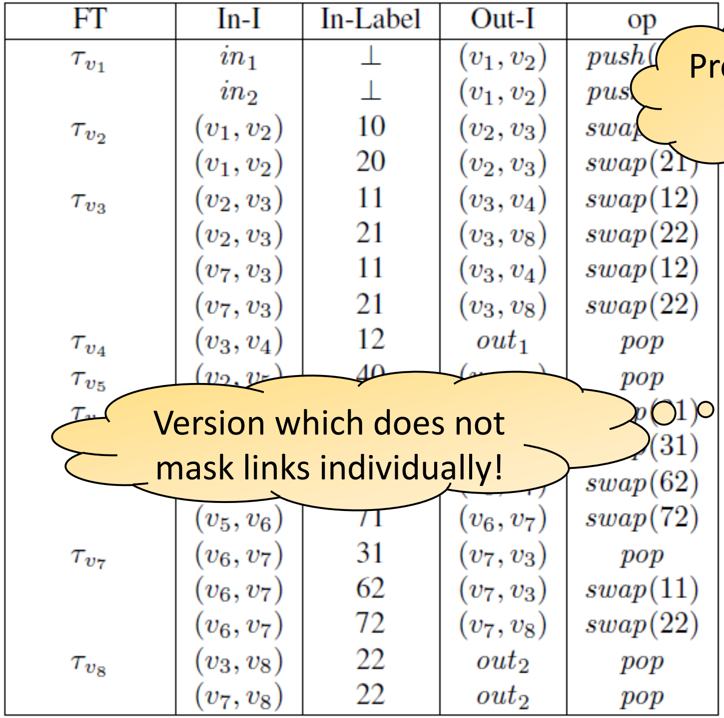
But masking links one-by-

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	$\bot$	(v_1, v_2)	[illegible]
	in_2	$\bot$	(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}	(v_2, v_5)	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

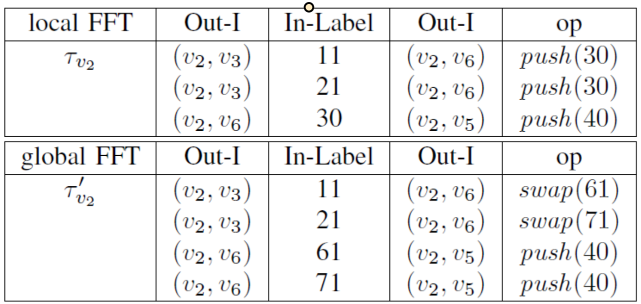
Version which does not mask links individually!

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)	$push(40)$

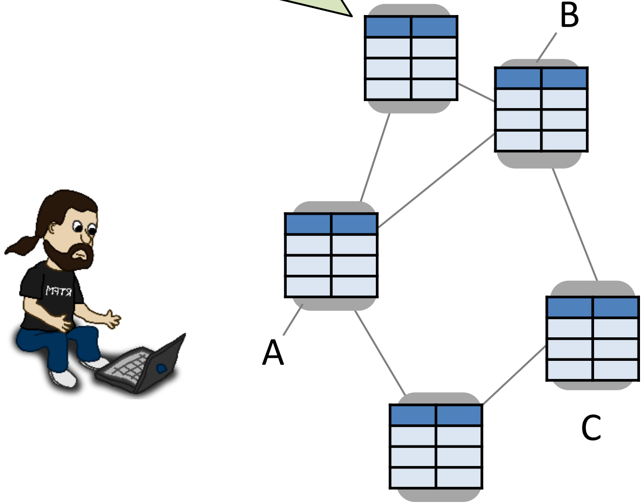
global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

Failover Tables

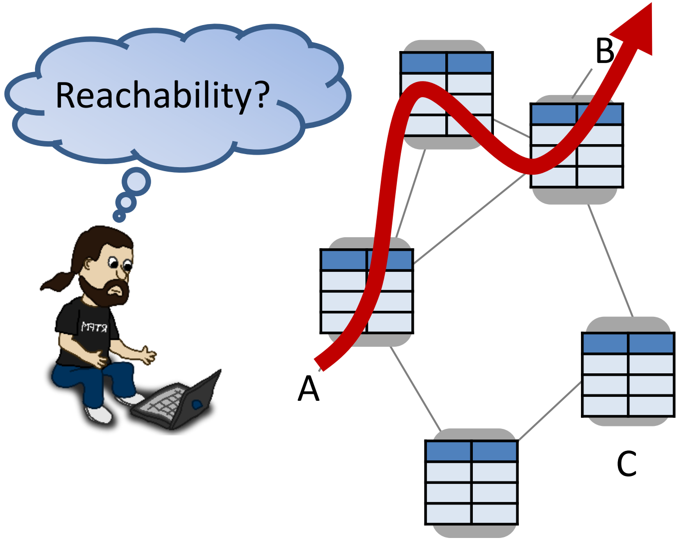
MPLS Tunnels in Today‘s ISP Networks

Responsibilities of a Sysadmin

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



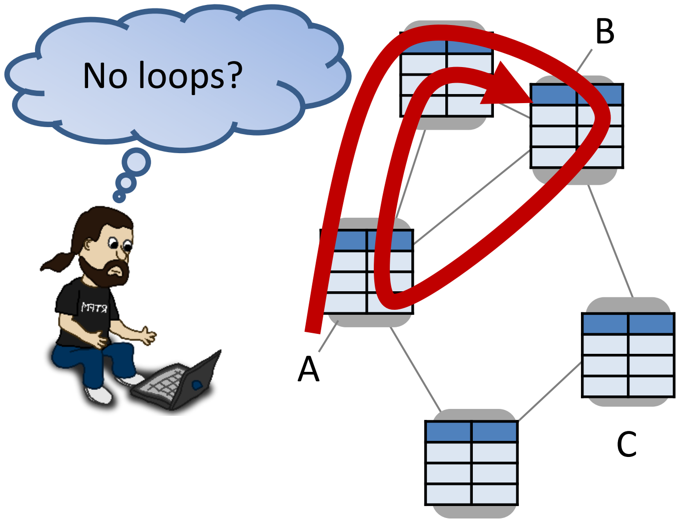
Responsibilities of a Sysadmin



Sysadmin responsible for:

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

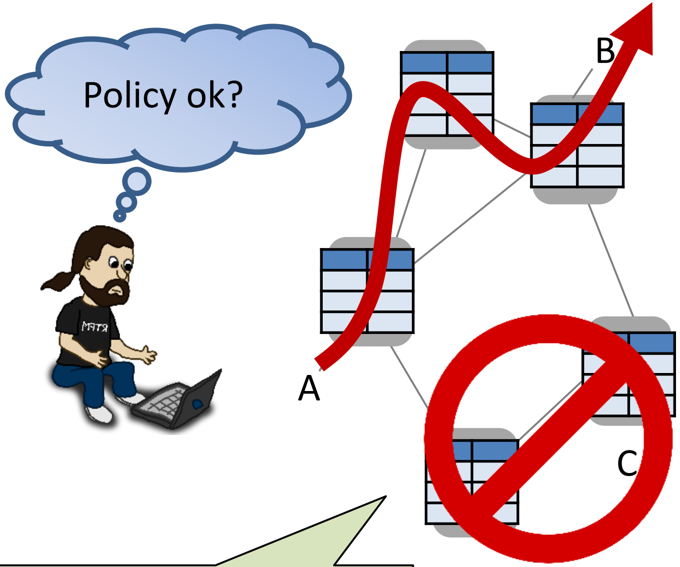
Responsibilities of a Sysadmin



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

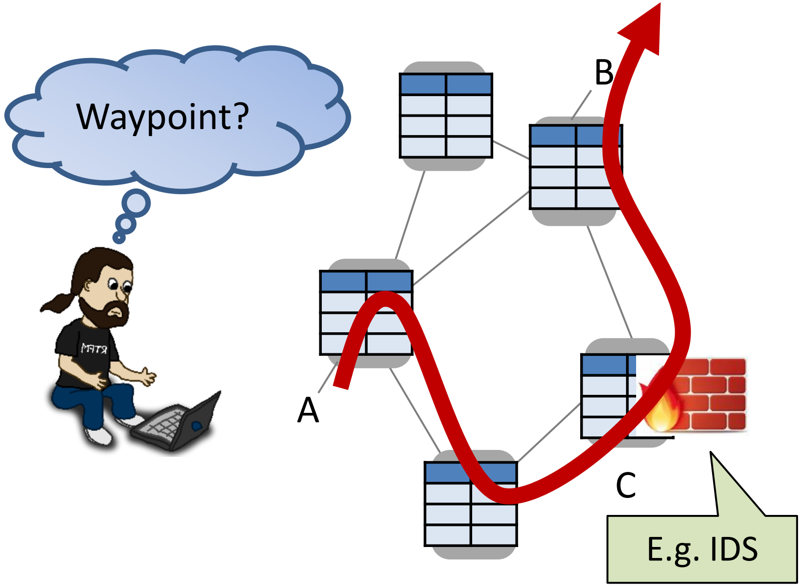


E.g. ***NORDUnet***: no traffic via Iceland (expensive!).

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?

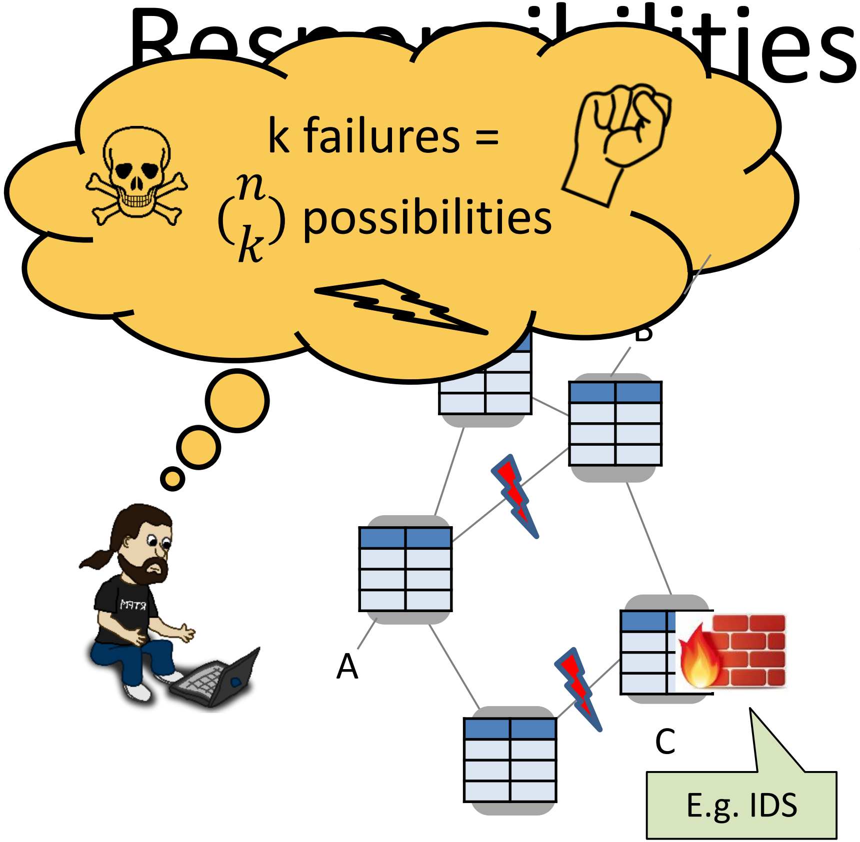
Responsibilities of a Sysadmin



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?!

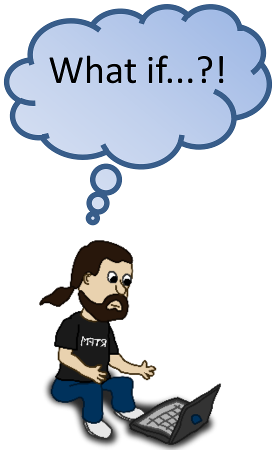
Can we automate such tests or even self-repair?

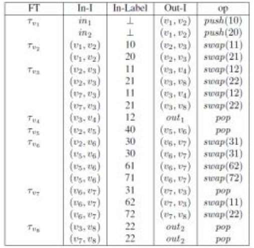
Can we automate such tests
or even self-repair?

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

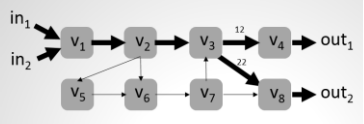
(INFOCOM 2018, CoNEXT 2018, IFIP Networking 2019)

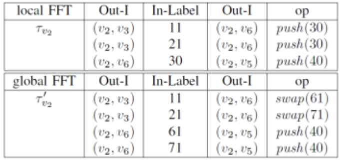
Leveraging Automata-Theoretic Approach





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.

Compilation

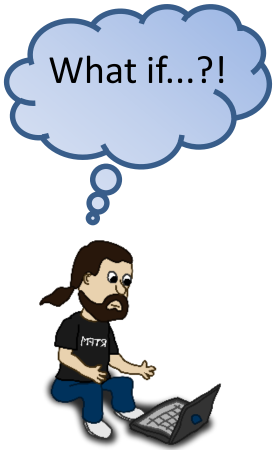
Interpretation

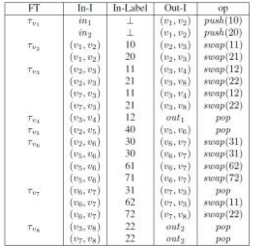
$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

Pushdown Automaton
and **Prefix Rewriting**
Systems Theory

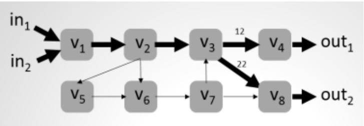
Leveraging Autom oach

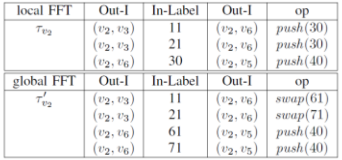
Use cases: Sysadmin ***issues queries*** to test certain properties, or do it on a ***regular basis*** automatically!



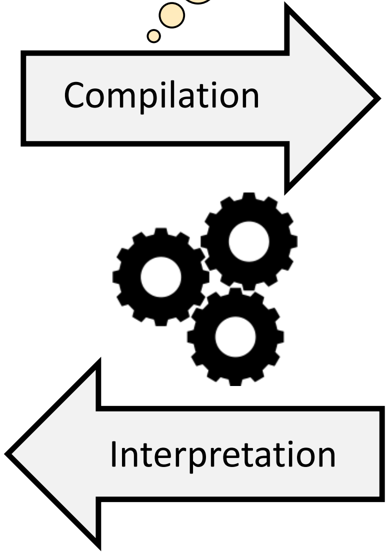


FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$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



Compilation

Interpretation

$$pX \Longrightarrow qXX$$

$$pX \Longrightarrow qYX$$

$$qY \Longrightarrow rYY$$

$$rY \Longrightarrow r$$

$$rX \Longrightarrow pX$$

MPLS **configurations**, Segment Routing etc.

Pushdown Automaton and **Prefix Rewriting Systems** Theory

Mini-Tutorial: A Network Model

- Network: a 7-tuple

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

Nodes

Links

Incoming interfaces

Outgoing 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

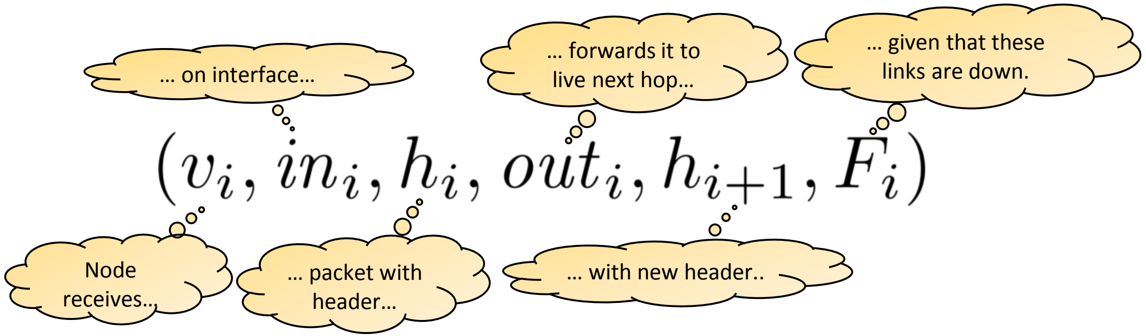
Routing function: for each set of failed links $F \subseteq E$, the routing function

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

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:



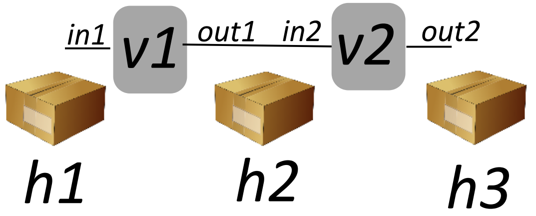
$$(v_i, in_i, h_i, out_i, h_{i+1}, F_i)$$

- 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:

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

Failover-Swap:

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

Failover-Pop:

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

Example rewriting sequence:

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

Try default

Try first backup

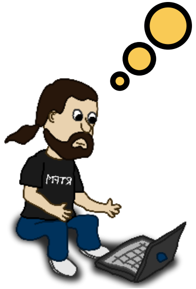
Try second backup

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?!

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



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

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

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

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

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

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

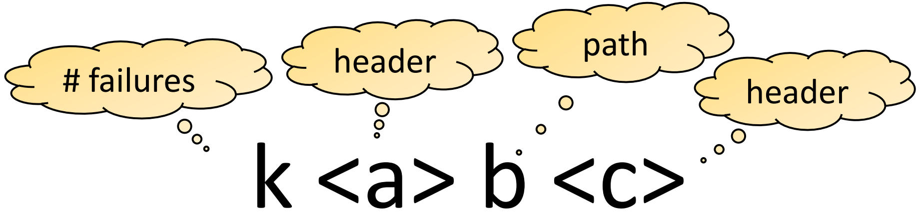
Tool and Query Language

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

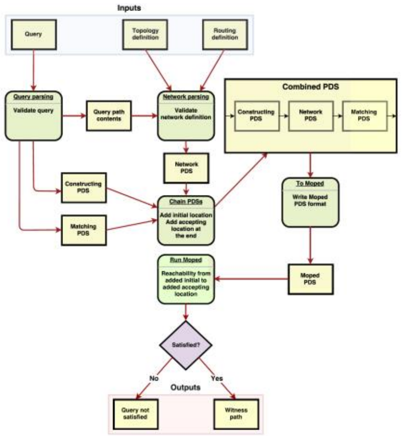
- In Python 3

Part 2: Reachability analysis of constructed PDS

- Using ***Moped*** tool



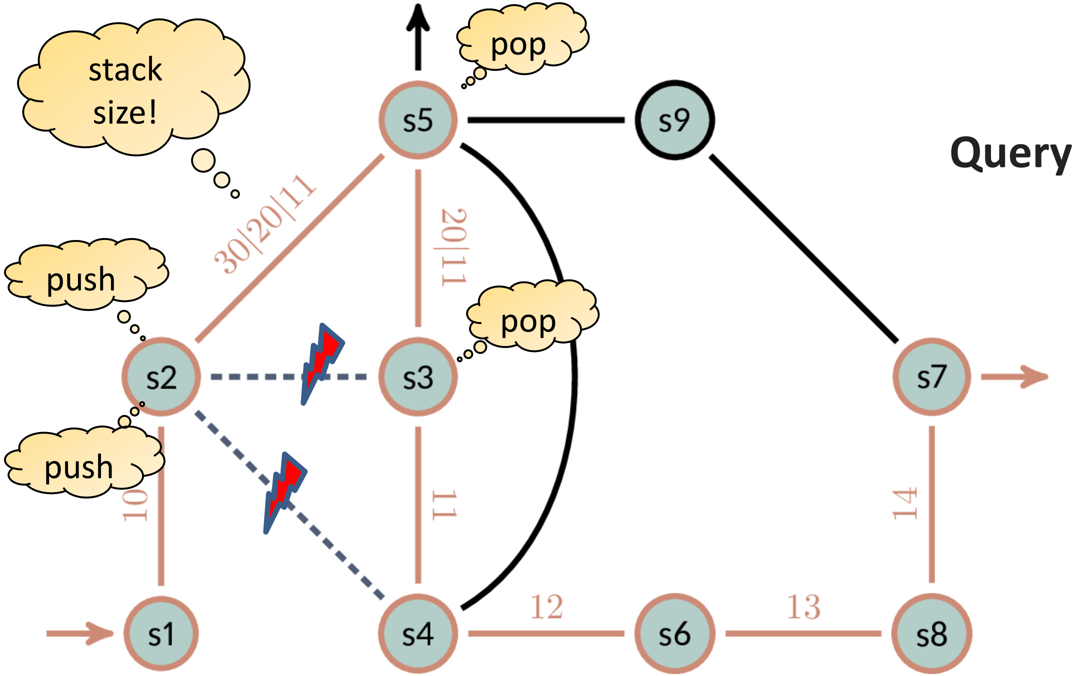
Regular query language



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?



2 failures

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

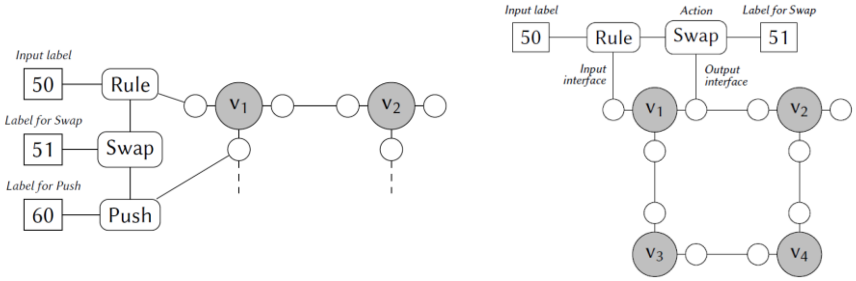
YES!

(Gives witness!)

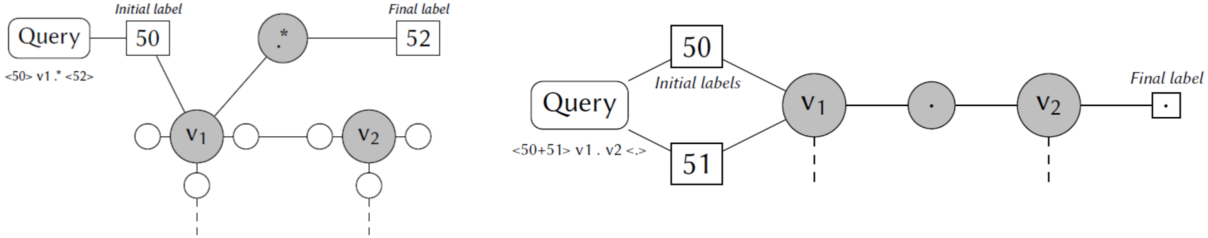
Formal methods are nice (give guarantees!)… But what about ML…?!

Speed Up Further and Synthesize: Deep Learning

- Yes sometimes **without losing guarantees**
- Extend **graph-based neural networks**
- **Predict** counter-examples and **fixes**



Network topologies and MPLS rules



Network topologies and query

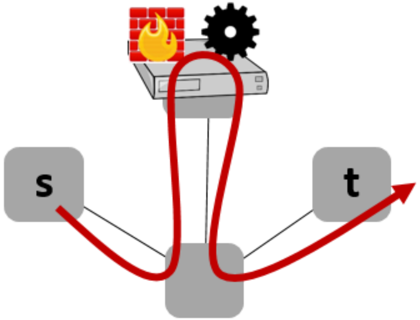
Roadmap

- Opportunities of self-* networks
  - Example 1: Demand-aware, self-adjusting networks ✓
  - Example 2: Self-repairing networks ✓

- Challenges of desinging self-* networks

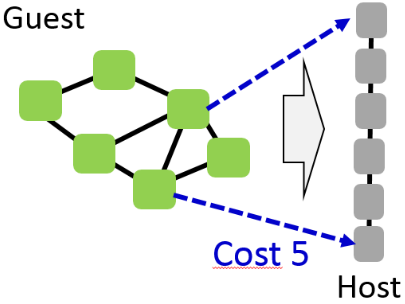
Challenge 1: Hard Problems

- Optimization problems are often **NP-hard**: hard *even for computers*!



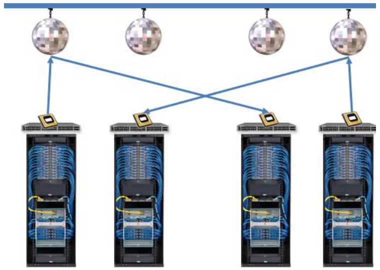
Waypoint routing:

disjoint paths



Embedding:

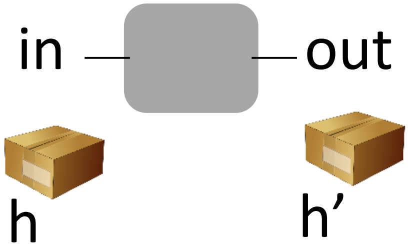
Minimum Lin. Arrangment



Topology design:

Graph spanners

It can get worse...: intractable!



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

(Simplified) MPLS rules:

prefix rewriting

in x L → *out x* ***OP***

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

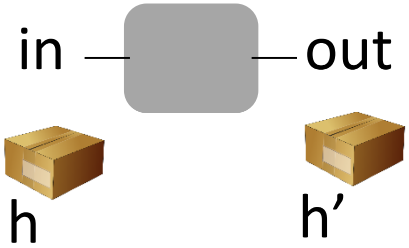
vs

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

arbitrary header rewriting

*in x L** → *out x L**

It can get worse…: intractable!



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

(Simplified) MPLS rules:

prefix rew

t x ***OP***

whe OP = {swap,push,pop}

Polynomial time

VS

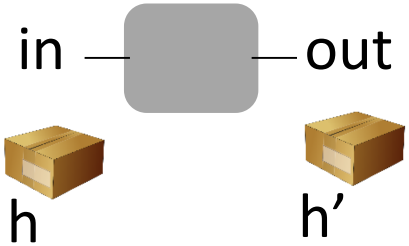
Rules of general network g., SDN):

arbitrary h ting

i → *out x L**

Undecidable!

It can get worse…: intractable!



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

What is a good tradeoff between generality and performance?

(Simpl

pr

Polynom

ut x O

whe OP = {swap,push,pop}

*i → out x L**

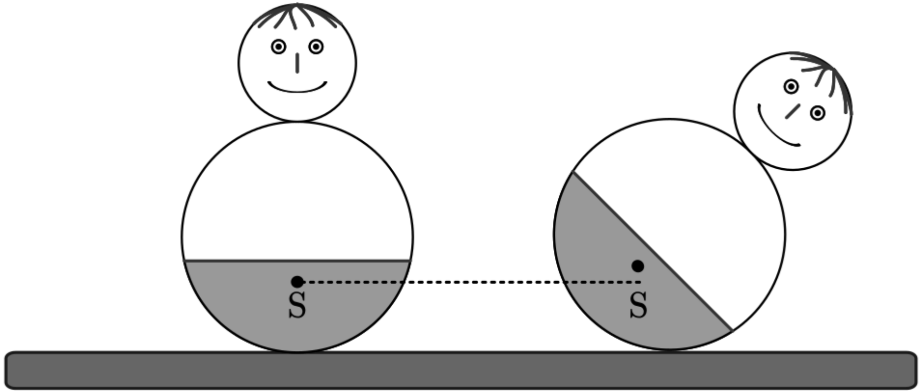
Challenge 2: Realizing Limits?

- Can a self-* network realize its **limits**?
- E.g., when quality of **input data** is not good enough?
- When to hand over to human? Or **fall back** to „safe/oblivious mode“?
- Can we learn from self-driving **cars**?

Challenge 3: Self-Stabilization

- Could be an attractive property of self-* network!

A **self-stabilizing** system guarantees that it *reconverges to a desirable configuration* or state, *from any initial state*.



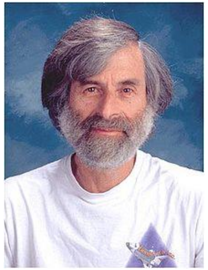
„Stehaufmännchen“

Self-Stabilization

Self-stabilizing algorithms pioneered by **Dijkstra** (1973): for example self-stabilizing mutual exclusion.

“I regard this as Dijkstra’s most brilliant work. Self-stabilization is a very important concept in fault tolerance.”

Leslie **Lamport** (PODC 1983)

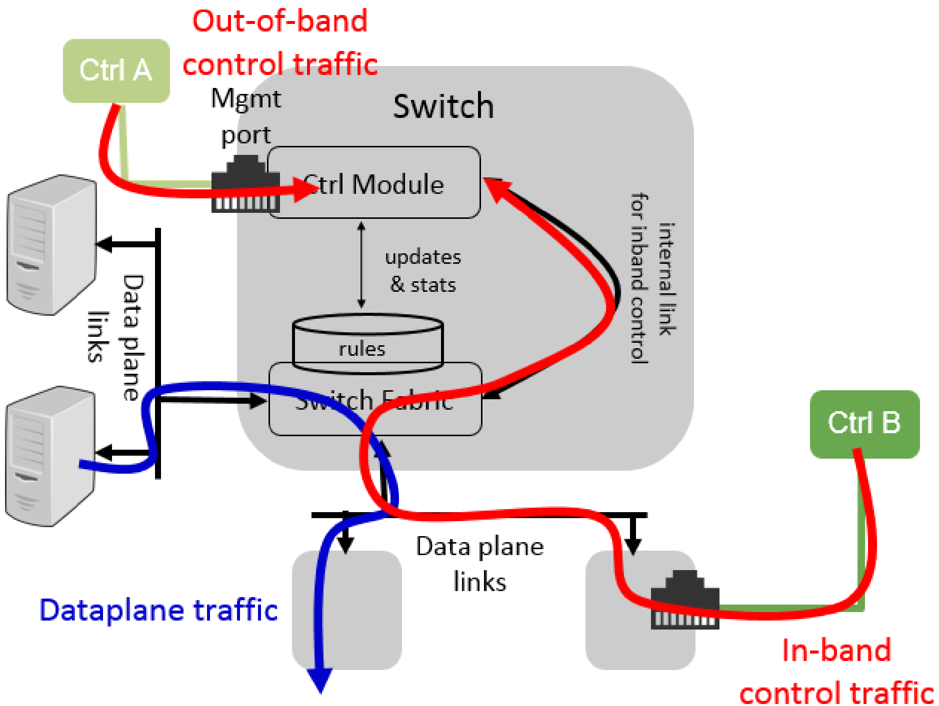


Some notable works by **Perlman** toward self-stabilizing Internet, e.g., self-stabilizing spanning trees.

Yet, many protocols in the Internet are *not* self-stabilizing. Much need for future work.

E.g., Self-Stabilizing SDN Control?

- Distributed SDN control plane which **self-organizes management** of switches?
- Especially challenging: **inband control** (how to distinguish traffic?)



Challenge 4: Uncertainties

- How to deal with **uncertainties**?
- How to maintain flexibilities?
- Use of principles from robotics? E.g., **empowerment**?

Conclusion

- **Flexibilities** in networks: great opportunities for **optimization** and **automation**

- **Demand-aware** and **self-adjusting** networks: beating the routing lower bounds of oblivious networks, ***reaching entropy bounds***

- Potential of **self-repairing** networks, self-stabilizing networks, etc.

- Much work ahead: ***tradeoff*** generality vs efficiency? How to self-monitor and **fall-back** if needed? Use of **formal methods** and ML?

Further Reading

Flexibilities and Complexity

On The Impact of the Network Hypervisor on Virtual Network Performance
Andreas Blenk, Arsany Basta, Wolfgang Kellerer, and Stefan Schmid.
IFIP Networking, Warsaw, Poland, May 2019.
Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges (Invited Paper)
Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid.
Proceedings of the IEEE (**PIEEE**), 2019.
Efficient Distributed Workload (Re-)Embedding
Monika Henzinger, Stefan Neumann, and Stefan Schmid.
ACM/IFIP **SIGMETRICS/PERFORMANCE**, Phoenix, Arizona, USA, June 201
Parametrized Complexity of Virtual Network Embeddings: Dynamic & Linear Programming Approximations
Matthias Rost, Elias Döhne, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), January 2019.
Charting the Complexity Landscape of Virtual Network Embeddings (Best Paper Award)
Matthias Rost and Stefan Schmid.
IFIP Networking, Zurich, Switzerland, May 2018.
Tomographic Node Placement Strategies and the Impact of the Routing Model
Yvonne Anne Pignolet, Stefan Schmid, and Gilles Tredan.
ACM **SIGMETRICS**, Irvine, California, USA, June 2018. hmid.
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.

Further Reading

Demand-Aware and Self-Adjusting Networks

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks (Editorial)
Chen Avin and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), October 2018.
Demand-Aware Network Design with Minimal Congestion and Route Lengths
Chen Avin, Kaushik Mondal, and Stefan Schmid.
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.
Documents: paper pdf, bibtex bib
Distributed Self-Adjusting Tree Networks
Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid.
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.
Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks
Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu.
IFIP Networking, Warsaw, Poland, May 2019.
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.
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.

Further Reading

Self-Repairing Networks

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.
Renaissance: A Self-Stabilizing Distributed SDN Control Plane
Marco Canini, Iosif Salem, Liron Schiff, Elad Michael Schiller, and Stefan Schmid.
38th IEEE International Conference on Distributed Computing Systems (**ICDCS**), Vienna, Austria, July 2018.
Empowering Self-Driving Networks
Patrick Kalmbach, Johannes Zerwas, Peter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid.
ACM SIGCOMM 2018 Workshop on Self-Driving Networks (**SDN**), Budapest, Hungary, August 2018.
DeepMPLS: Fast Analysis of MPLS Configurations using Deep Learning
Fabien Geyer and Stefan Schmid.
IFIP Networking, Warsaw, Poland, May 2019.