Self-* networks: when flexibility meets algorithms

Stefan Schmid (University of Vienna)

LKN Retreat

The Trend: Flexibilities

Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany…

Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany…

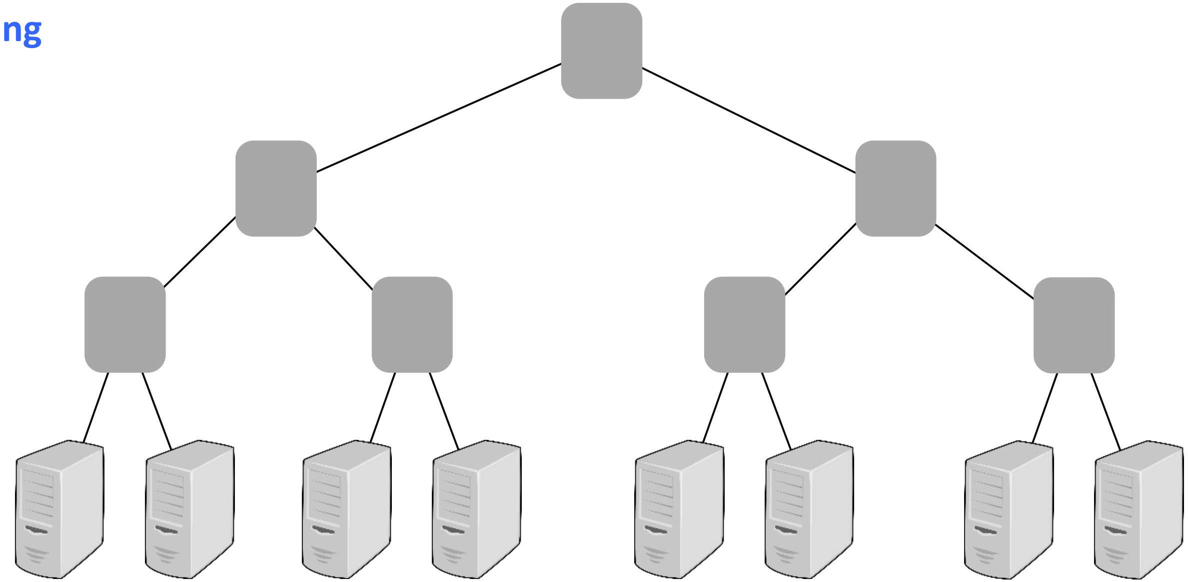
Flexibilities: Along 3 Dimensions

…ful Germany…

Another Trend: Improved Visibility of the Networks

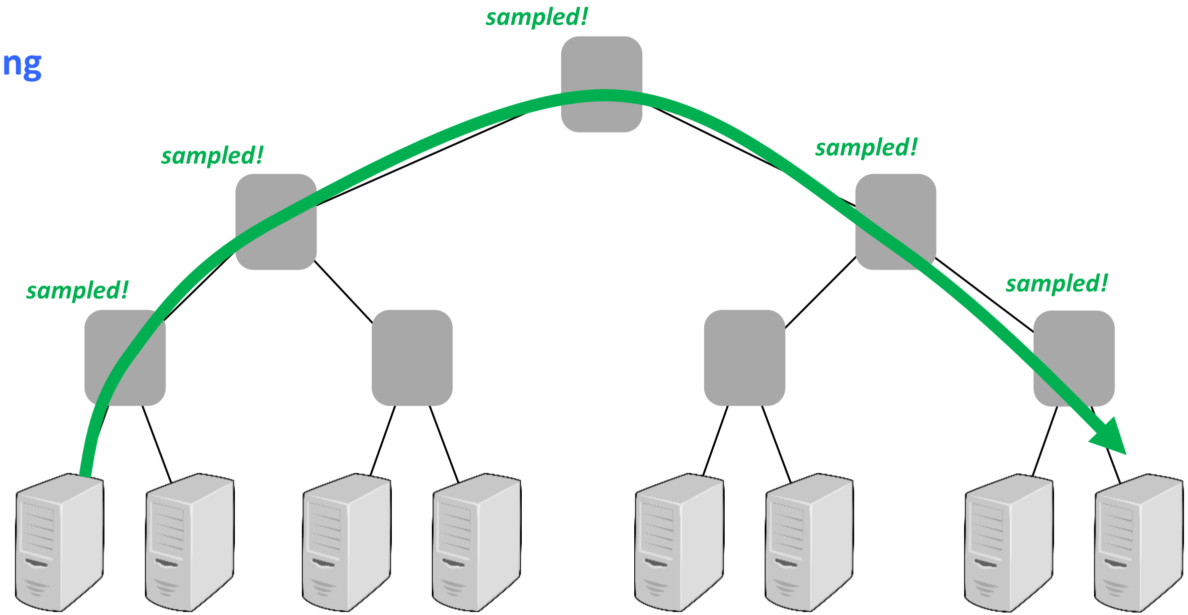
Visibility: SDN, Telemetry, Sketching

- Can also improve ***security***
- Traditionally: e.g., **trajectory sampling**
  - Sample packets with ***hash(imm. header) ∈[x,y]***
  - See routes of ***some* packets**



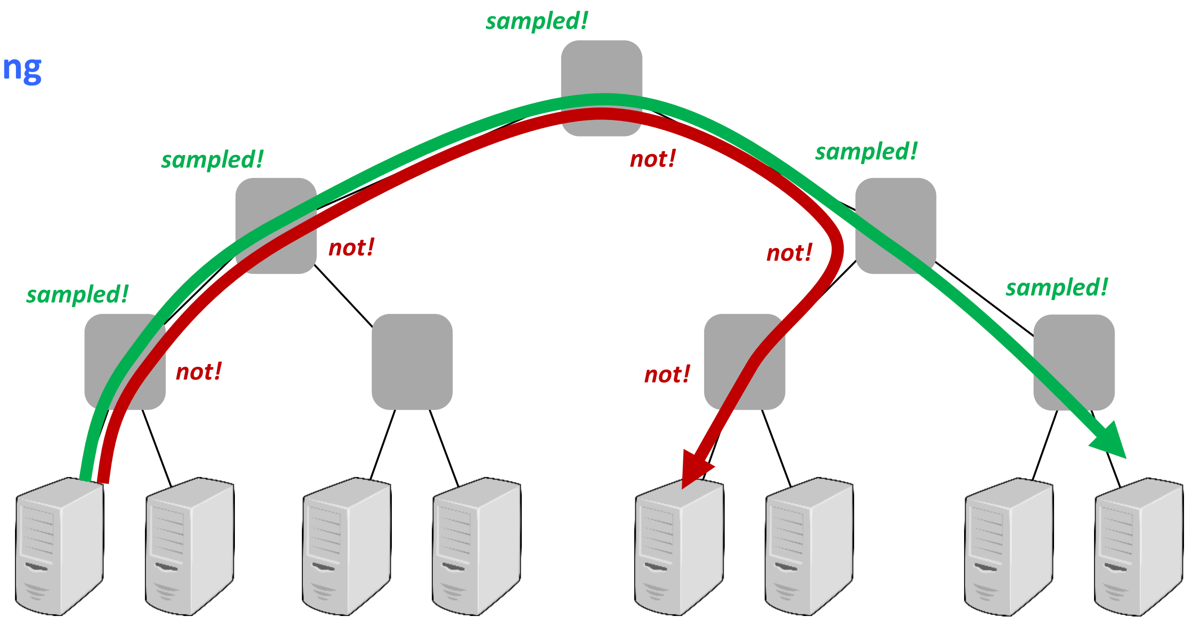
Visibility: SDN, Telemetry, Sketching

- Can also improve ***security***
- Traditionally: e.g., **trajectory sampling**
  - Sample packets with ***hash(imm. header) ∈[x,y]***
  - See routes of ***some* packets**



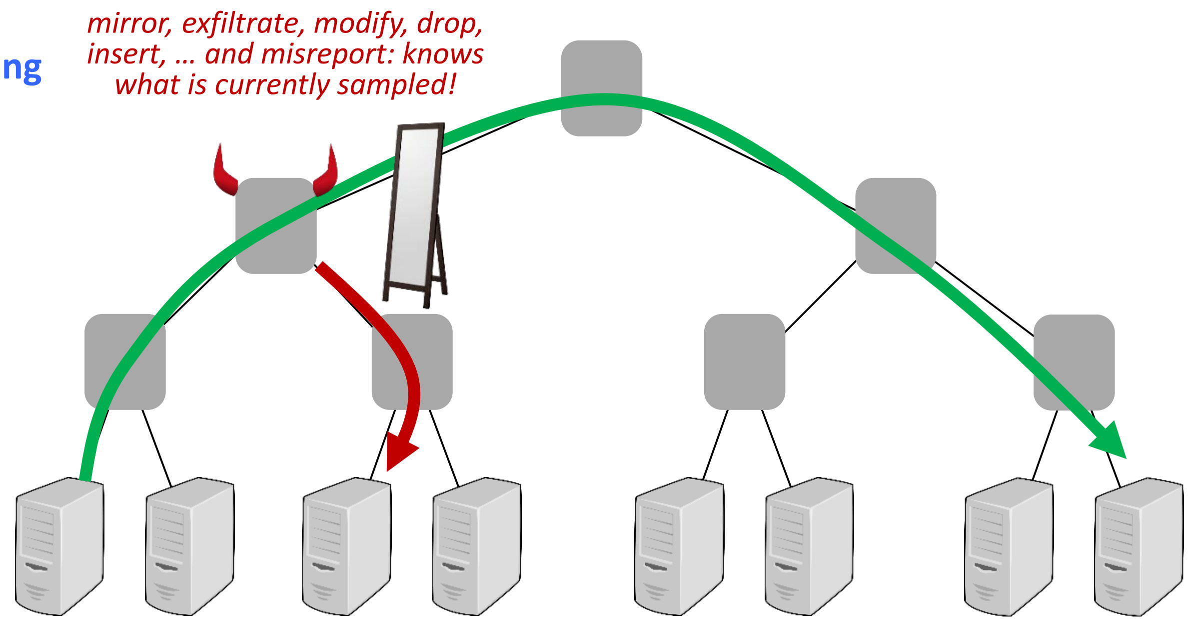
Visibility: SDN, Telemetry, Sketching

- Can also improve ***security***
- Traditionally: e.g., **trajectory sampling**
  - Sample packets with ***hash(imm. header) ∈[x,y]***
  - See routes of ***some* packets**
  - ***Others not!*** *(Usually later…)*



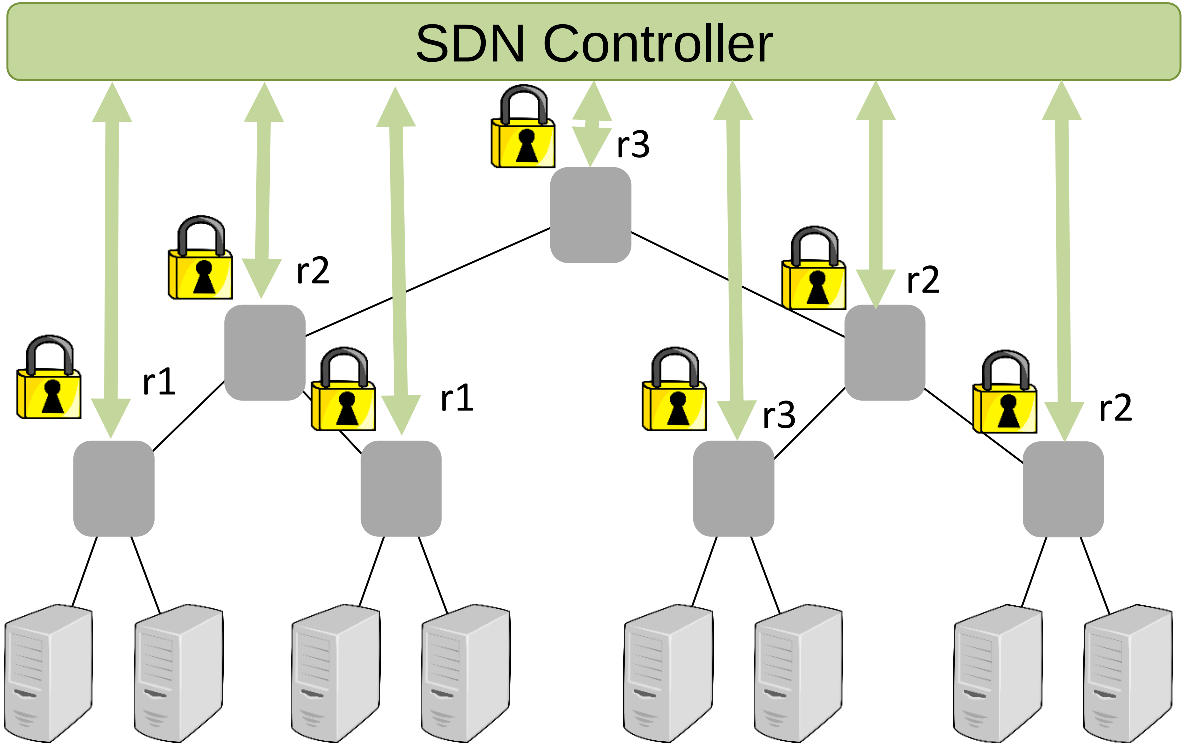
Visibility: SDN, Telemetry, Sketching

- Can also improve ***security***
- Traditionally: e.g., **trajectory sampling**
  - Sample packets with ***hash(imm. header) ∈ [x,y]***
  - See routes of ***some* packets**
  - ***Others not!*** *(Usually later…)*
- BSI question: What can we do if switches may be ***malicious***?
  - Problem: all switches sample the ***same space***: known!
  - Can exploit, e.g., ***know when unobserved***.



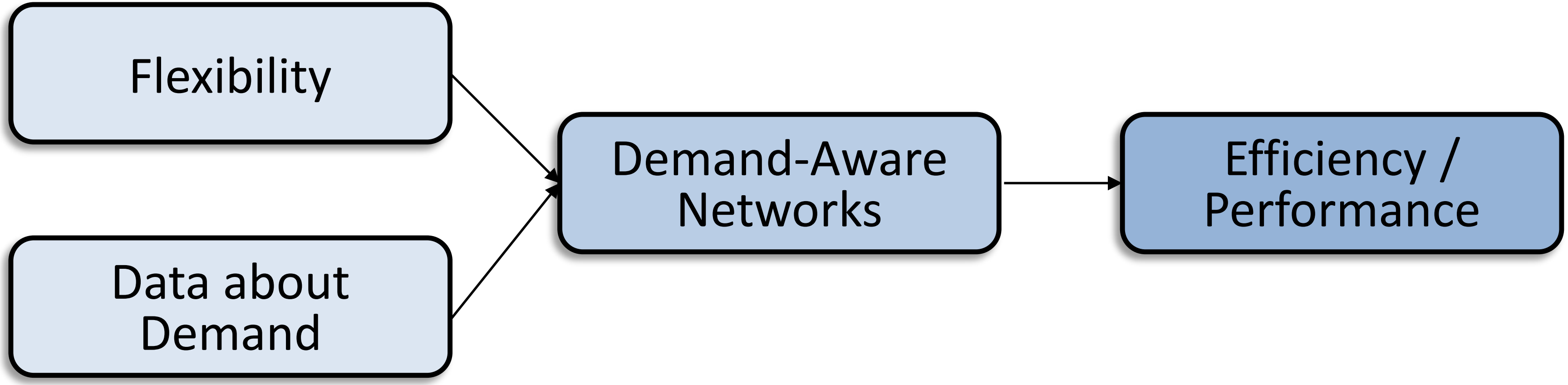
Visibility: SDN, Telemetry, Sketching

- Solution: **adversarial trajectory sampling with SDN**
- Idea:
  - Use ***secure*** channels between controller and switches to distribute hash ranges
  - Give ***different hash ranges*** hash ranges to different switches, but add some ***redundancy***: risk of being caught!
- In general: obtaining live data from the network ***becomes easier!***



Preacher: Network Policy Checker for Adversarial Environments. Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid. SRDS 2019.

Together, Enables A Paradigm Shift: Demand-Aware Networks


Flexibility
Data about Demand
Demand-Aware Networks
Efficiency / Performance

A Case Study: Flexible Topologies

Germany…

Enabling optical technologies for reconfigurable networks

Example: Manya Ghobadi et al.

Kudos for some slides!

Also for WAN!

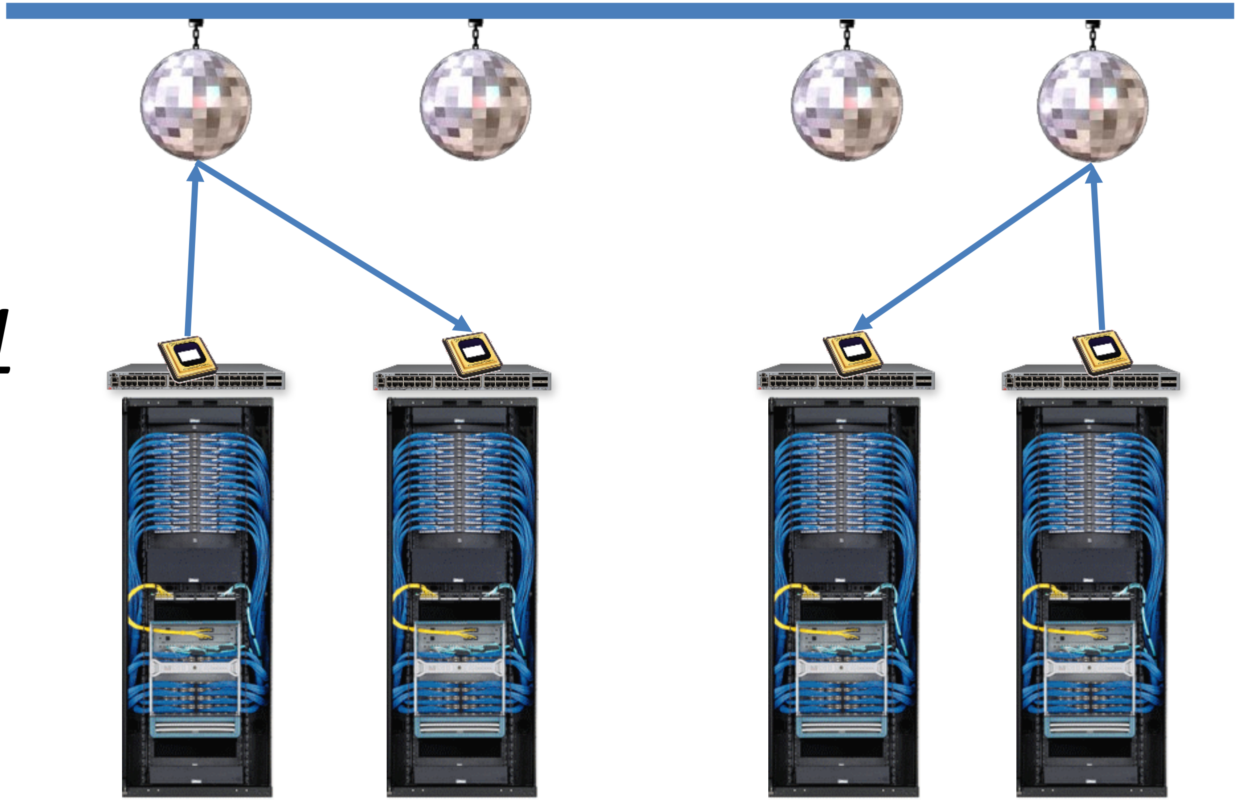
Enabling optical technologies for reconfigurable networks

Example: Manya Ghobadi et al.

Kudos for some slides!

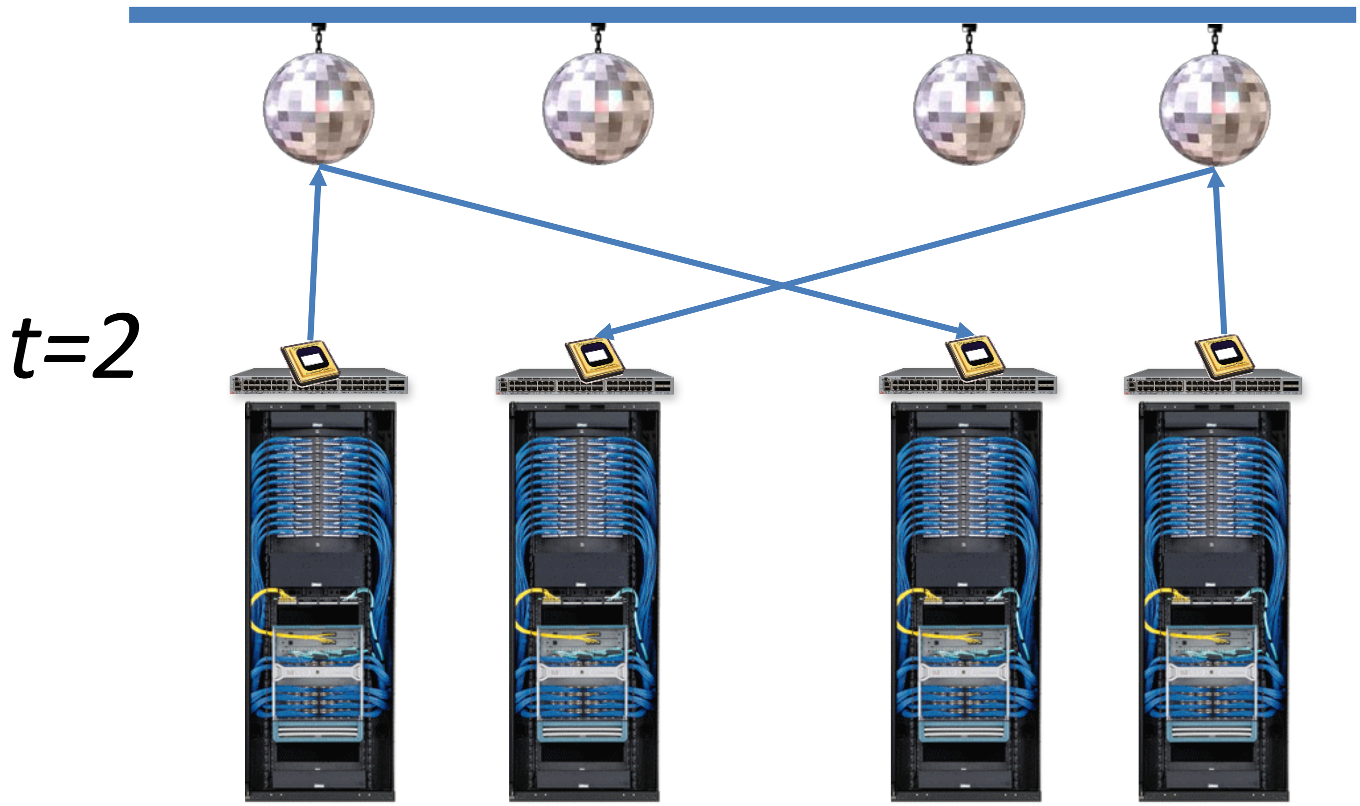
Example: ProjecToR

t=1



- Based on **free-space optics**

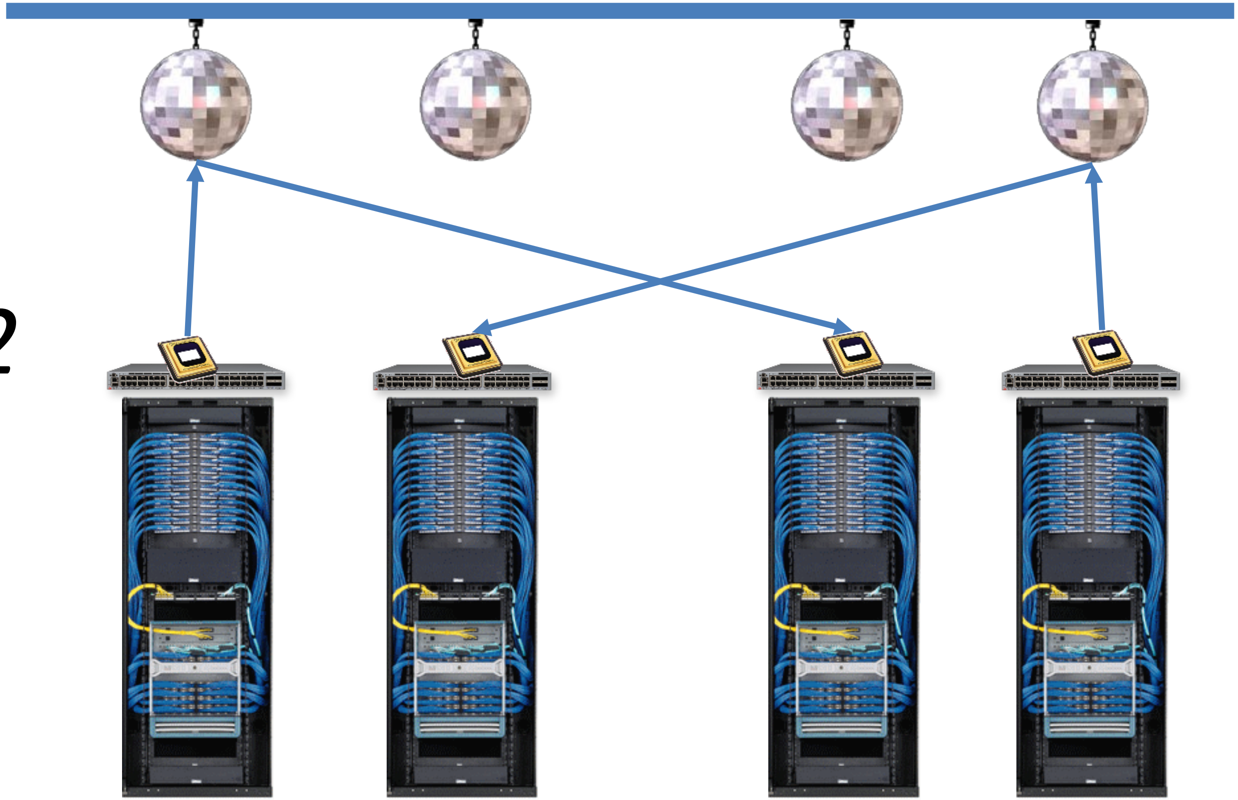
Example: ProjecToR



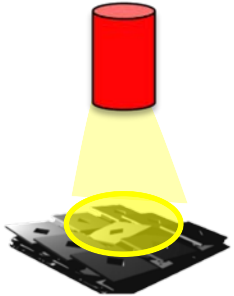
- Based on **free-space optics**

Example: ProjecToR

t=2

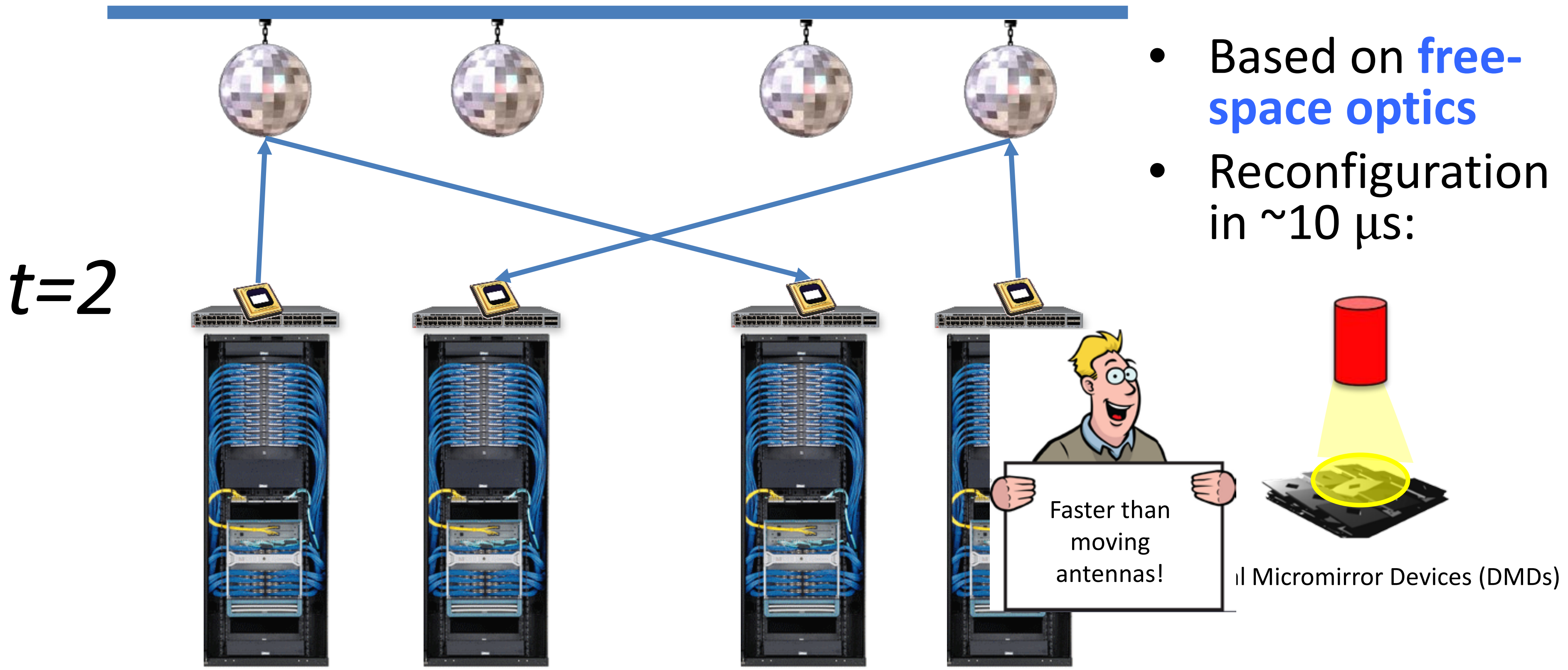


- Based on **free-space optics**
- Reconfiguration in ~10 μs:



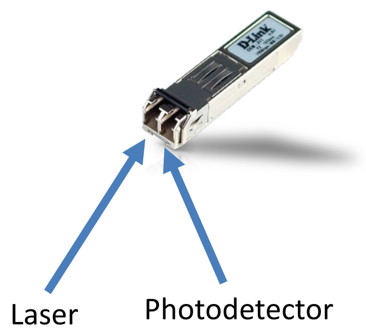
Digital Micromirror Devices (DMDs)

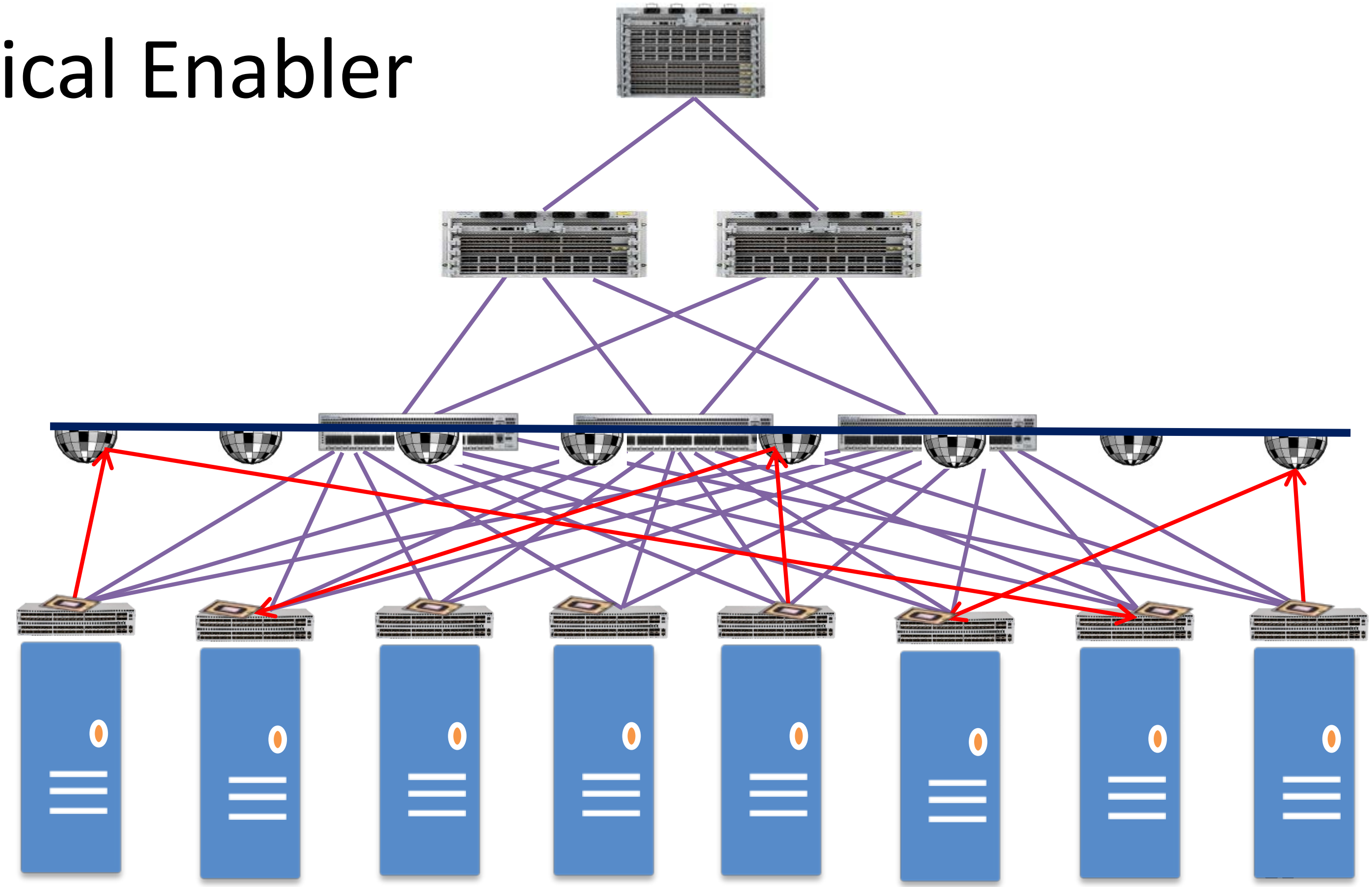
Example: ProjecToR



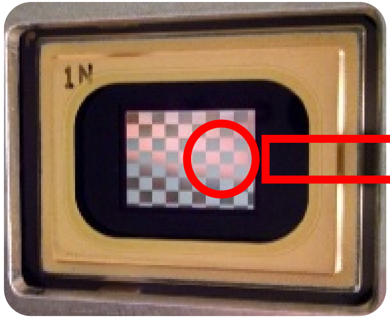
- Based on **free-space optics**
- Reconfiguration in ~10 μs:

ProjecToR in More Details: Technological Enabler

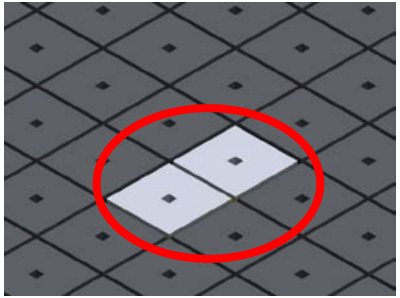




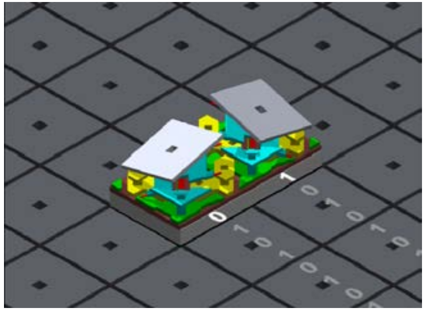
ProjecToR in More Details: DMDs



Texas Instruments



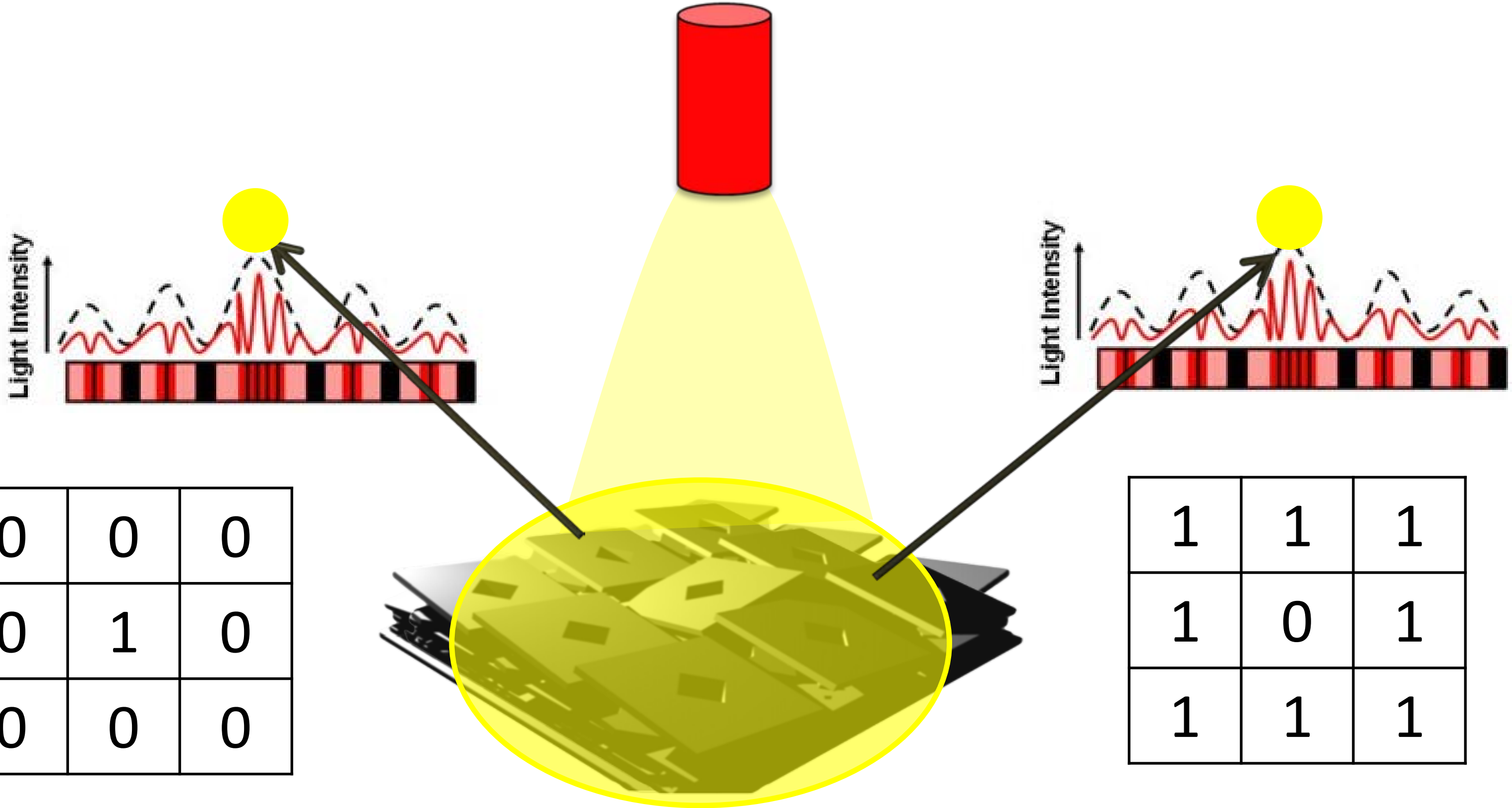
Array of micromirrors



Memory cell

- Each micromirror can be turned on/off
- Essentially a **0/1-image**: e.g., array size 768 x 1024
- Direction of the diffracted light can be finely tuned

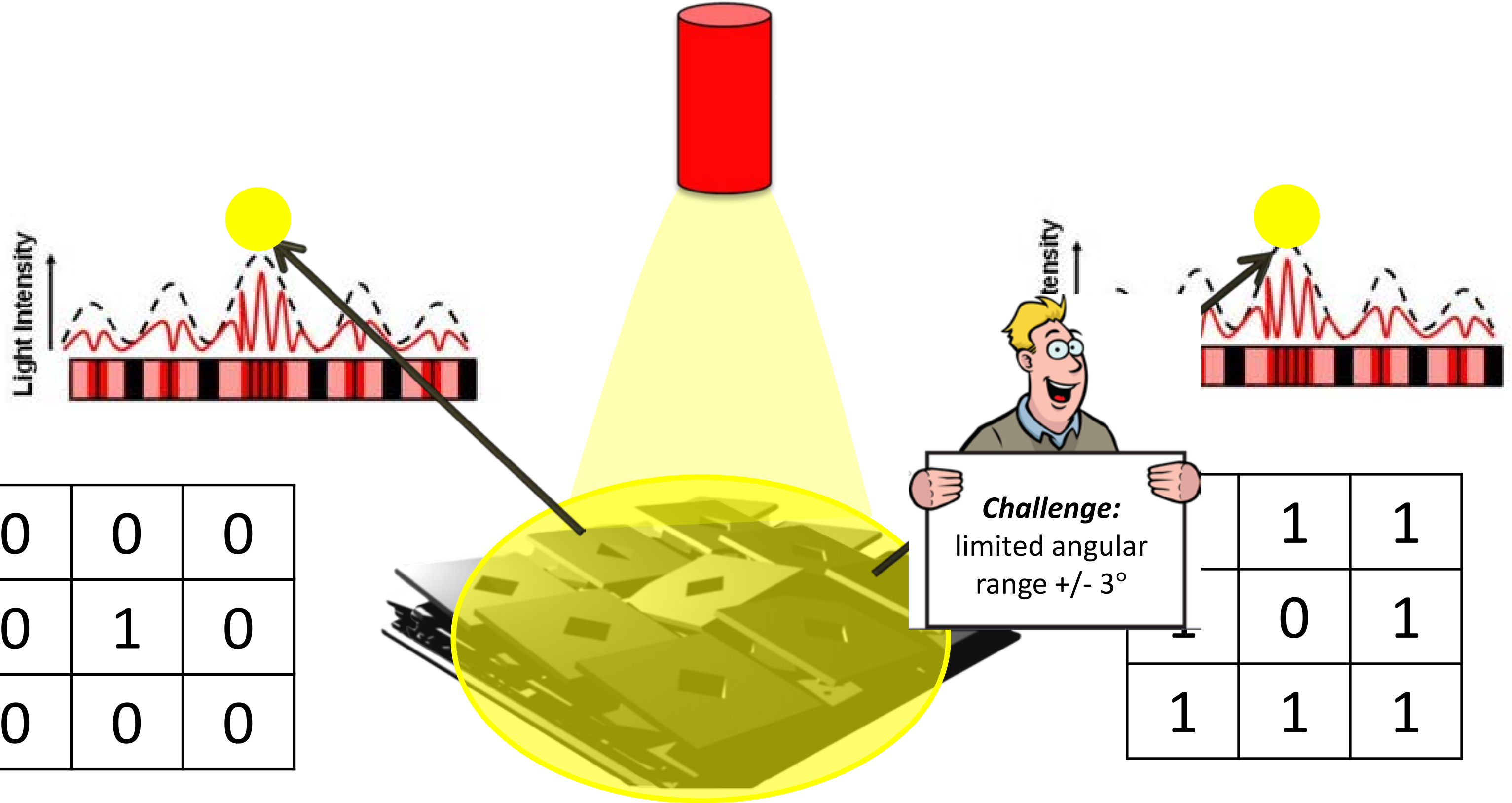
ProjecToR in More Details: DMDs to Redirect Light *Fast*



0	0	0
0	1	0
0	0	0

1	1	1
1	0	1
1	1	1

ProjecToR in More Details: DMDs to Redirect Light *Fast*



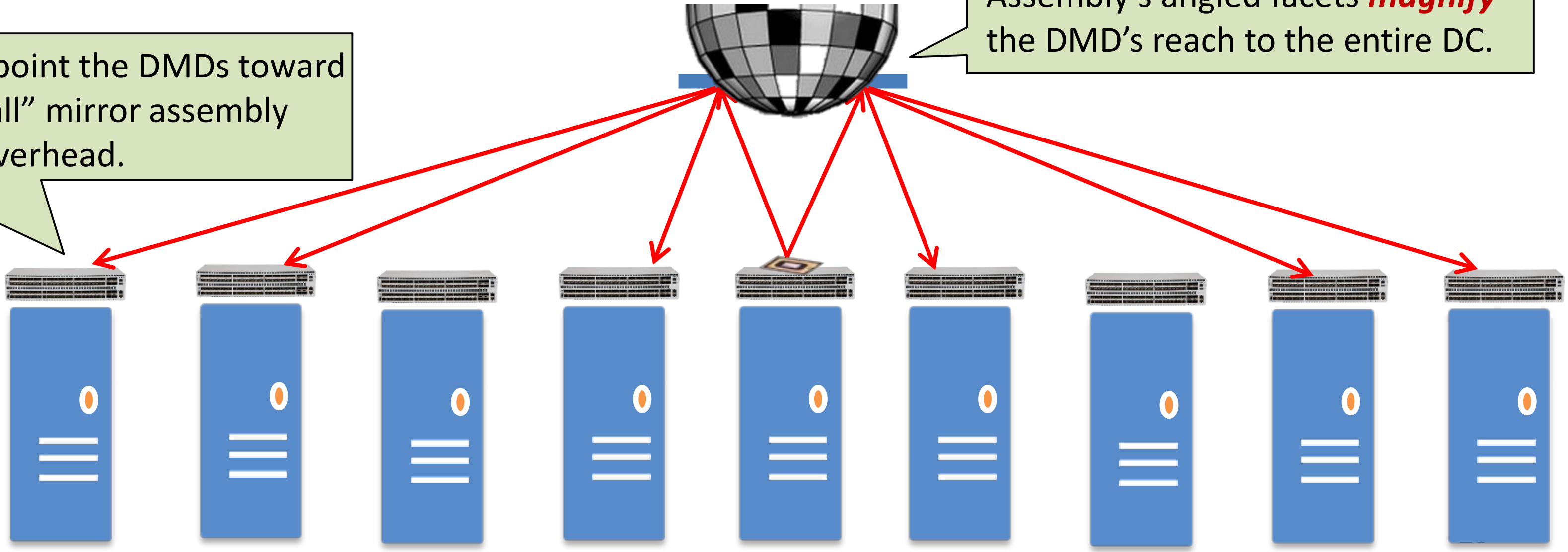
0	0	0
0	1	0
0	0	0

	1	1
	0	1
1	1	1

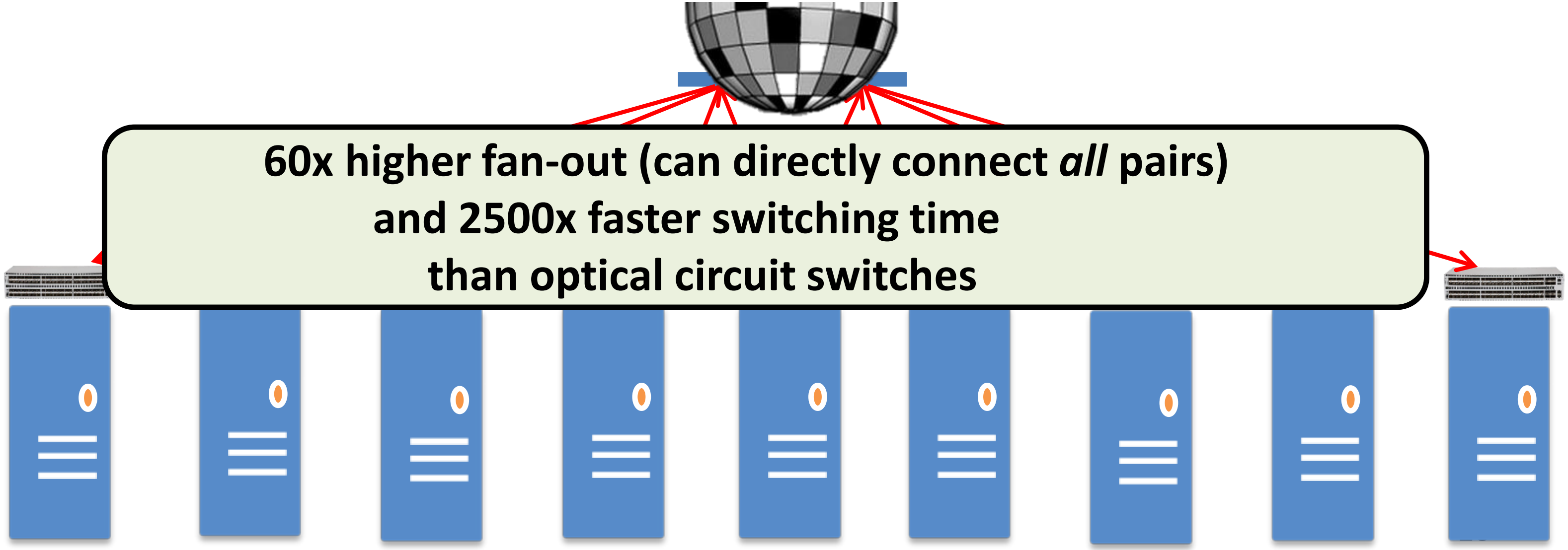
ProjecToR in More Details: Coupling DMDs with angled mirrors

Coupling: point the DMDs toward a "disco-ball" mirror assembly installed overhead.

Assembly's angled facets ***magnify*** the DMD's reach to the entire DC.



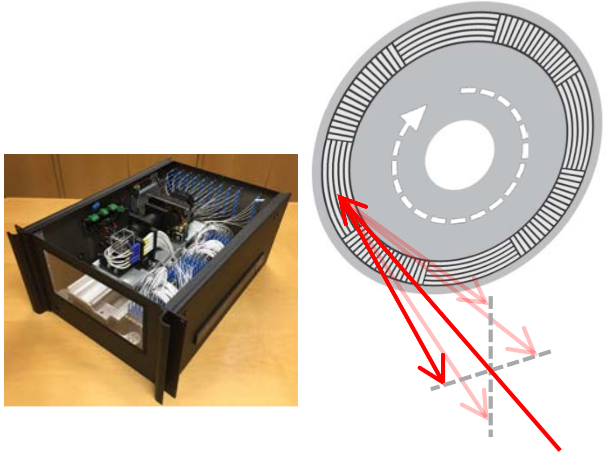
ProjecToR in More Details: Coupling DMDs with angled mirrors



Other Technologies

Based on silicon photonics

2-NEMS



Rotating disks

Further reading:
Wade et al., A Bandwidth-Dense, Low Power Electronic-Photonic Platform and Architecture for Multi-Tbps Optical I/O [OFC'18]
Porter et al., "Integrating Microsecond Circuit Switching into the Data Center", Sigcomm'13

Timeline

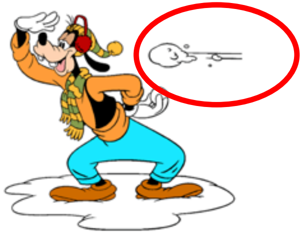
Reconfiguration time: from milliseconds ***to microseconds*** (and decentralized).

Survey of Reconfigurable Data Center Networks. Foerster and Schmid. SIGACT News, 2019.

- 2009
  - *Flyways* [51]: Steerable antennas (narrow beamwidth at 60 GHz [78]) to serve hotspots
- 2010
  - *Helios* [33]/*c-Through* [98, 99]: Hybrid switch architecture, maximum matching (Edmond's algorithm [30]), single-hop reconfigurable connections ($O(10)ms$ reconfiguration time).
  - *Proteus* [21, 89]: k reconfigurable connections per ToR, multi-hop path stitching, multi-hop reconfigurable connections (weighted b-matching [69], edge-exchanges for connectivity [72], wavelength assignment via edge-coloring [67] on multigraphs)
- 2011
  - Extension of *Flyways* [51] to better handle practical concerns such as stability and interference for 60GHz links, along with greedy heuristics for dynamic link placement [45]
- 2012
  - *Mirror Mirror on the ceiling* [106]: 3D-beamforming (60 Ghz wireless), signals bounce off the ceiling
- 2013
  - *Mordia* [31, 32, 77]: Traffic matrix scheduling, matrix decomposition (Birkhoff-von-Neumann (BvN) [18, 97]), fiber ring structure with wavelengths ($O(10)\mu s$ reconfiguration time)
  - *SplayNets* [6, 76, 82]: Fine-grained and online reconfigurations in the spirit of self-adjusting datastructures (all links are reconfigurable), aiming to strike a balance between short route lengths and reconfiguration costs
- 2014
  - *REACToR* [56]: Buffer burst of packets at end-hosts until circuit provisioned, employs [77]
  - *Firefly* [14] Combination of Free Space Optics and Galvo/switchable mirrors (small fan-out)
- 2015
  - *Solstice* [57]: Greedy perfect matching based hybrid scheduling heuristic that outperforms BvN [77]
  - Designs for optical switches with a reconfiguration latency of $O(10)ns$ [3]
- 2016
  - *ProjecToR* [39]: Distributed Free Space Optics with digital micromirrors (high fan-out) [38] (Stable Matching [26]), goal of (starvation-free) low latency
  - *Eclipse* [95, 96]: $(1 - 1/e^{(1-\varepsilon)})$-approximation for throughput in traffic matrix scheduling (single-hop reconfigurable connections, hybrid switch architecture), outperforms heuristics in [57]
- 2017
  - *DAN* [7, 8, 11, 12]: Demand-aware networks based on reconfigurable links only and optimized for a demand snapshot, to minimized average route length and/or minimize load
  - *MegaSwitch* [23]: Non-blocking circuits over multiple fiber rings (stacking rings in [77] doesn't suffice)
  - *Rotornet* [63]: Oblivious cyclical reconfiguration w. selector switches [64] (Valiant load balancing [94])
  - *Tale of Two Topologies* [105]: Convert locally between Clos [24] topology and random graphs [87, 88]
- 2018
  - *DeepConf* [81]/*xWeaver* [102]: Machine learning approaches for topology reconfiguration
- 2019
  - Complexity classifications for weighted average path lengths in reconfigurable topologies [34, 35, 36]
  - *ReNet* [13] and *Push-Down-Trees* [9] providing statically and dynamically optimal reconfigurations
  - *DisSplayNets* [75]: fully decentralized *SplayNets*
  - *Opera* [60]: Maintaining expander-based topologies under (oblivious) reconfiguration

When Are Demand-Aware Networks Useful?

A Simple Answer

Demand-Oblivious Networks = 

Seriously: We believe, often, in practice!

	A	B	C	D
A	0	3	3	3
B	3	0	3	3
C	3	3	0	3
D	3	3	3	0

In theory: traffic matrix uniform and static

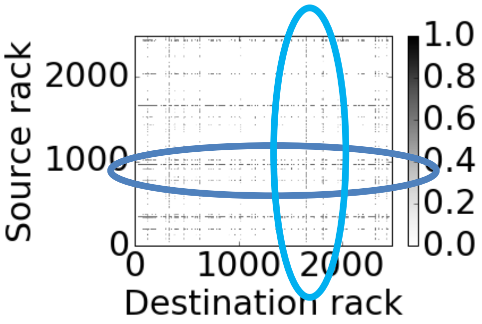
	A	B	C	D
A	0	0 / 6	0 / 6	0
B	0	0	0	0 / 7
C	0 / 8	0	0 / 6	0 / 7
D	0	0 / 12	0 / 8	0

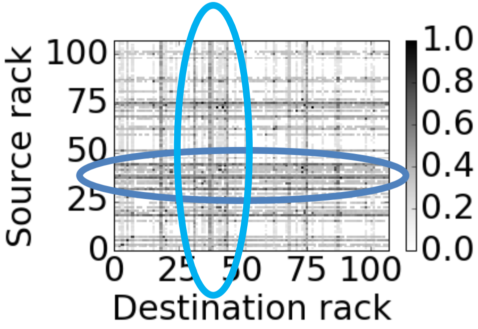
In practice: **skewed** and **dynamic**

Empirical Motivation

Observation 1:

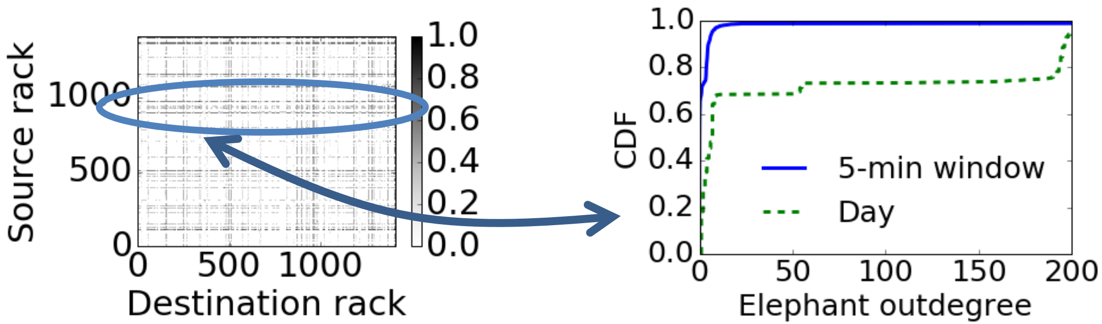
- Many rack pairs exchange ***little traffic***
- Only some ***hot rack pairs*** are active





Observation 2:

- Some source racks send large amounts of traffic ***to many other racks***



Microsoft data: 200K servers across 4 production clusters, cluster sizes: 100 - 2500 racks.
Mix of workloads: MapReduce-type jobs, index builders, database and storage systems.

So: How *much* structure is there?

How to ***measure*** it?
And which ***types of structures***? E.g., **temporal** structure in addition to **non-temporal** structure?
Tricky!

Often only intuitions in the literature…

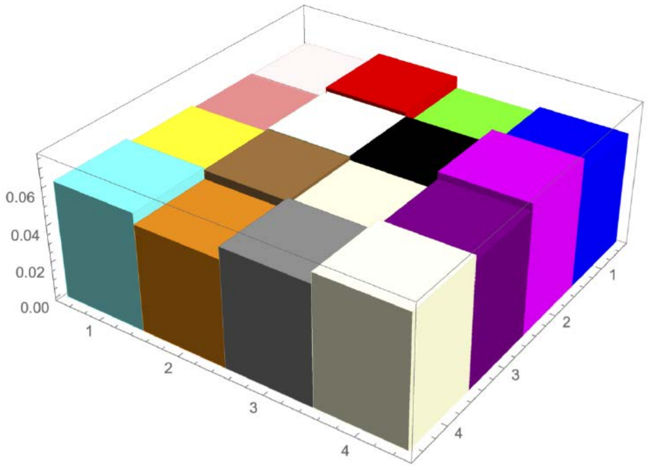
“less than 1% of the rack pairs account for 80% of the total traffic”

“only a few ToRs switches are hot and most of their traffic goes to a few other ToRs”

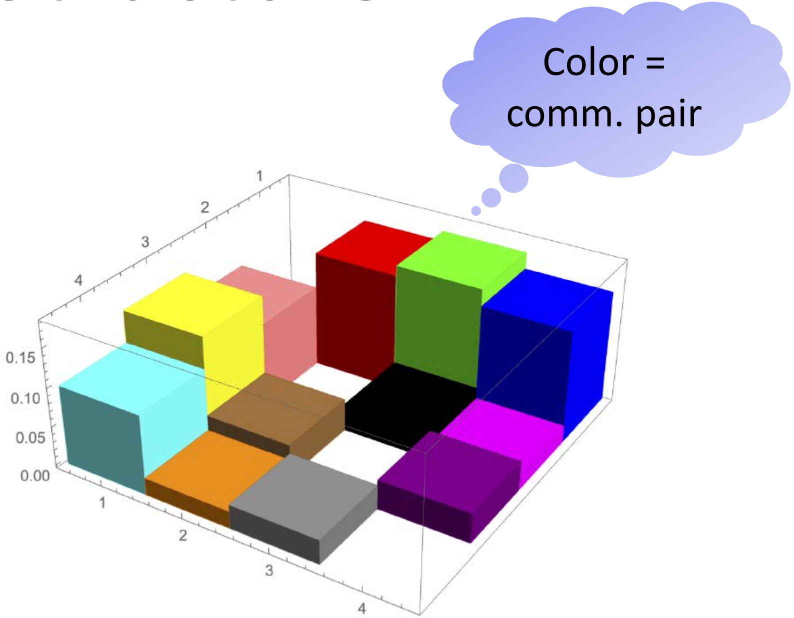
“over 90% bytes flow in elephant flows”

… and it *is* intuitive!

Non-temporal Structure



VS

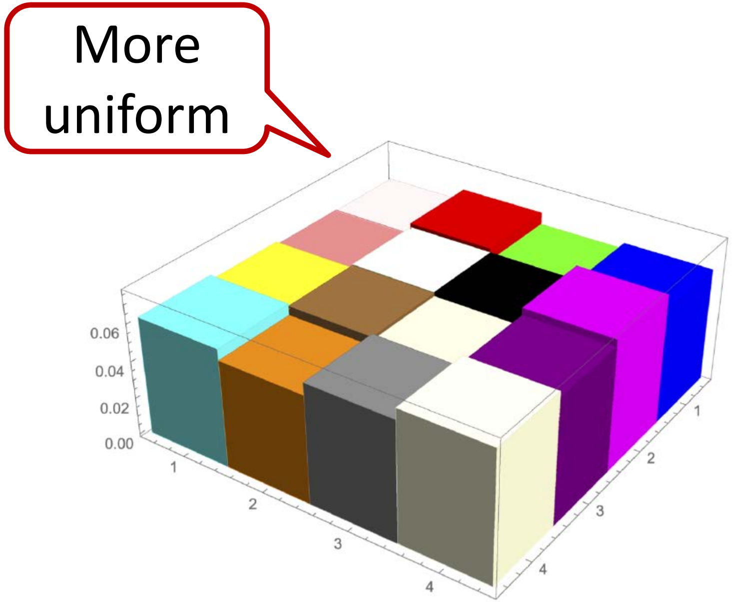


Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

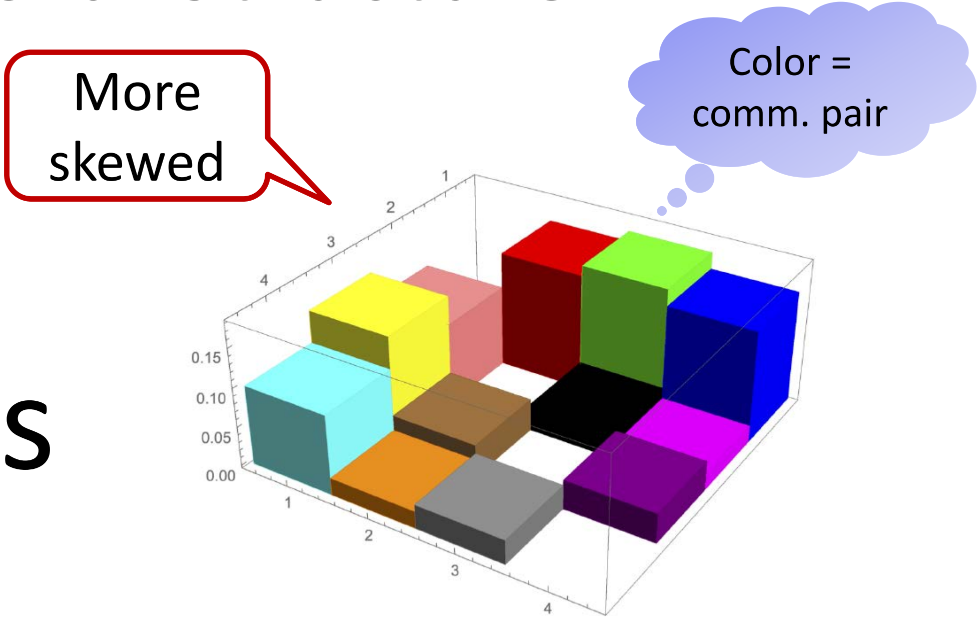
Which one has *more structure*?

… and it *is* intuitive!

Non-temporal Structure



VS

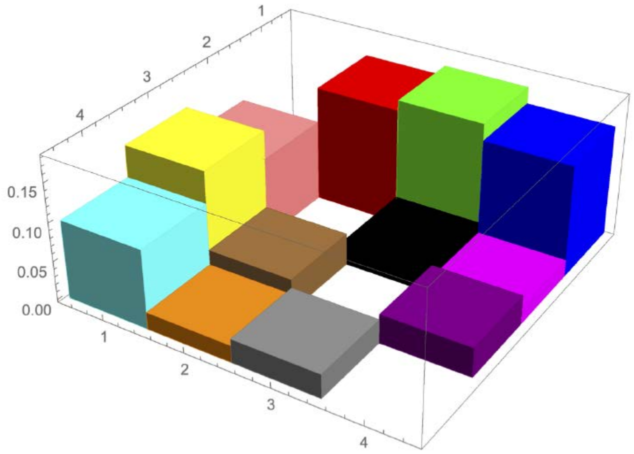


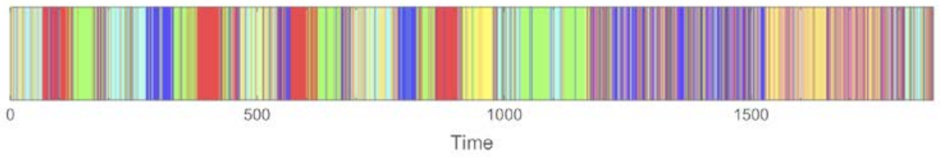
Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

Which one has *more structure*?

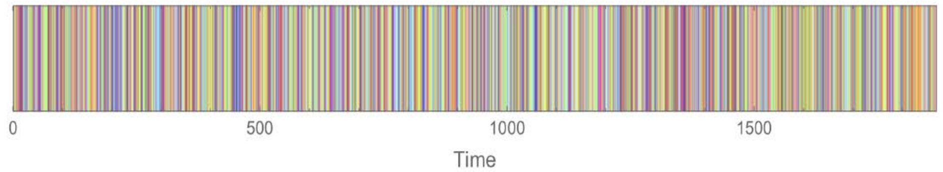
… and it *is* intuitive!

Temporal Structure





VS

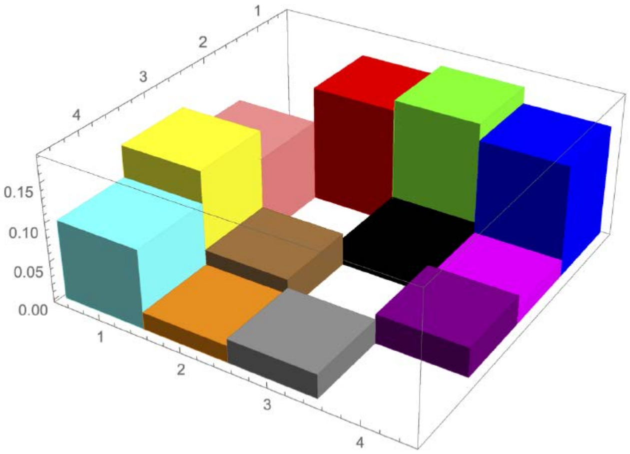


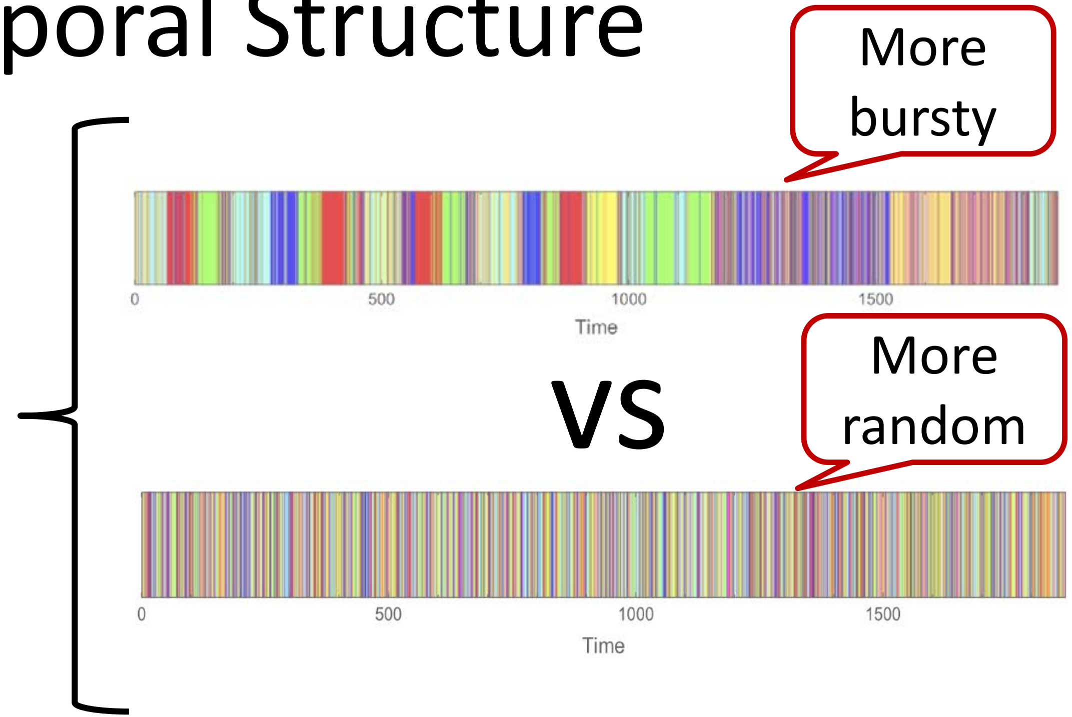
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

… and it *is* intuitive!

Temporal Structure



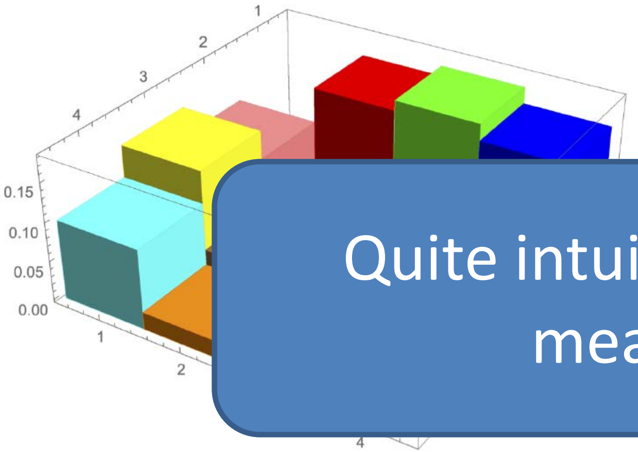


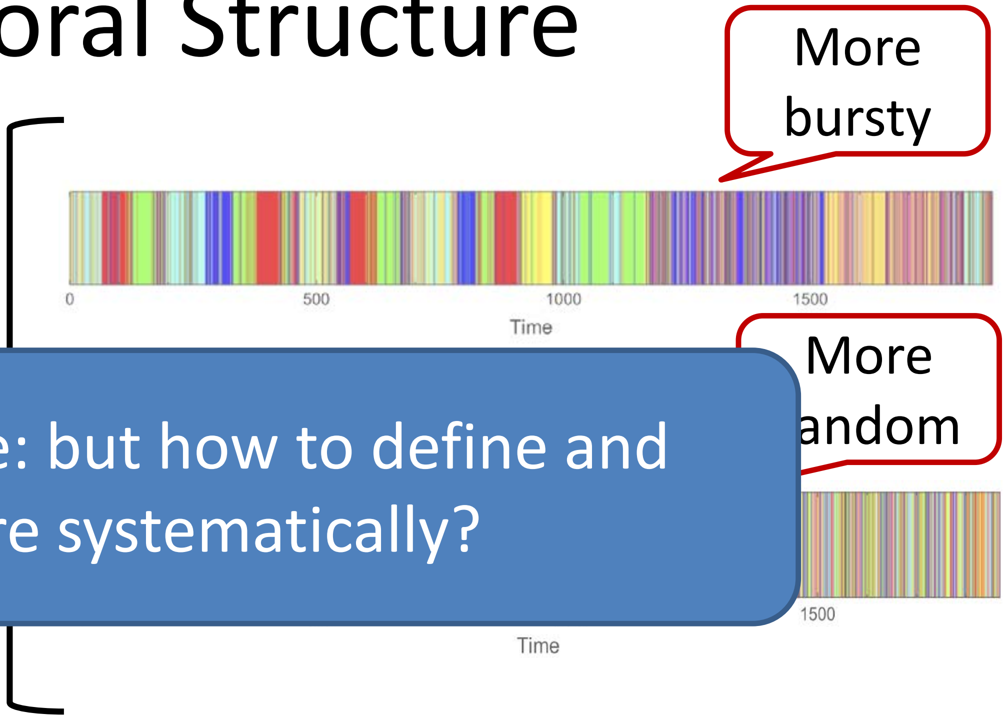
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

… and it *is* intuitive!

Temporal Structure





Quite intuitive: but how to define and measure systematically?

Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

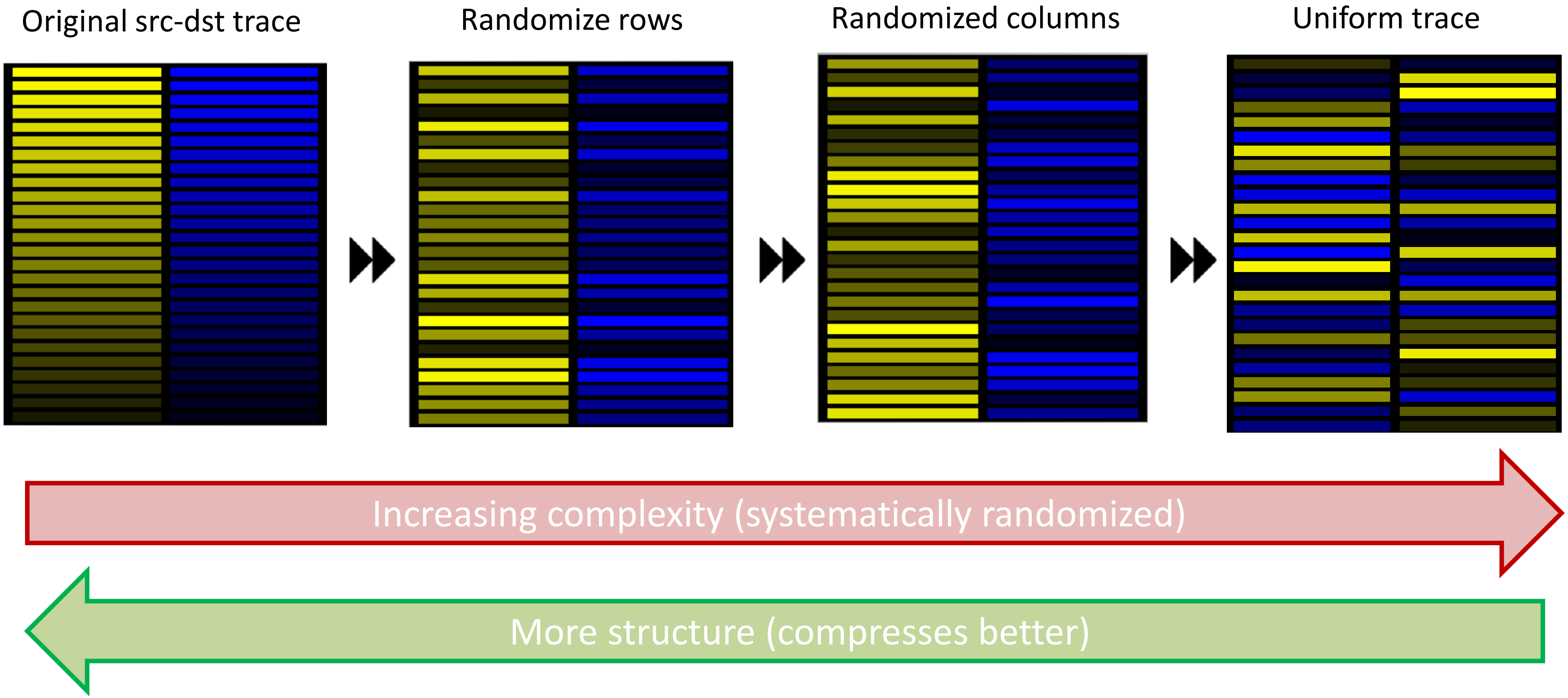
The Trace Complexity

- An **information-theoretic** approach: how can we ***measure the entropy*** (rate) of a traffic trace?

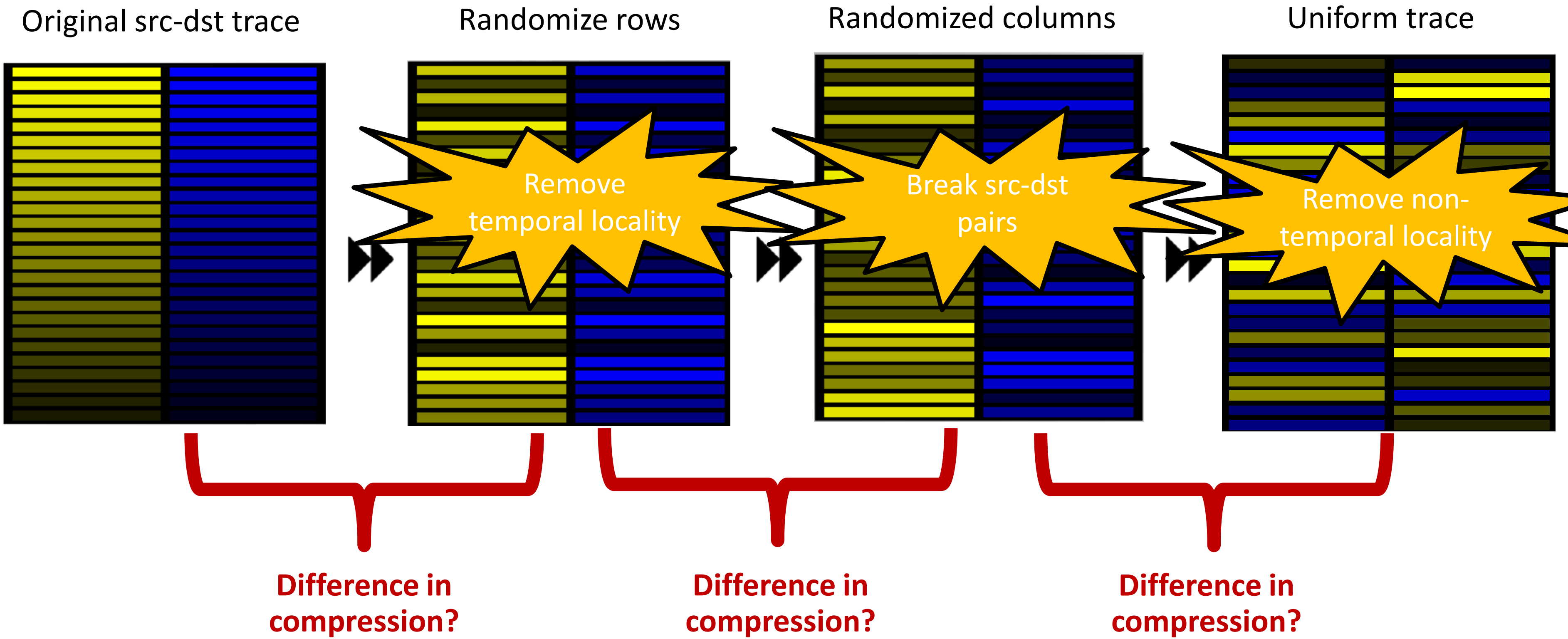
- Henceforth called the **trace complexity**

- Simple approximation: „**shuffle&compress**“
  - Remove structure by iterative *randomization*
  - Difference of compression *before and after* randomization: structure

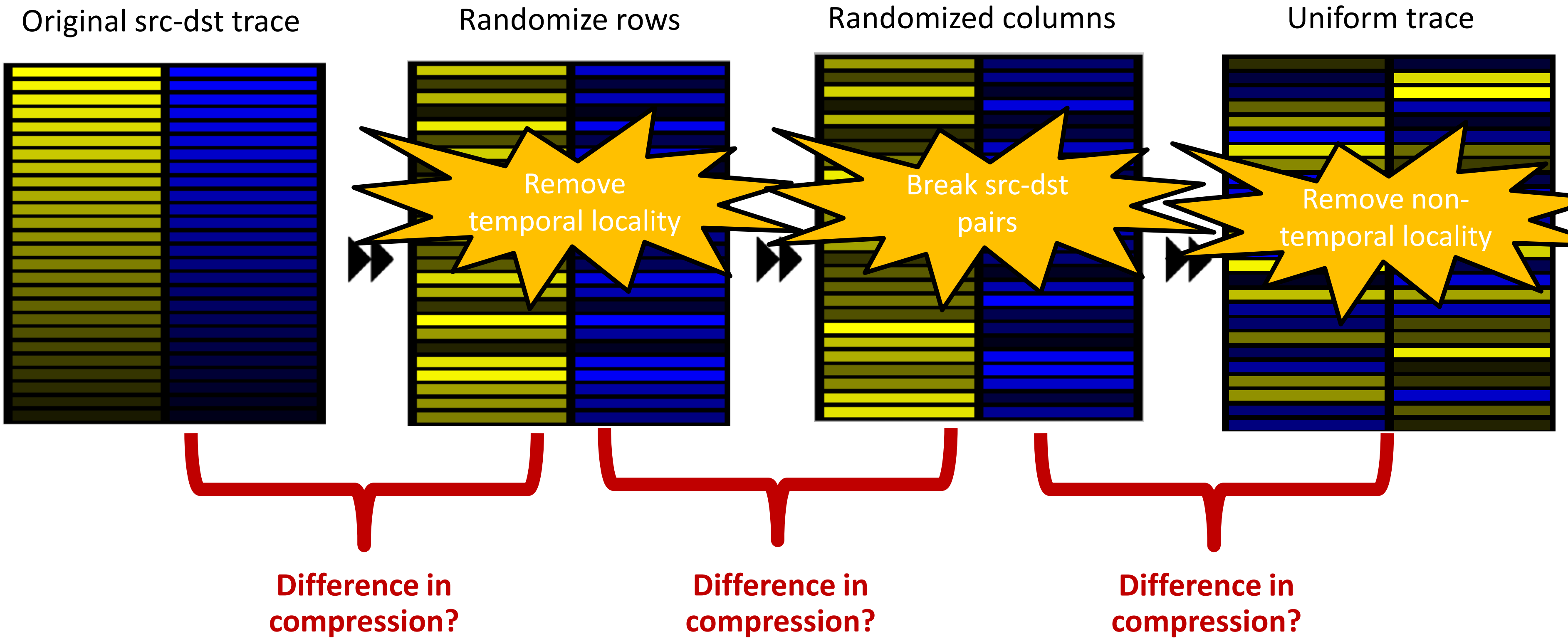
The Trace Complexity



The Trace Complexity

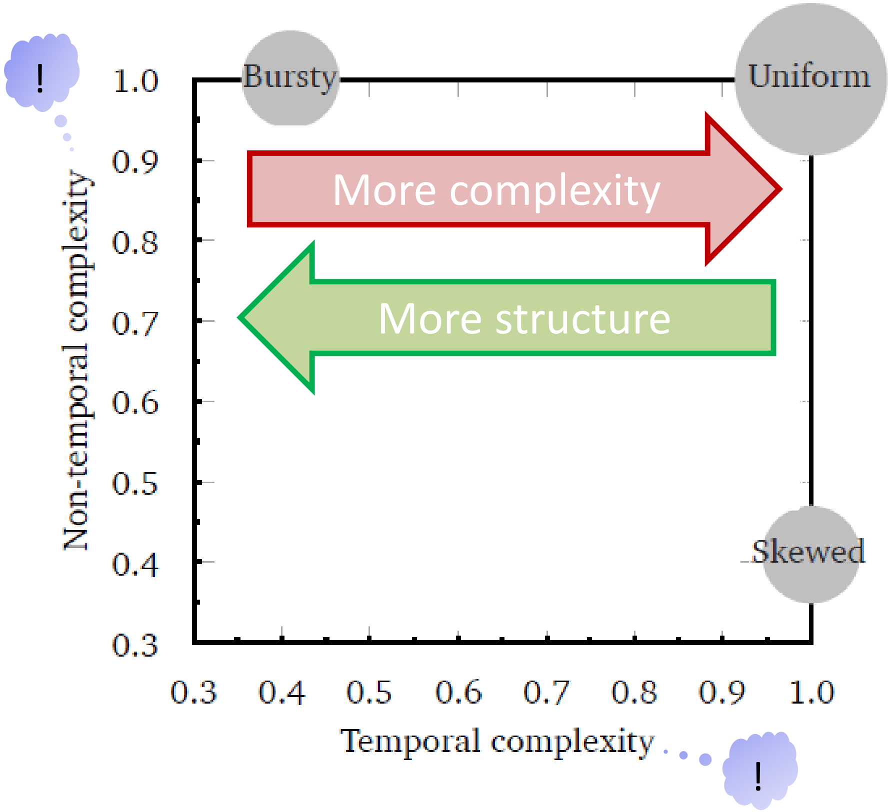


The Trace Complexity



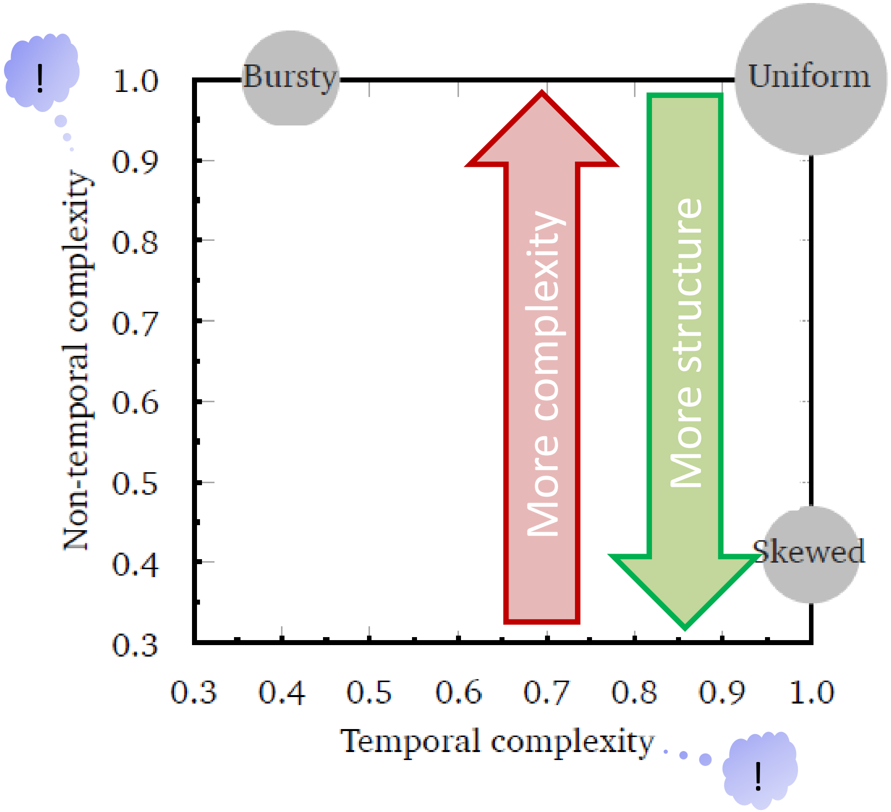
Can be used to define a „complexity map“!

The Complexity Map



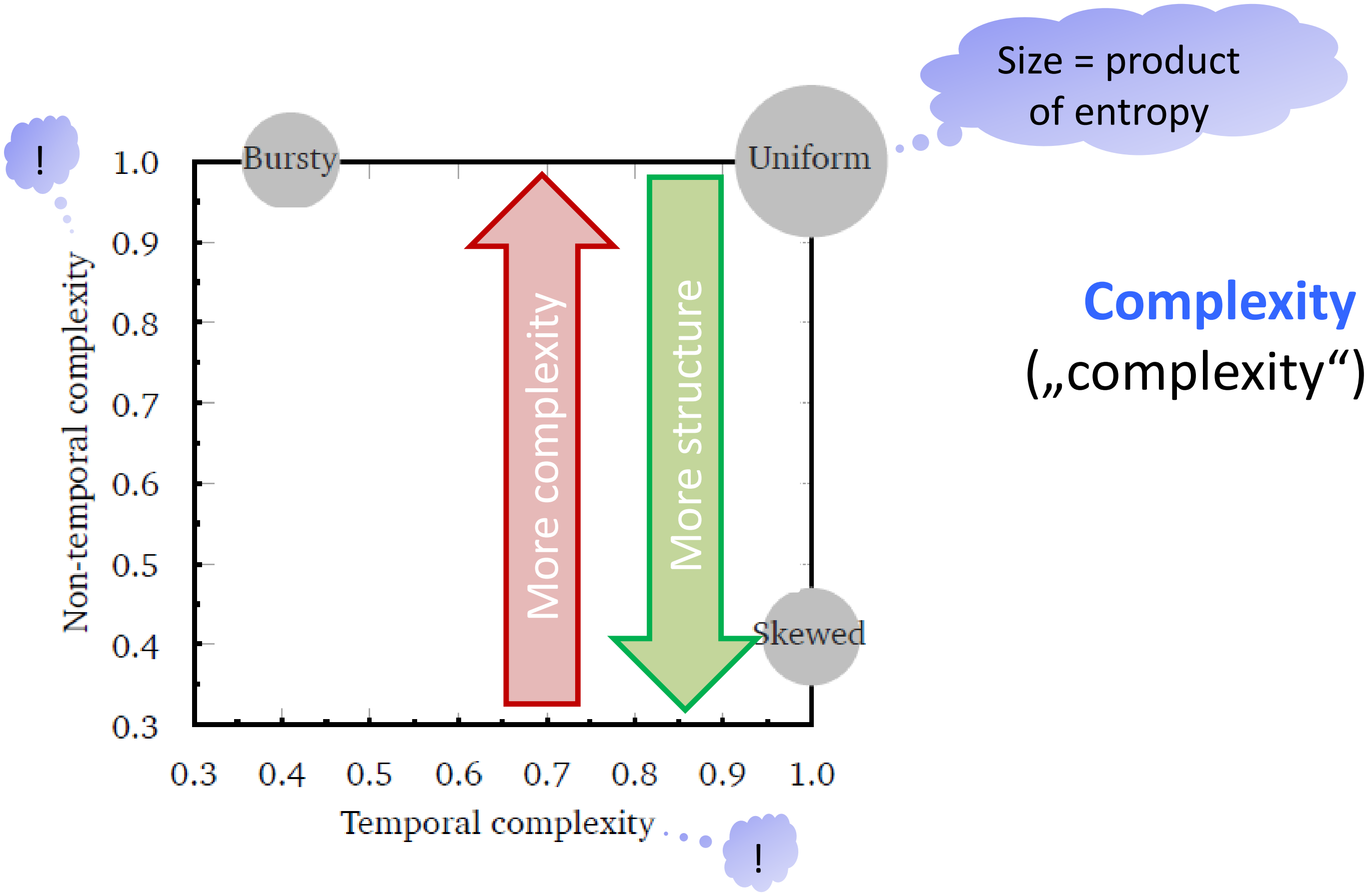
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



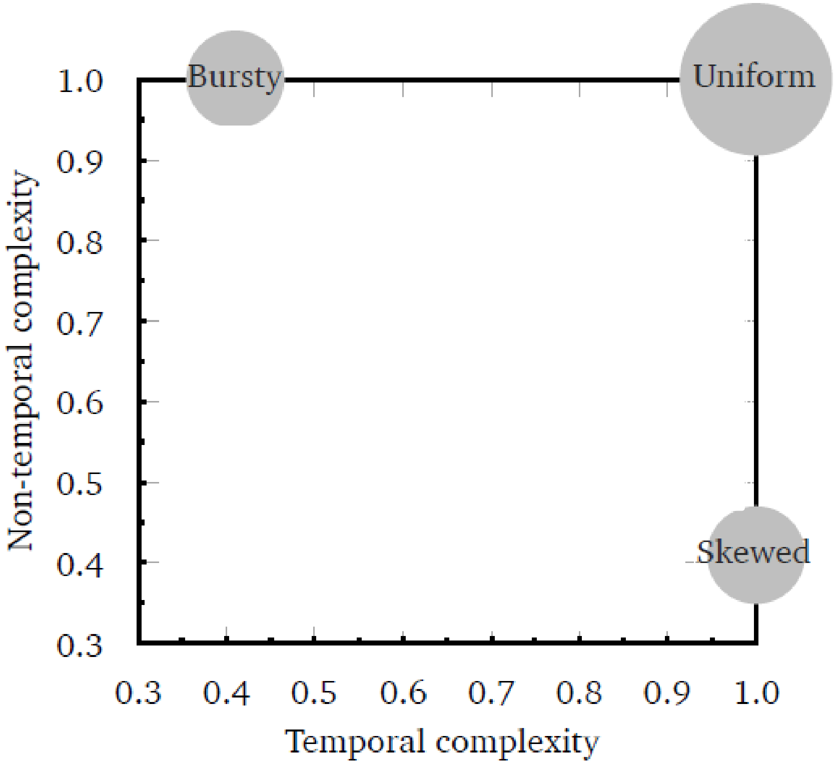
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



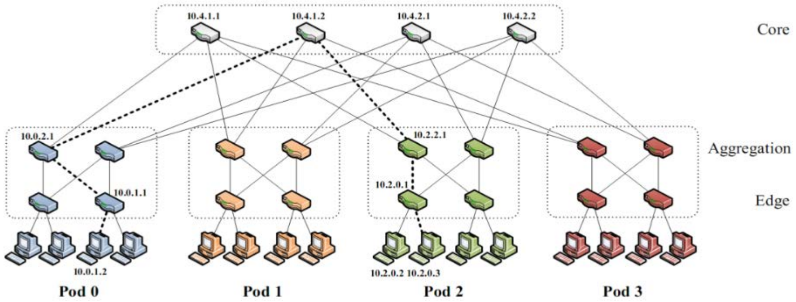
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map

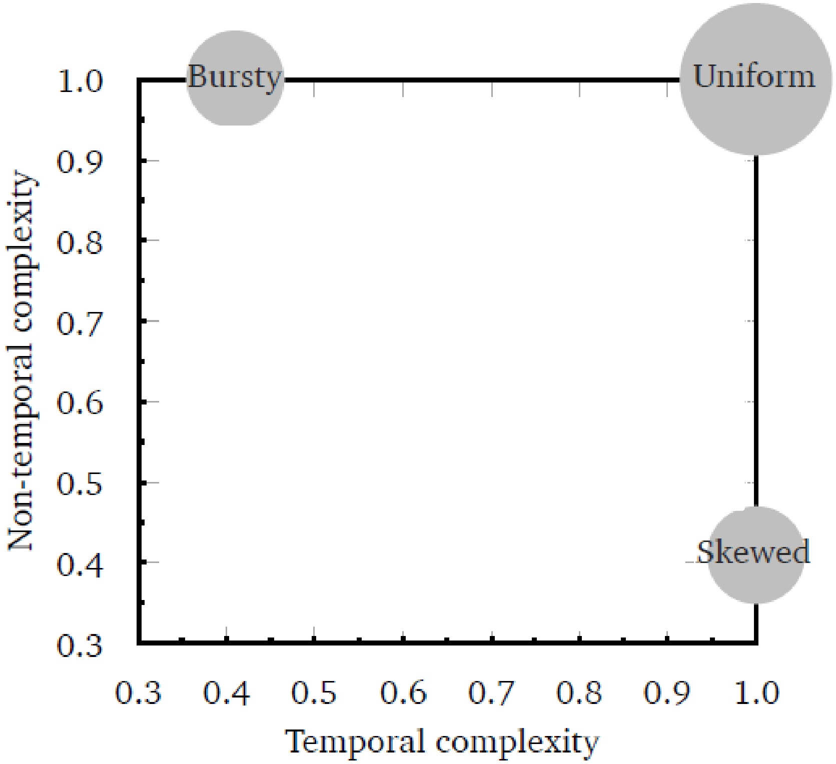


Uniform: Today's datacenters

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

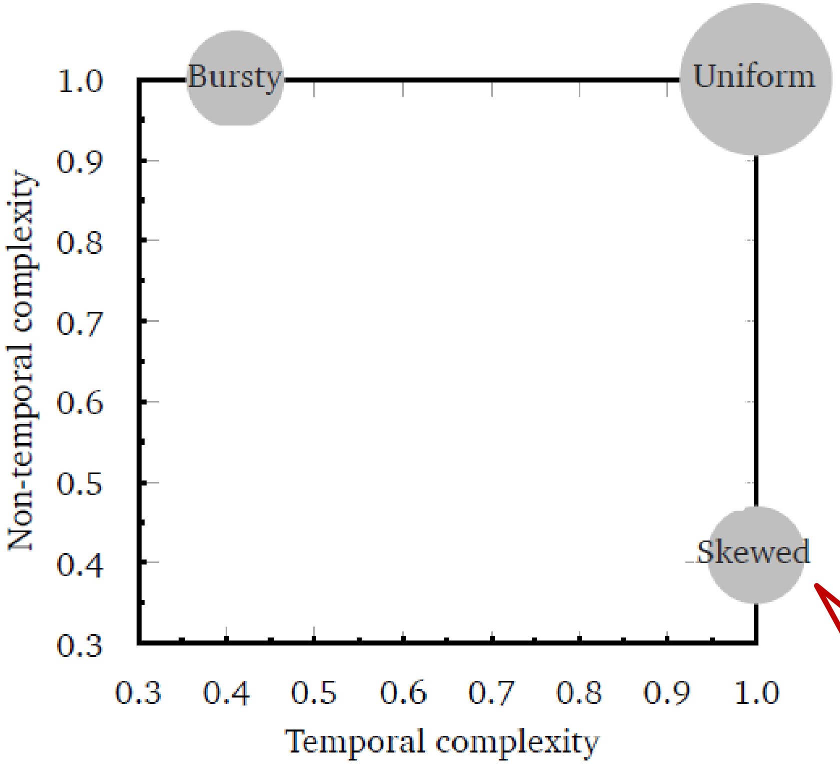


The Complexity Map



Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

The Complexity Map

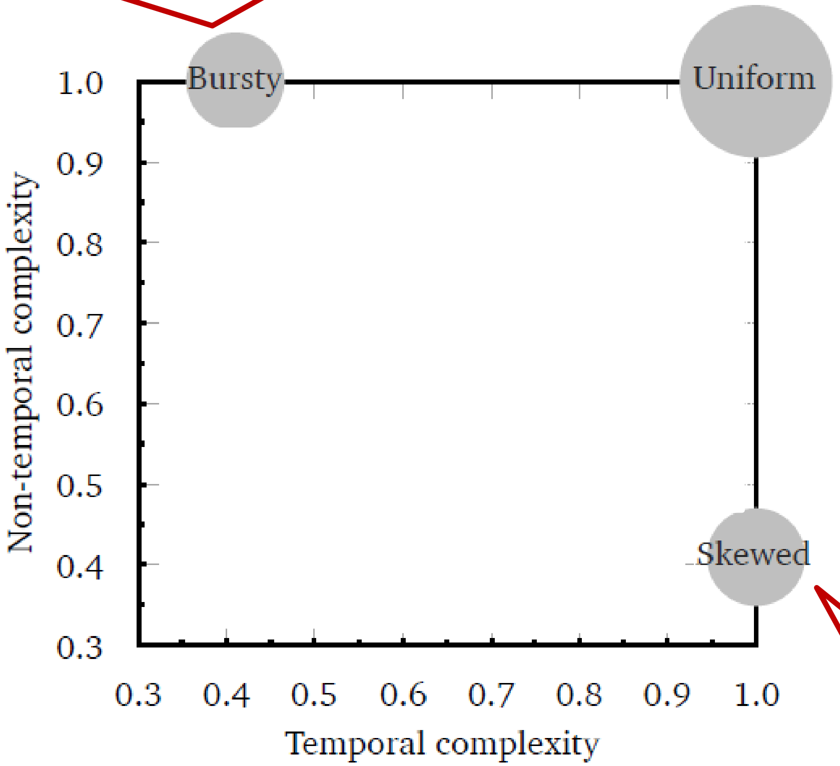


Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

Non-temporal structure could be exploited already with ***static demand-aware networks***!

To exploit **temporal** structure, need ***adaptive demand-aware ("self-adjusting") networks***.

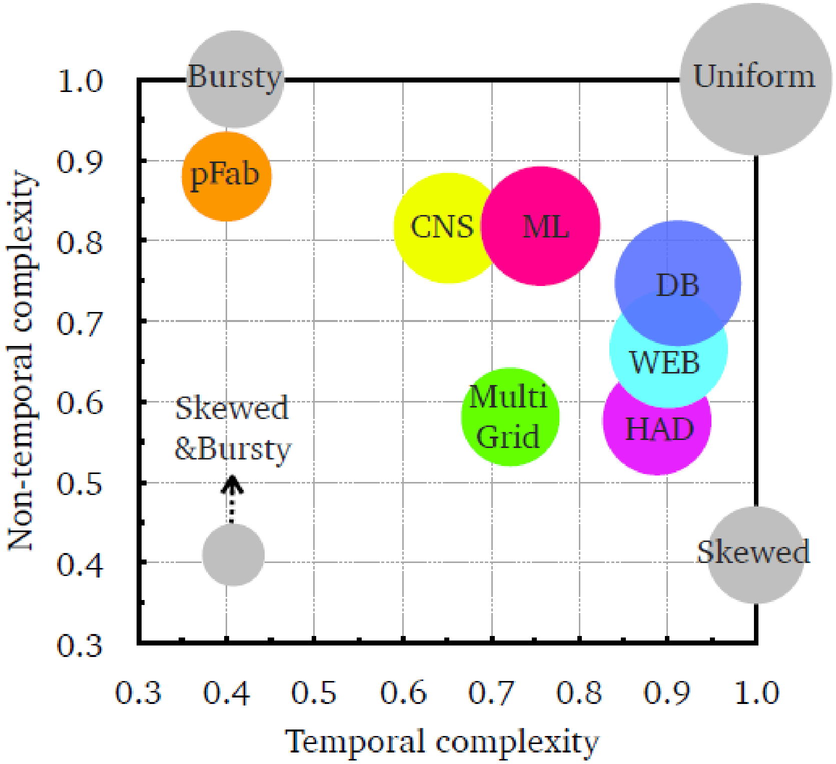
...plexity Map



Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

Non-temporal structure could be exploited already with ***static demand-aware networks***!

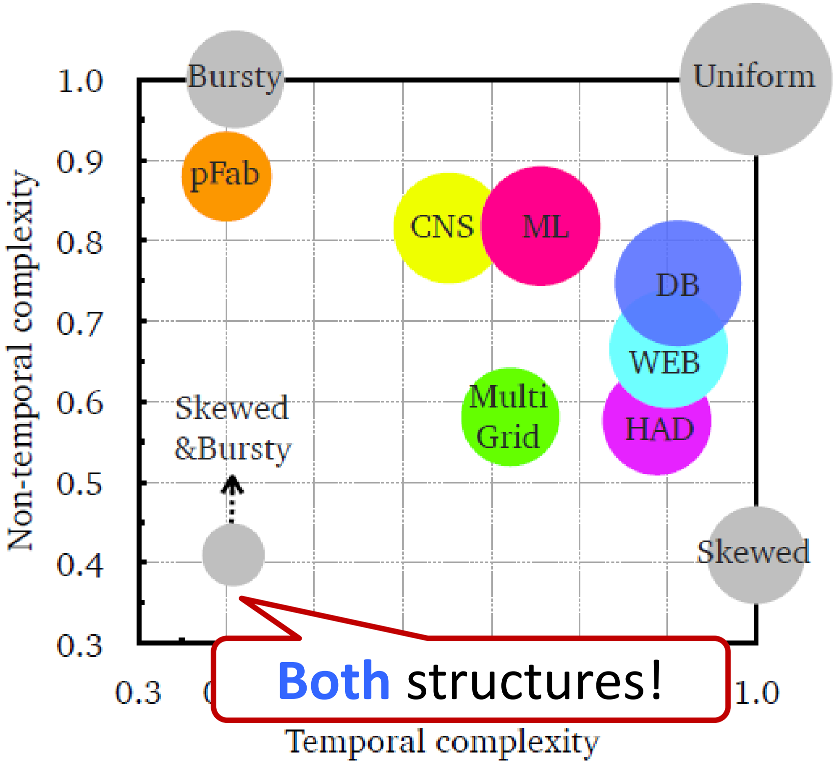
The Complexity Map



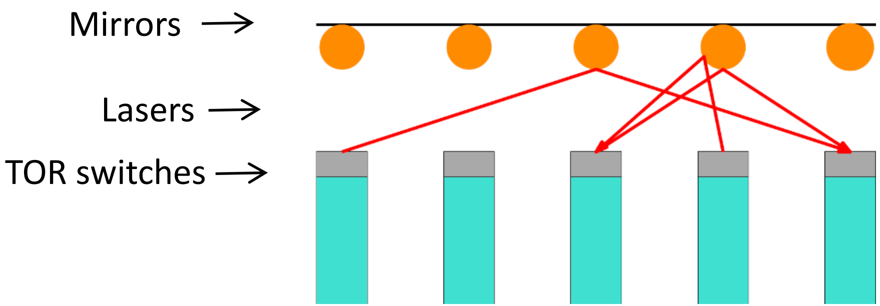
Observation: different applications feature quite significant (and different!) **temporal** and **non-temporal** structures.

- **Facebook** clusters: DB, WEB, HAD
- **HPC** workloads: CNS, Multigrid
- Distributed **Machine Learning** (ML)
- Synthetic traces like **pFabric**

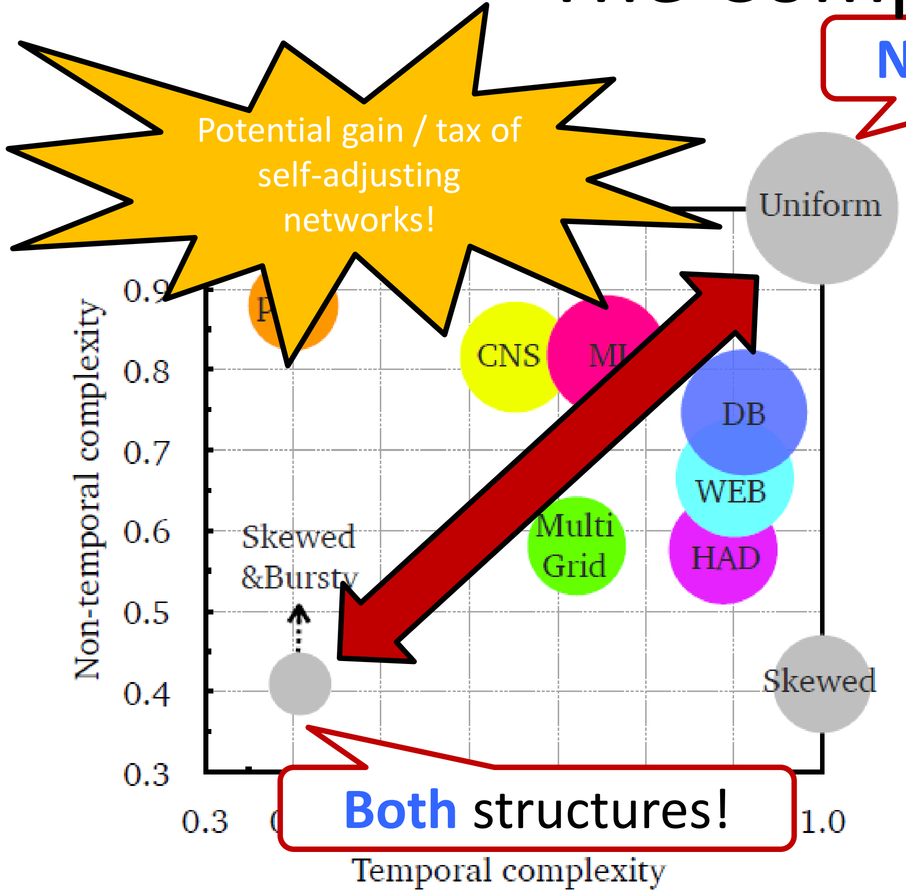
The Complexity Map



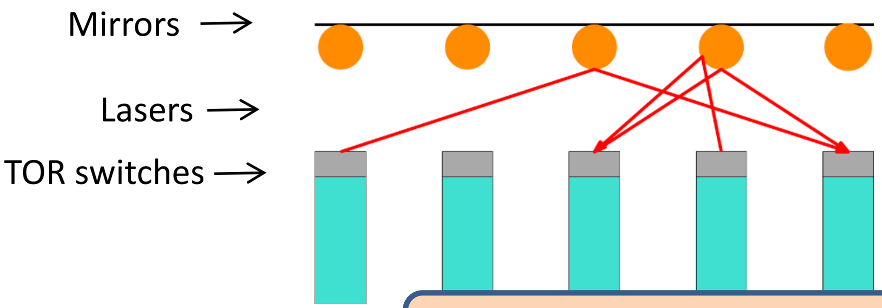
Goal: Design **self-adjusting networks** which leverage ***both*** dimensions of structure!



The Complexity Map



Goal: Design **self-adjusting networks** which leverage ***both*** dimensions of structure!



Measuring the Complexity of Packet Traces. Avin, Ghobadi, Griner, Schmid. **ArXiv** 2019.

So: How to design networks which exploit this structure? How good can they be?

Metrics again!

Roadmap

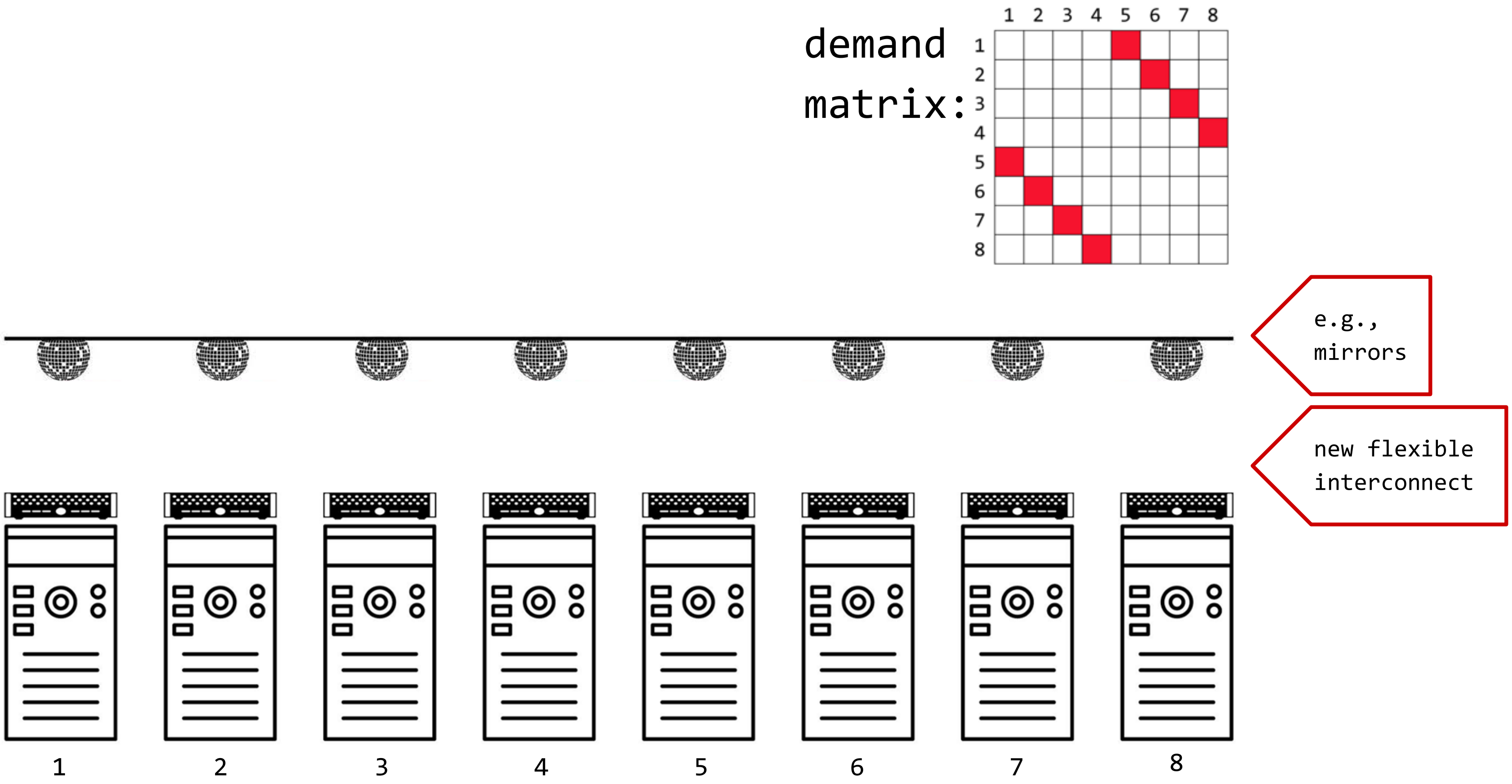
- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures

Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures

A Simple Example

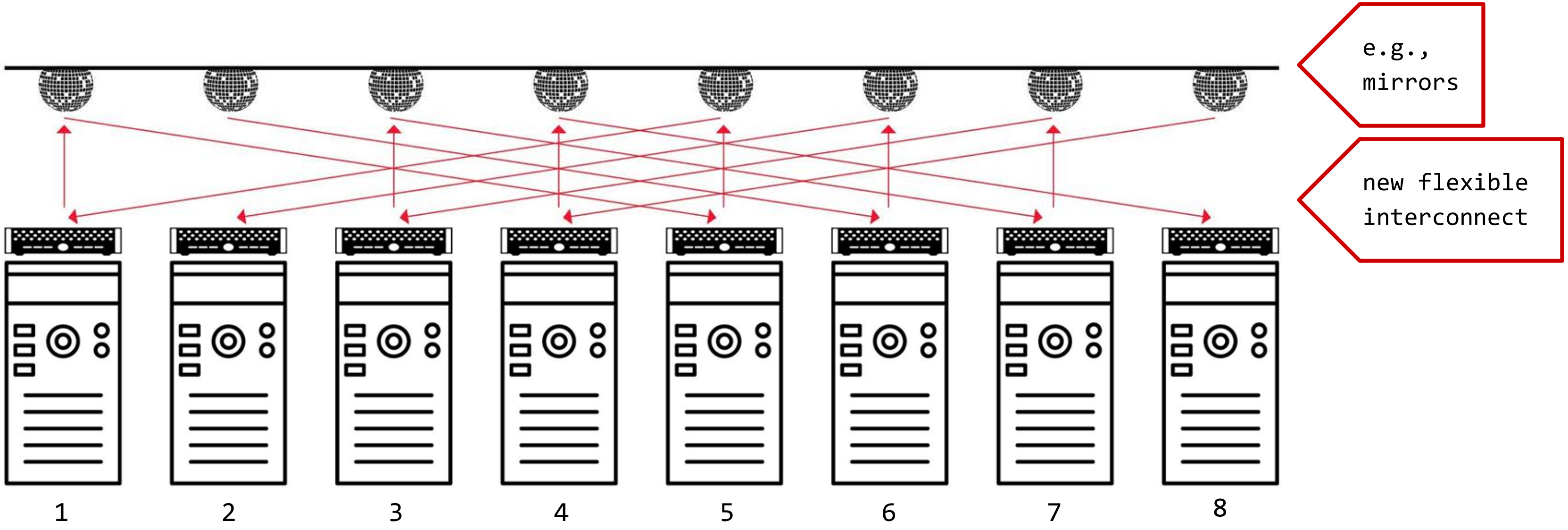
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

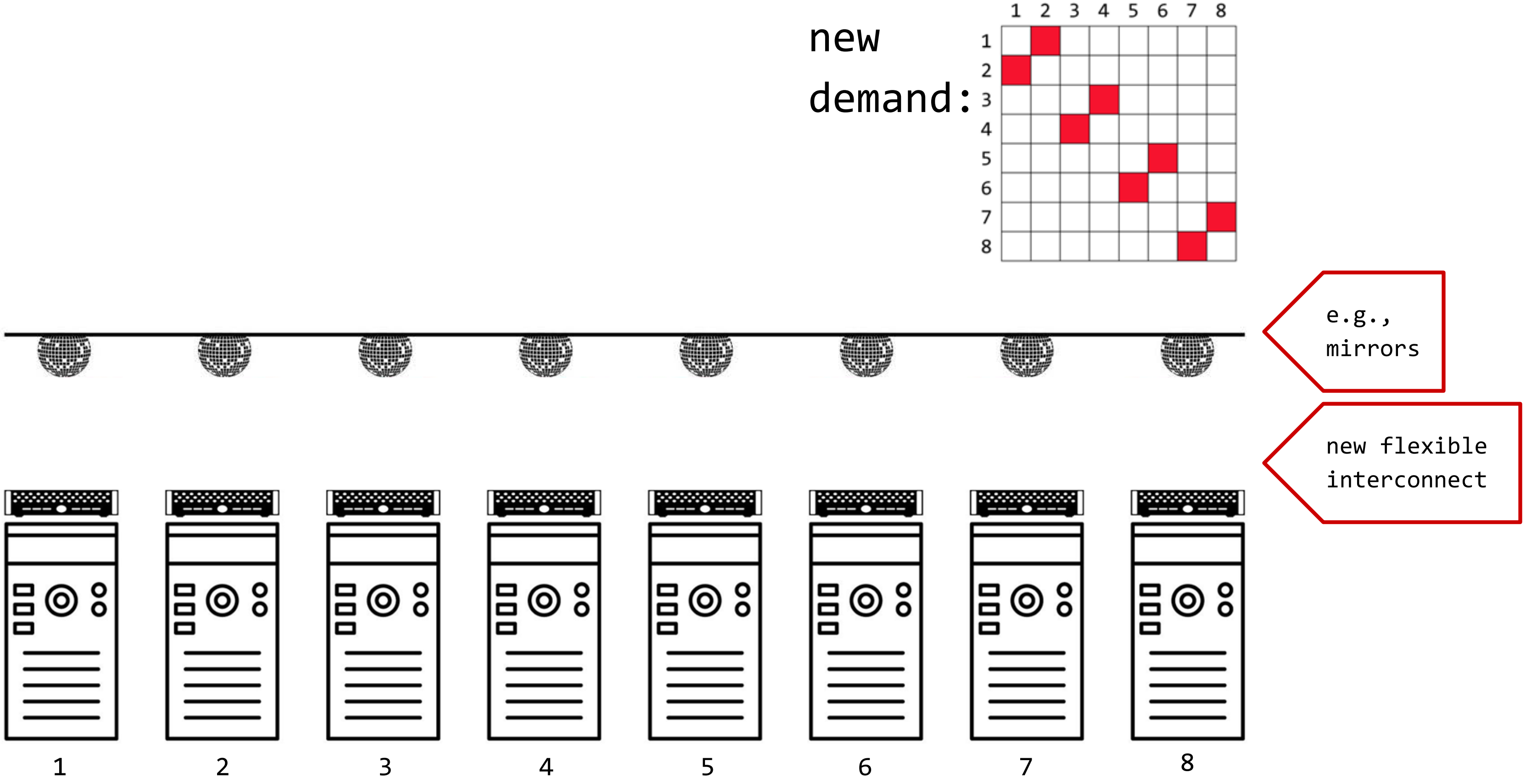

demand matrix:
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
e.g., mirrors
new flexible interconnect
1
2
3
4
5
6
7
8

Matches demand

demand matrix:

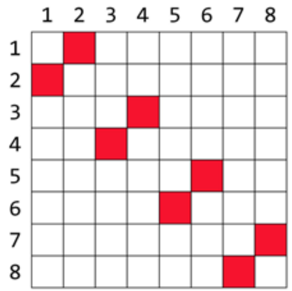
	1	2	3	4	5	6	7	8
1					■			
2						■		
3							■	
4								■
5	■							
6		■						
7			■					
8				■				

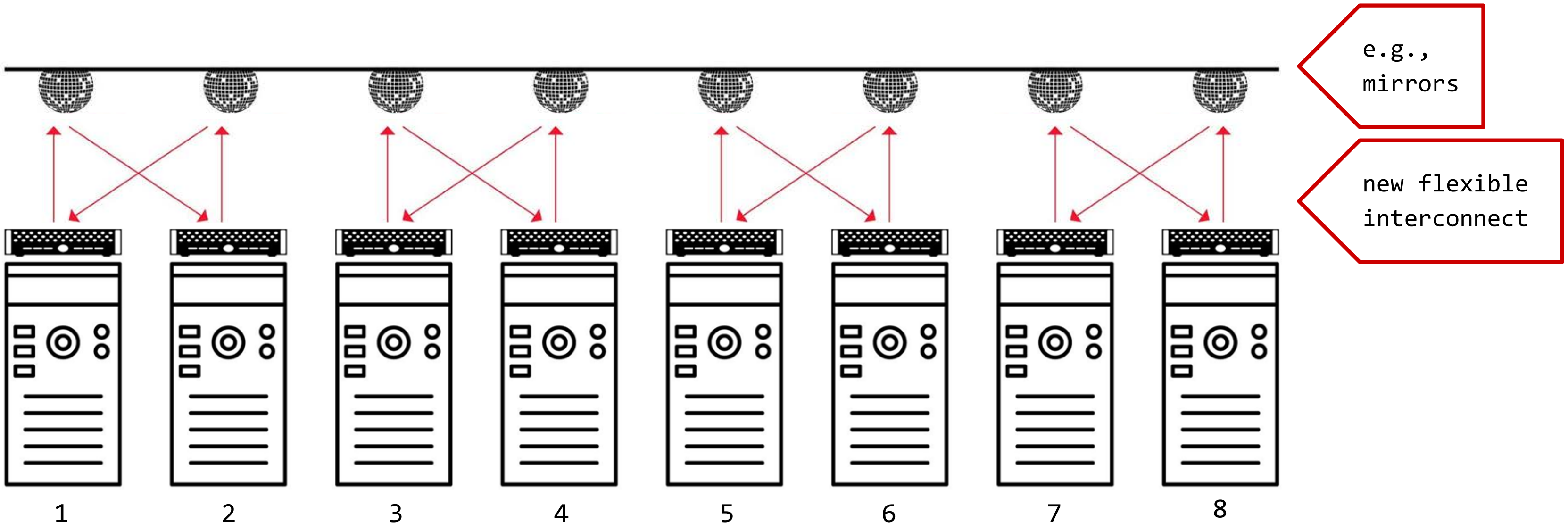



new
demand:
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
e.g.,
mirrors
new flexible
interconnect
1
2
3
4
5
6
7
8

Matches demand

new demand:





More Formally

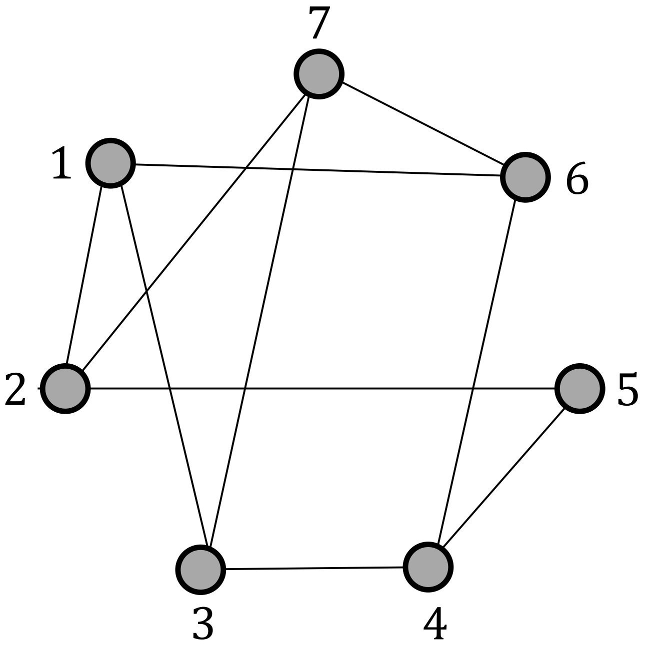
Input: Workload

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

$\mathcal{D}$

Output: Constant-Degree DAN



N

Input: Workload

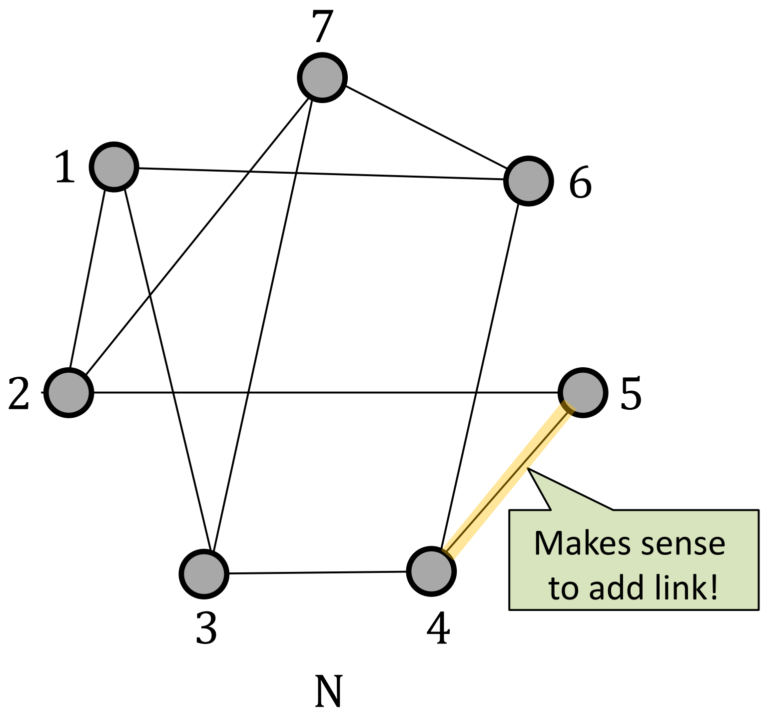
Destinations

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	$\frac{2}{65}$	[illegible]	[illegible]	[illegible]
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.

$\mathcal{D}$

Output: Constant-Degree DAN



Input: Workload

Output: Constant-Degree DAN

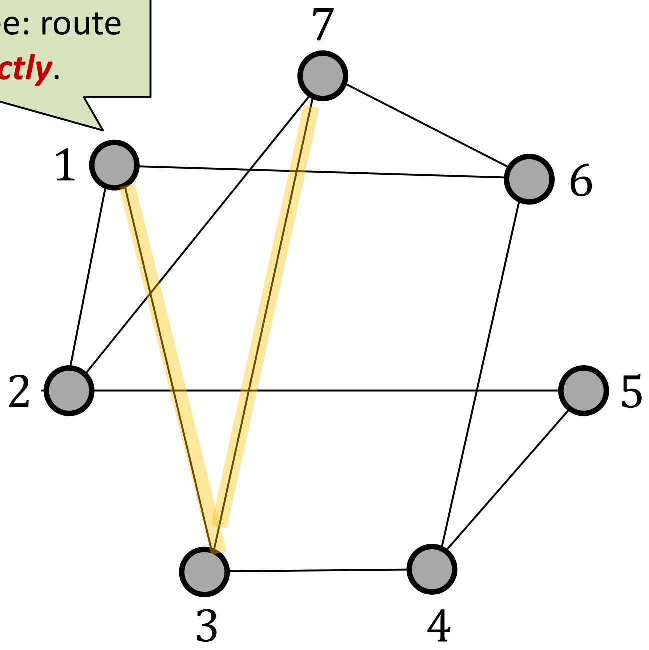
1 communicates to many.

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

$\mathcal{D}$

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



N

Input: Workload

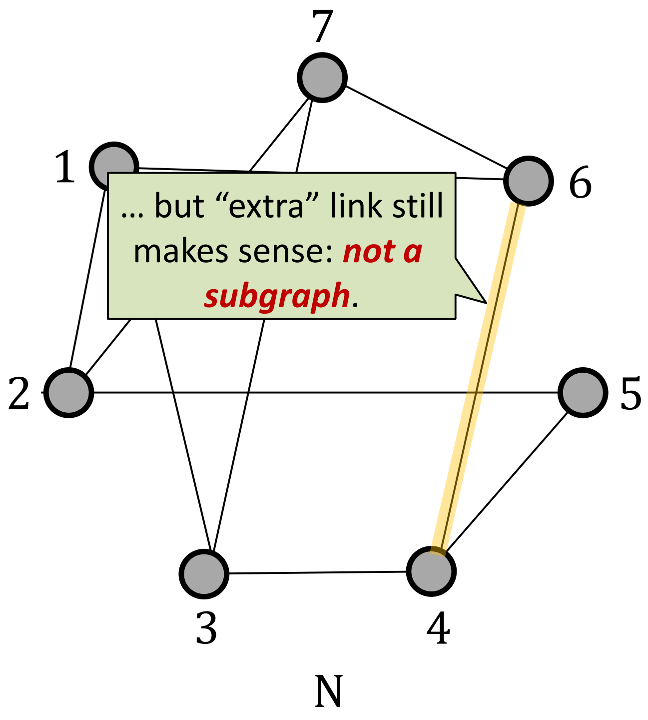
Destinations

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

$\mathcal{D}$

4 and 6 don't communicate…

Output: Constant-Degree DAN



Objective: Expected Route Length

DAN N of degree **Δ**

path length on *N*

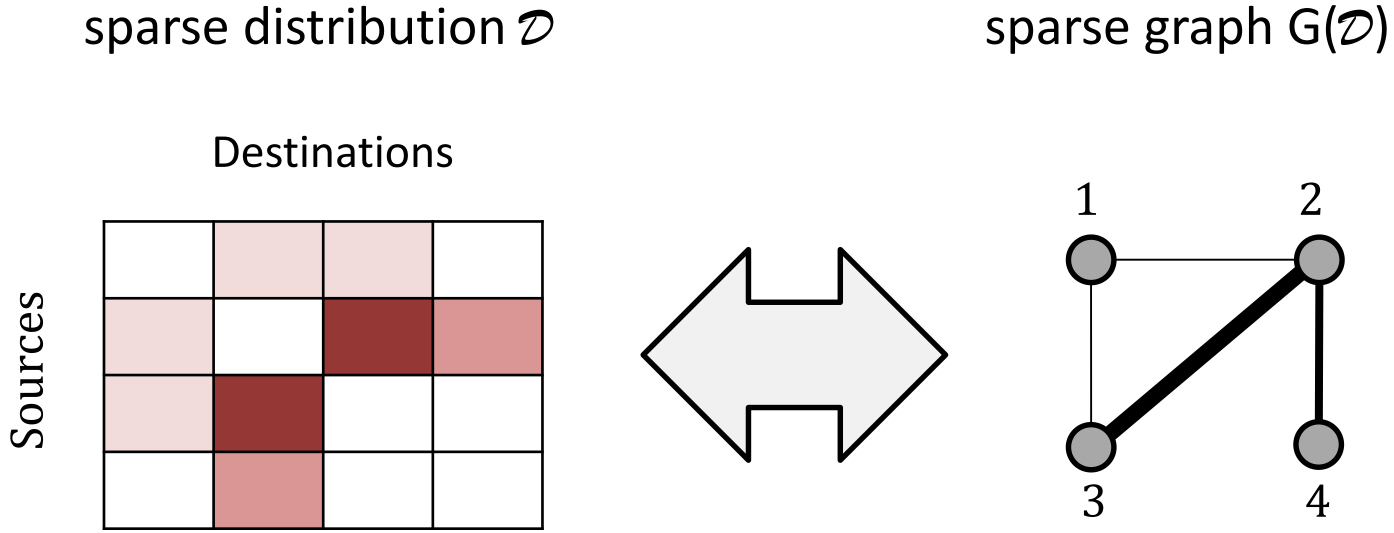
$$\mathrm{ERL}(\mathcal{D},\mathrm{N}) = \sum_{(u,v)\in\mathcal{D}} p(u,v)\cdot d_N(u,v)$$

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

frequency

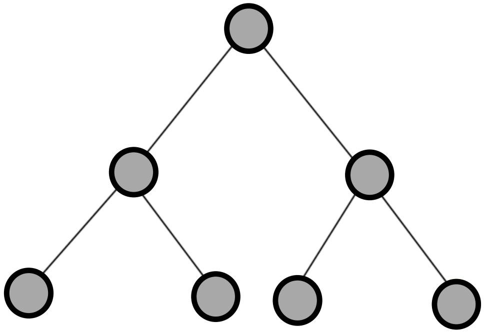
Remark

- Can represent demand matrix as a **demand graph**



Some Examples

- DANs of $\Delta = 3$:
  - E.g., complete binary **tree**
  - $d_N(u,v) \leq 2 \log n$
  - Can we do ***better*** than *log n*?

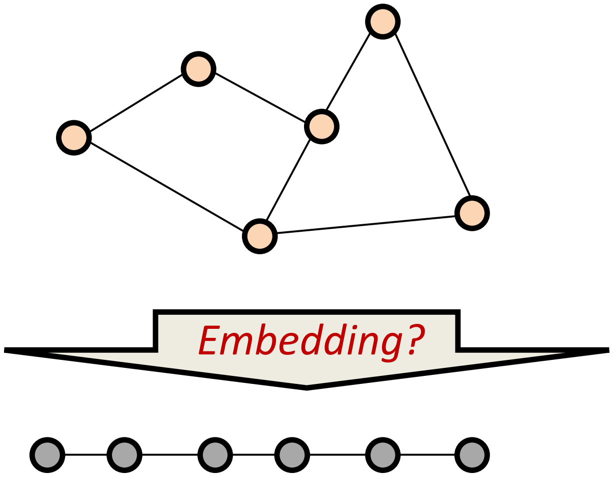


- DANs of $\Delta = 2$:
  - E.g., set of **lines** and **cycles**

Remark: Hardness Proof

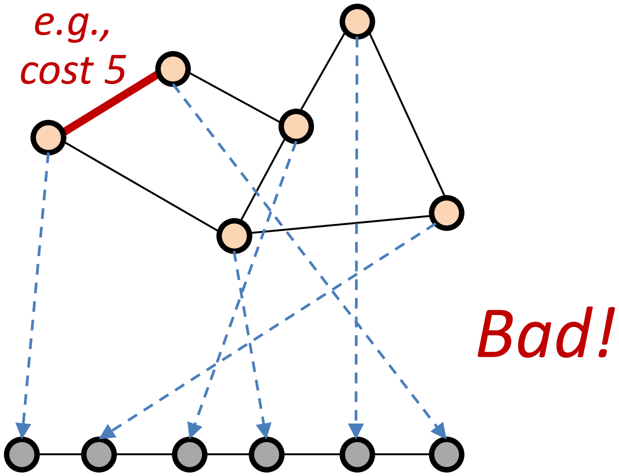
DAN design can be NP-hard

- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



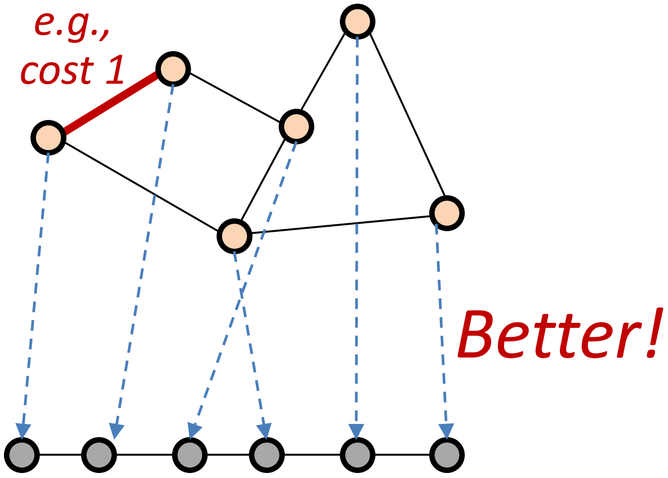
DAN design can be NP-hard

- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A "Virtual Network Embedding Problem", VNEP
  - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

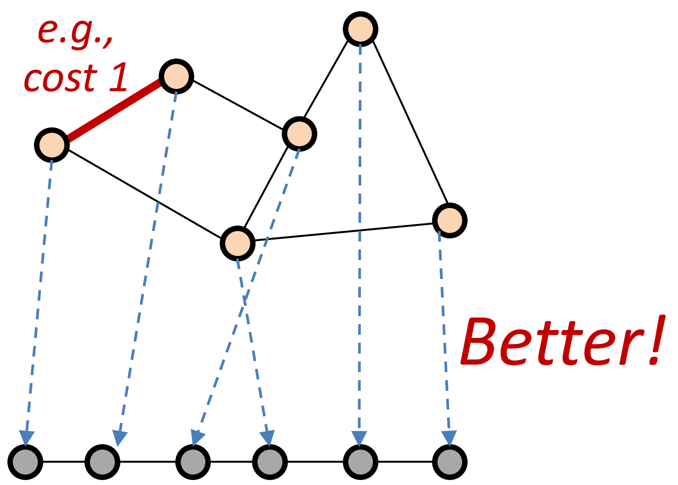
- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

- **Example Δ = 2:** A Mini… …inear Arrangement (**MI…** …blem
  - A "Virtual N… …bedding Problem", VNEP
  - ***Minin…*** …engths of virtual edges

NP-hard, and so is DAN design.



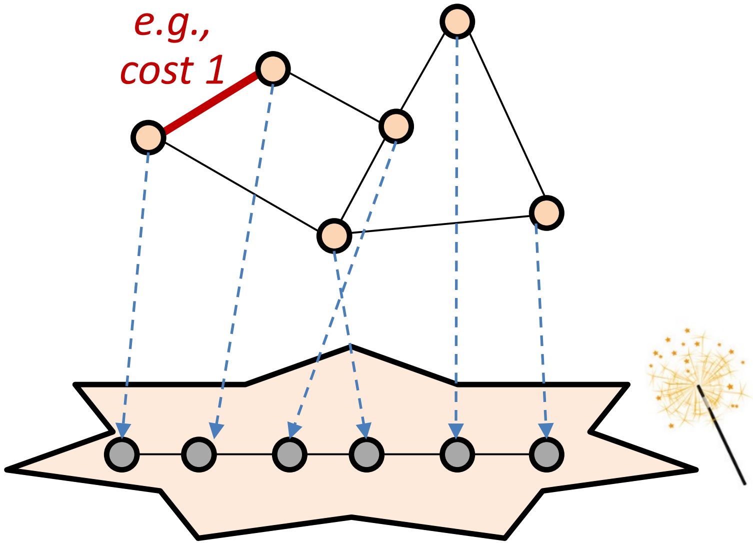
DAN design can be NP-hard

- **Example Δ = 2:** A Mini... inear Arrangement (MI... blem
  - A “Virtual N... bedding Problem”, VNEP
  - *Minim...* lengths of virtual edges

NP-hard, and so is DAN design.

- But what about > 2? ***Embedding*** problem still hard, but we have an additional **degree of freedom**:

Do topological flexibilities make problem easier or harder?!



A new knob for optimization!

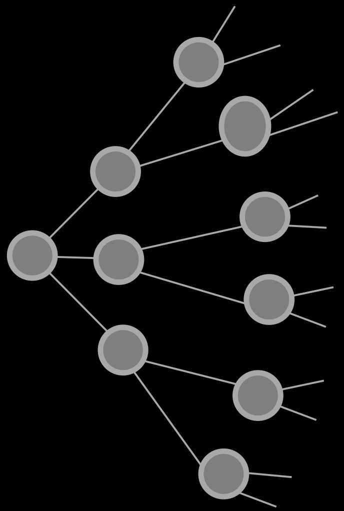
Rewinding the Clock: Degree-Diameter Tradeoff

Each network with n nodes and max degree $\Delta > 2$ must have a diameter of at least $\log(n)/\log(\Delta-1)-1$.

Example: constant Δ, $\log(n)$ diameter

Kudos to: Pedro Casas

Proof Idea



$1 \quad \Delta \quad \Delta(\Delta - 1) \quad \ldots$

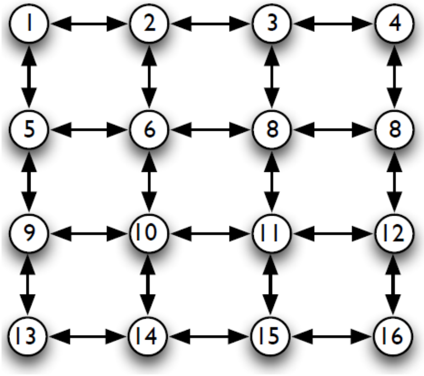
In k steps, reach at most $1 + \sum \Delta(\Delta - 1)^k$ nodes

Kudos to: Pedro Casas

Is there a better tradeoff in DANs?

Sometimes, DANs can be much better!

Example 1: low-degree demand

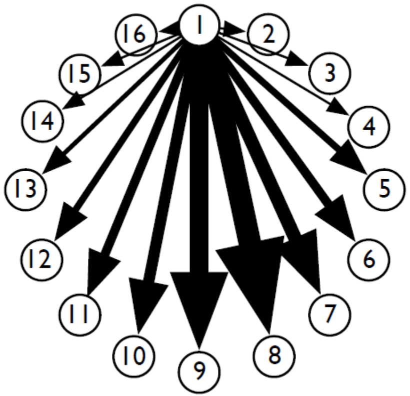


If **demand graph** is of degree Δ, it is trivial to design a **DAN** of degree Δ which achieves an ***expected route length of 1***.

Just take DAN = demand graph!

Sometimes, DANs can be much better!

Example 2: skewed demand

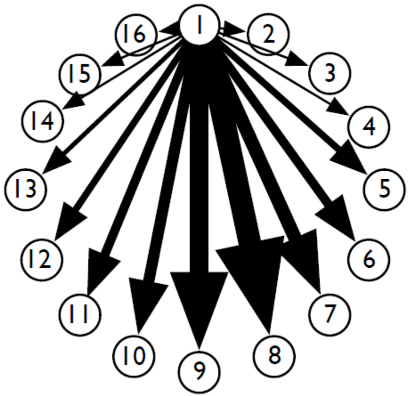


If **demand** is highly skewed, it is also possible to achieve an ***expected route length of O(1)*** in a constant-degree DAN.

?

Sometimes, DANs can be much better!

Example 2: skewed demand



If **demand** is highly skewed, it is also possible to achieve an ***expected route length of O(1)*** in a constant-degree DAN.

E.g., arrange neighbors of node 1 in a **Huffman** tree!

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. Chen Avin and Stefan Schmid. ACM **SIGCOMM CCR**, October 2018

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**!

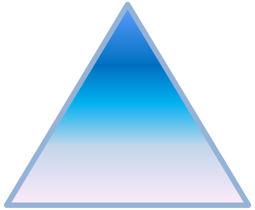
Intuition: Entropy Lower Bound

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 3***



- How good can this tree be? Cannot do better than Δ-ary **Huffman tree** for its destinations
- **Entropy** lower bound on ERL known for binary trees, e.g. ***Mehlhorn*** 1975

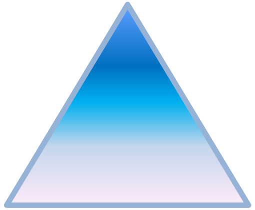
Lower Bound Idea: Leverage Coding or Datastructure

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

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

- DAN just for a ***single (source) node 3***



- How good can this tree be? Cannot do better than Δ-ary **Huffman tree** for its destinations
- **Entropy** lower bound on ERL known for binary trees, e.g. ***Mehlhorn*** 1975

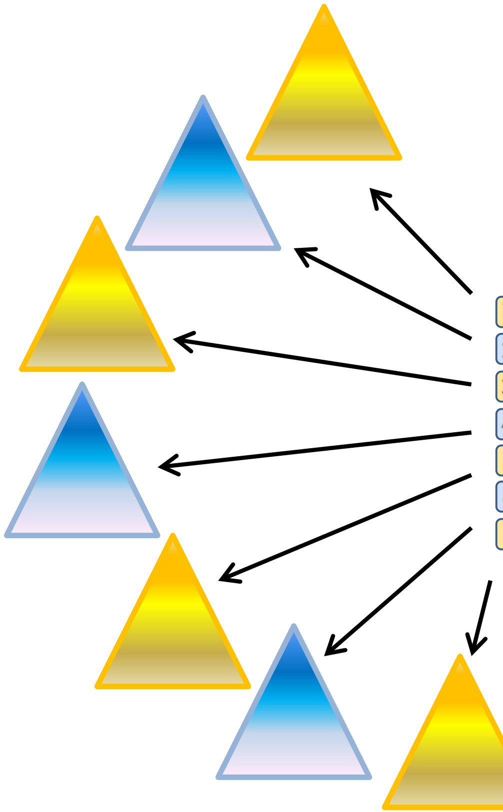
So: Entropy of the *Entire* Demand

sources destinations

- **Proof idea** (EPL=$\Omega(H_\Delta(Y|X))$):

entropy degree

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

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

	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

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

$\mathcal{D}$

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

Do this in **both dimensions**:

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

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

	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

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

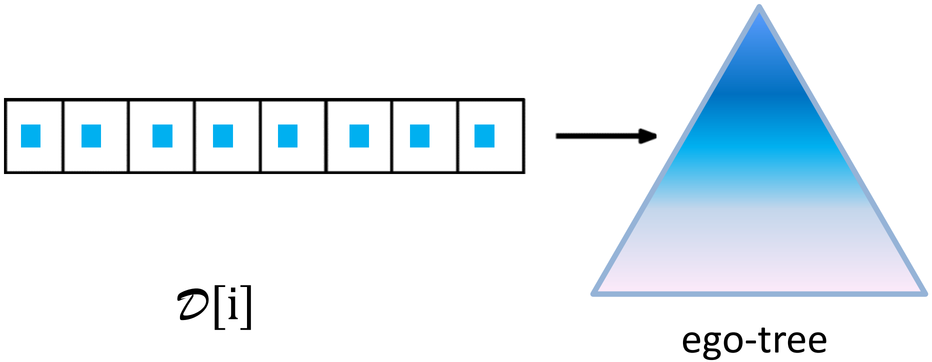
$\mathcal{D}$

Demand-Aware Network Designs of Bounded Degree. Chen Avin, Kaushik Mondal, and Stefan Schmid. **DISC**, 2017.

Achieving Entropy Limit: Algorithms

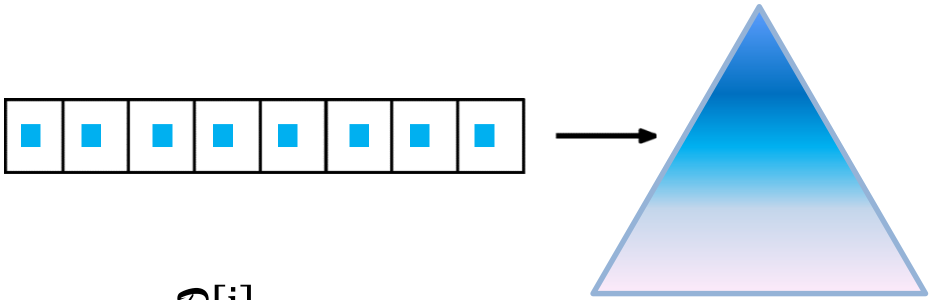
Ego-Trees Revisited

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



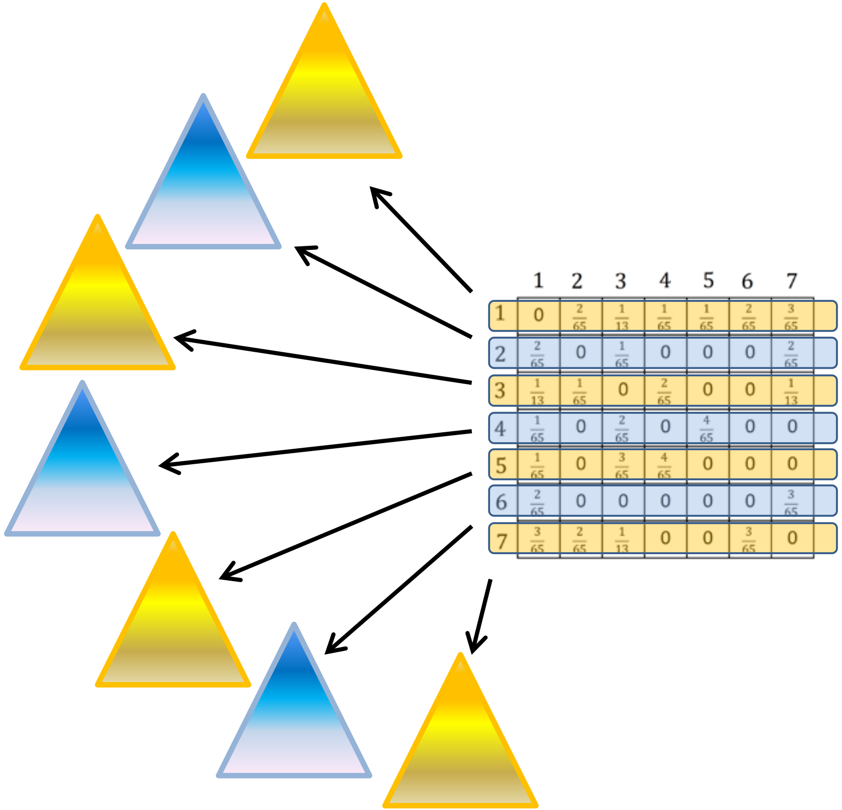
Ego-Trees Revisited

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



$\mathcal{D}[i]$

ego-tree

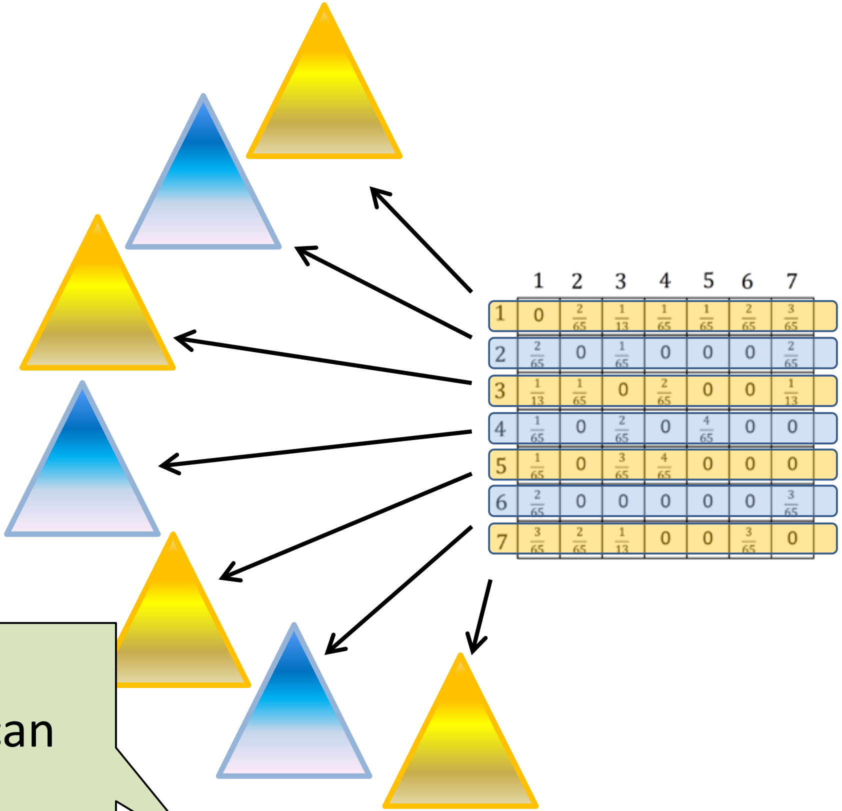


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$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

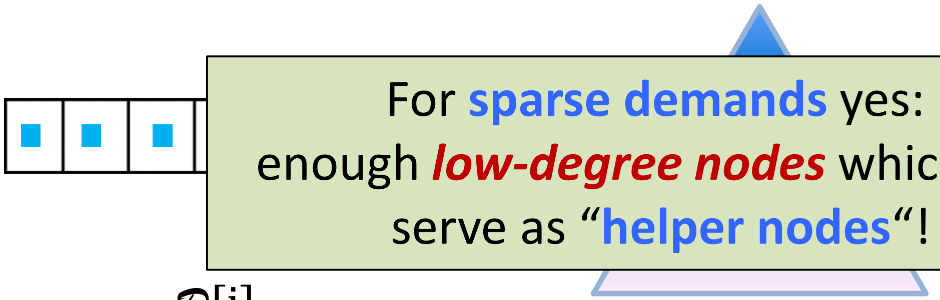
Can we merge the trees ***without distortion*** and ***keep degree low***?

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	$\frac{1}{65}$	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



$\mathcal{D}$[i]

ego-tree

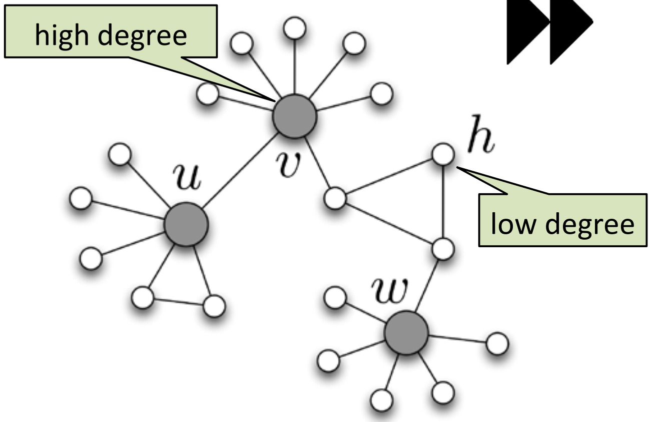
For **sparse demands** yes:
enough ***low-degree nodes*** which can
serve as “**helper nodes**“!

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

From Trees to Networks

Idea: Degree Reduction

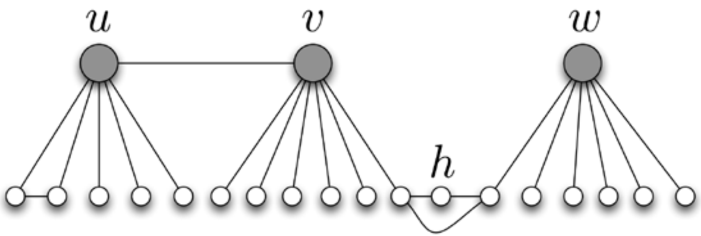
① **Demand graph**



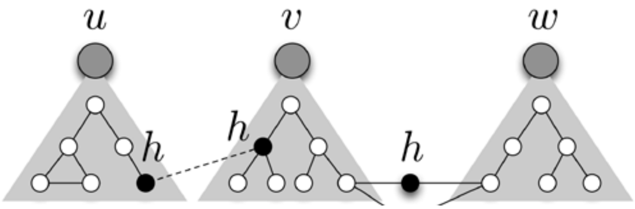
Taking union of ego-trees results in ***high degree***:
u and *v* will appear in many ego-trees

Demand-Aware Network Designs of Bounded Degree. Chen Avin, Kaushik Mondal, and Stefan Schmid. **DISC**, 2017.

② **Hierarchical representation**

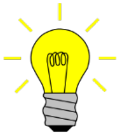


③ **Add low-degree nodes as helpers**



Node *h* ***helps edge (u, v)*** by participating in *ego-tree(u)* as a relay node toward *v* and *in ego-tree(v)* as a relay toward *u*

But: How to design DANs which also leverage *temporal structure*?



Inspiration from **self-adjusting datastructures** again!

Roadmap

- Entropy: A metric for demand-aware networks?
  - Empirical motivation
  - A lower bound
  - Algorithms achieving entropy bounds

- From static to dynamic demand-aware networks
  - A connection to self-adjusting datastructures

An Analogy

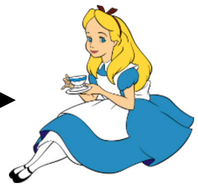
Static vs dynamic demand-aware networks!?

DANs vs **SANs**?

„Coming to the LKN retreat?“

An Analogy to Coding

00110101…



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

worst case network:
Full BW

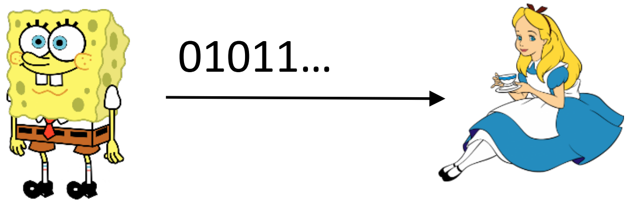
worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

„Coming to the LKN retreat?“

An Analogy to Coding



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

worst case network: Full BW
worst case coding: 00, 01, 10, 11

log diameter

log # bits / symbol

DAN!

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

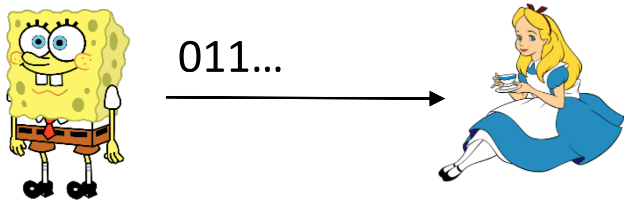
static Demand-Aware Nets
static Huffman: 1, 01, 001, 000

entropy?

entropy / symbol

„Coming to the LKN retreat?“

An Analogy to Coding



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***

entropy?

entropy / symbol

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

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

dynamic
Demand-Aware Nets

dynamic
Huffman codes

„Coming to the LKN retreat?“

An Analogy to Coding

011…

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***

static Demand-Aware Nets
static Huffman: 1, 01, 001, 000

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

dynamic Demand-Aware Nets
dynamic Huffman codes

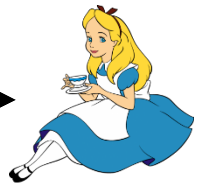
Can exploit ***spatial locality***!

Additionally exploit ***temporal locality***!

An Analogy to Coding

„Coming to the LKN retreat?“

011…



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 ***know***

if demand ***unknown*** but

„Cheating“: need to know demand!

Need online algorithms!

Can exploit
spati

Aware Nets

static Huffman:
1, 01, 001, 000

dynamic
Demand

Huf codes

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**

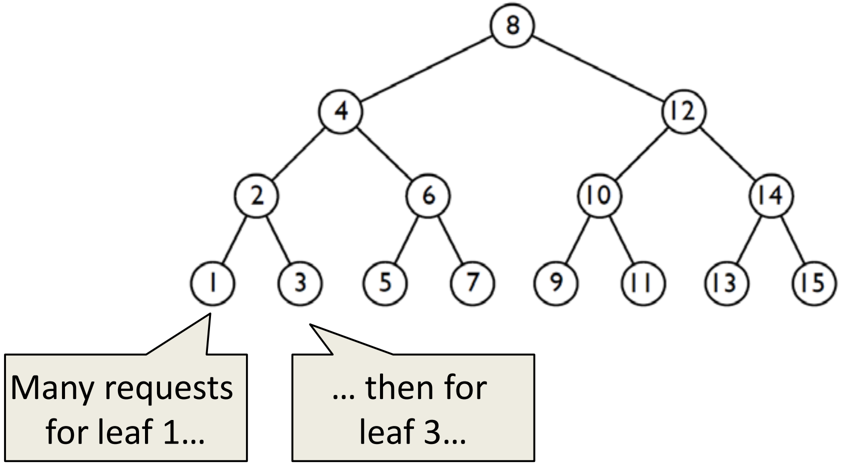
- Example **demand**:

  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*!



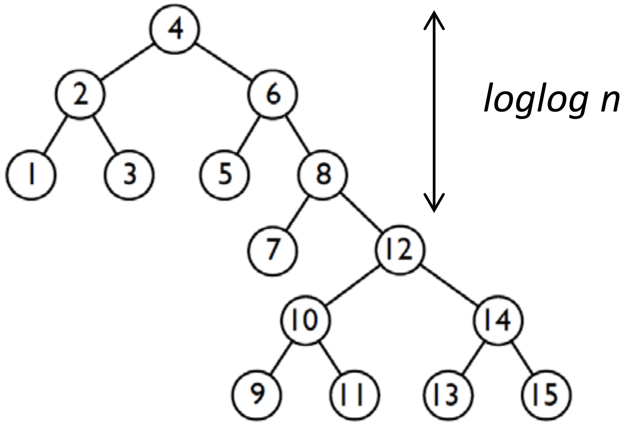
… Demand-Aware …

- **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)***

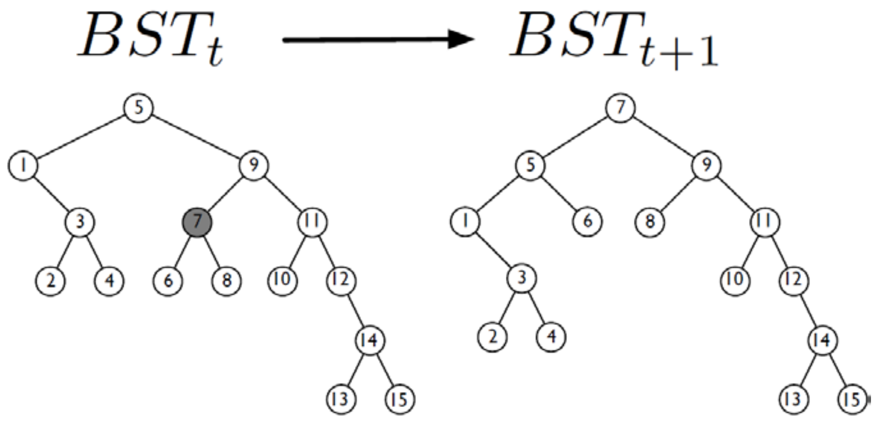


Amortized cost corresponds to ***empirical entropy of demand***!

… Self-Adjusting!

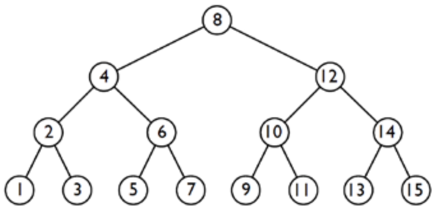
- **Demand-aware reconfigurable** BSTs can additionally take advantage of ***temporal locality***

- 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

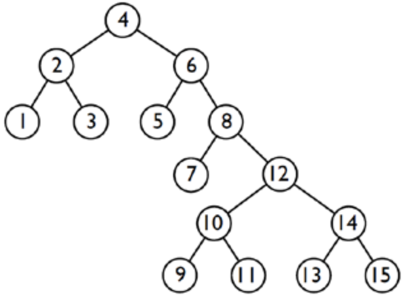
Oblivious



Lookup

O(log n)

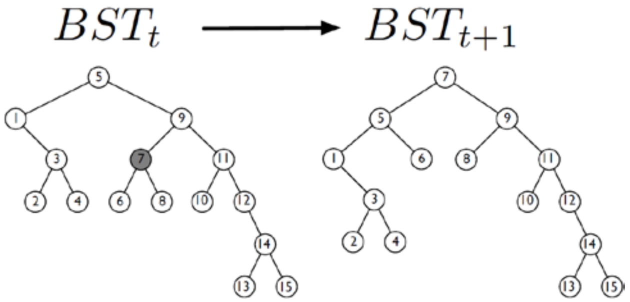
Demand-Aware



Exploit **spatial locality**:

empirical entropy O(loglog n)

Self-Adjusting

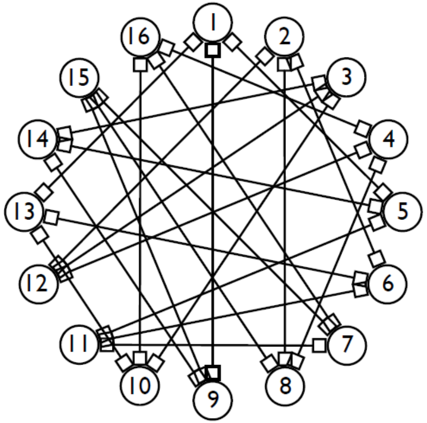


Exploit **temporal locality** as well:

O(1)

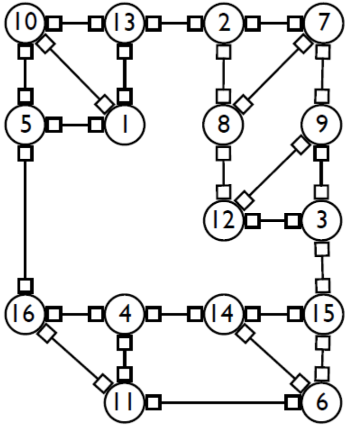
Analogously for Networks

Oblivious



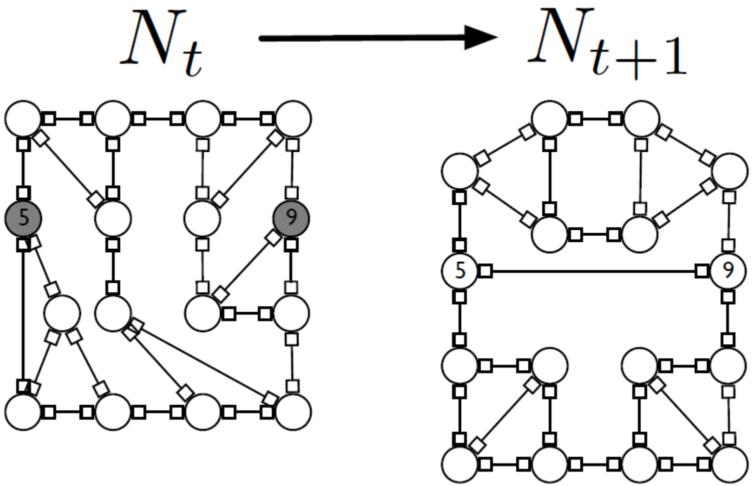
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**

SAN



Exploit **temporal locality** as well

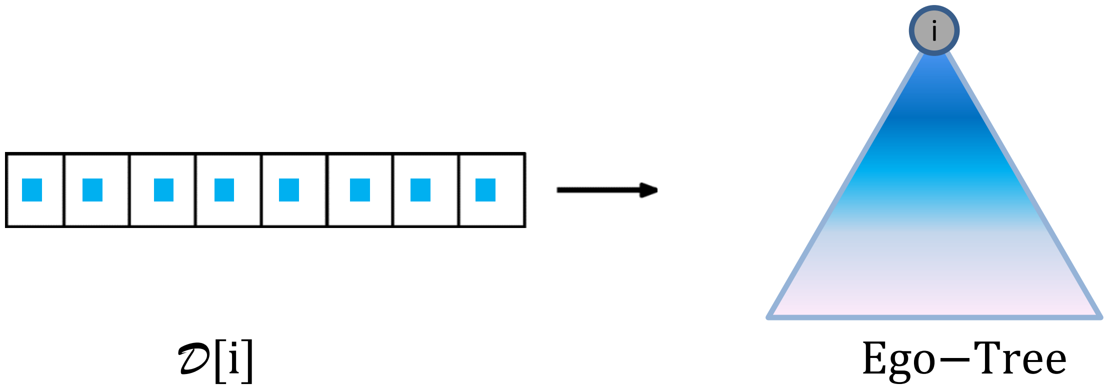
Avin, S.: Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **SIGCOMM CCR** 2018.

Algorithms for Self-Adjusting Networks

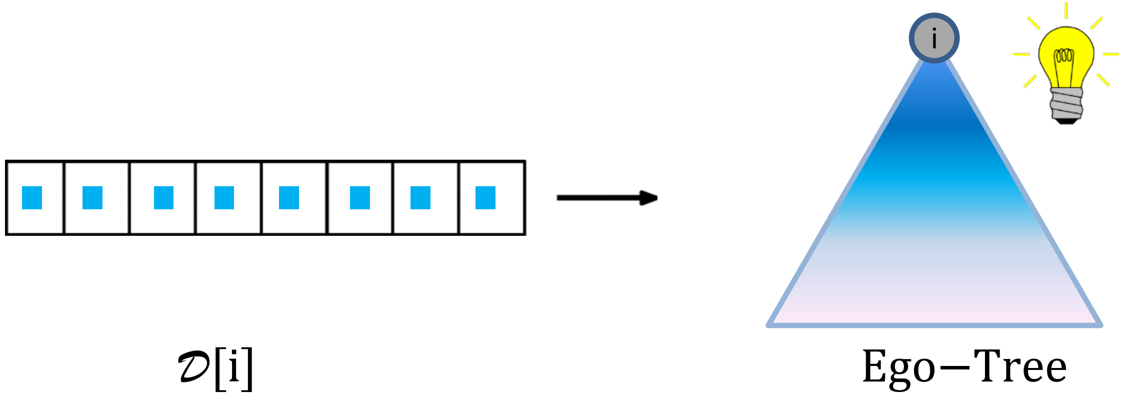

From trees to networks!

Ego-trees strike back!

Total Recall: Ego-Trees!



Total Recall: Ego-Trees!

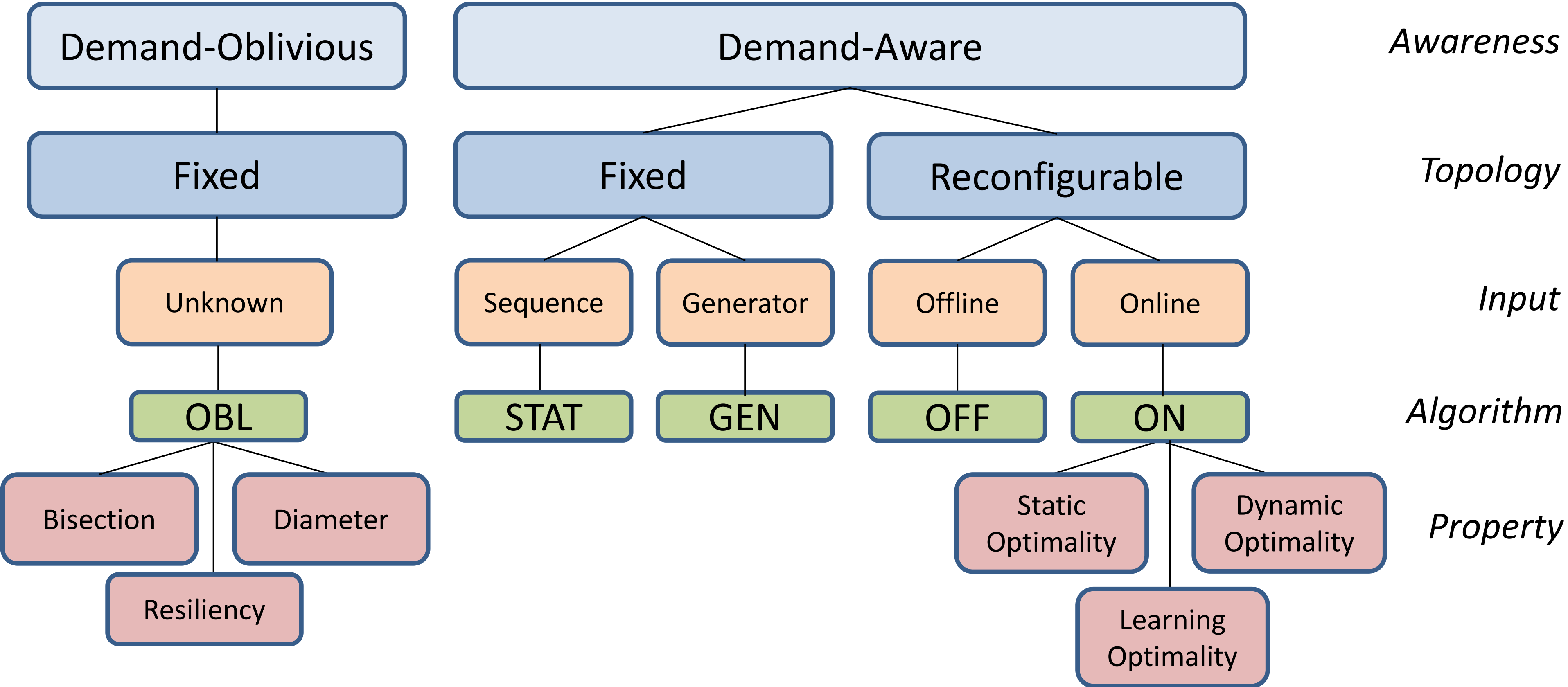


Idea: use our old approach but now let each node ***adjust its ego-tree***!

A Dynamic Ego-Tree:
Splay Tree

Uncharted Landscape!

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



Flexibilities and Algorithms: Opportunities and Challenges

Optimizing Individual Routers

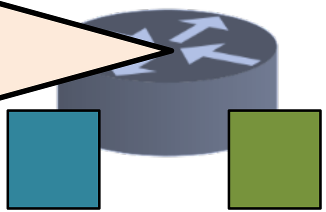
Online Aggregation of the Forwarding Information Base: Accounting for Locality and Churn. Marcin Bienkowski, Nadi Sarrar, Stefan Schmid, and Steve Uhlig. IEEE/ACM Transactions on Networking (**TON**), 2018.

Poor IP Routers

Forwarding Information Base (FIB)

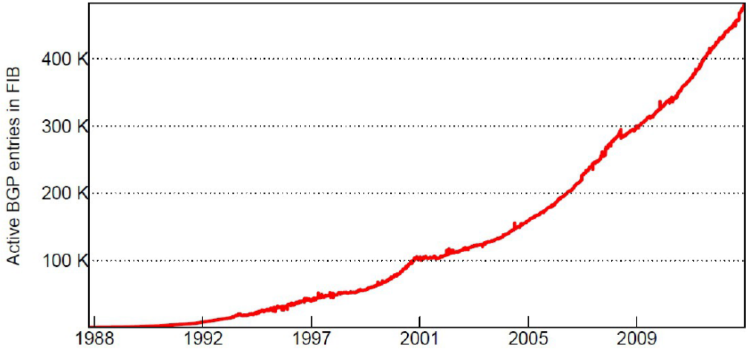
00* to	
01* to	
1* to	

most specific prefix **fast**?



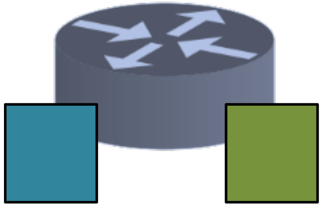
Ports

TCAM memory expensive and power-hungry…



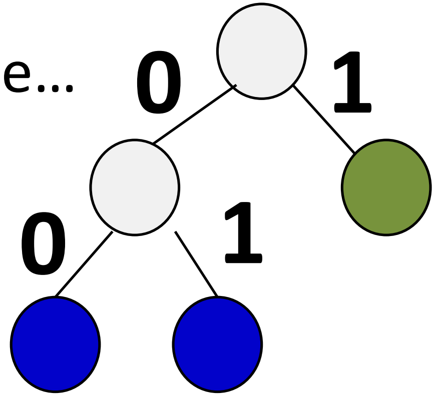
… and requirements grow quickly (e.g., virtualization). IPv6 does not help.

Idea: Represent as Trie

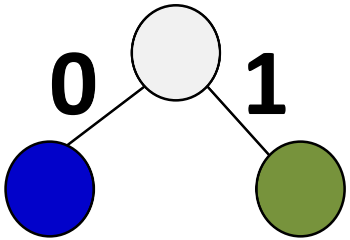


00* to
01* to
1* to

represent as trie…

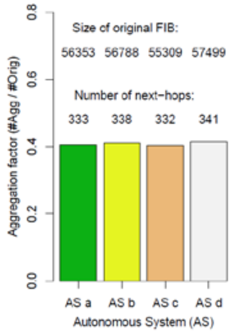


… and compress it!

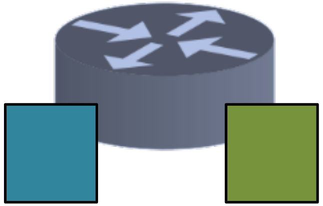


Good Potential:

Down to 40% (RouteView), depending on # ports.

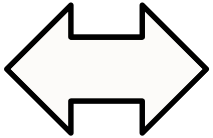


Idea: Represent as Trie

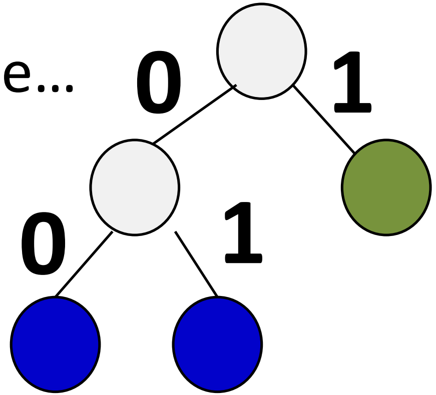


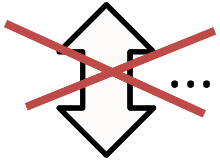
00* to	
01* to	
1* to	

represent as trie...



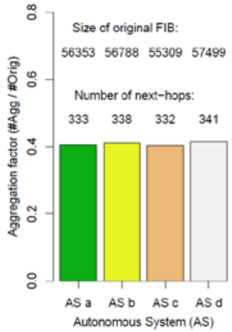
BGP update

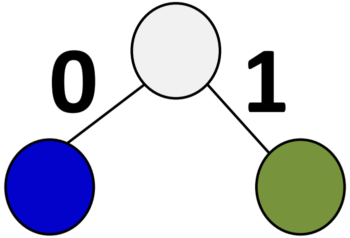




... and compress it!

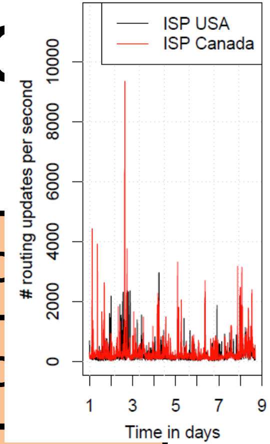
But may introduce
***update churn*!**



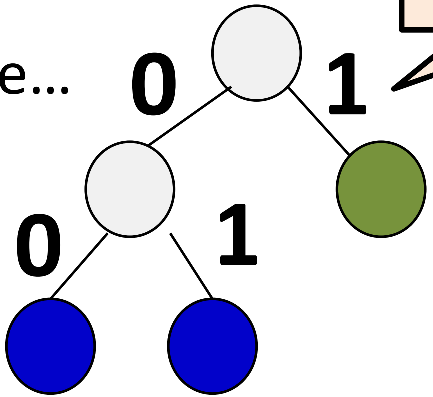


Id… epresent as Trie

00* to
01* to
1* to



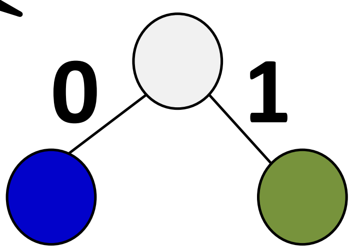
…present as trie…



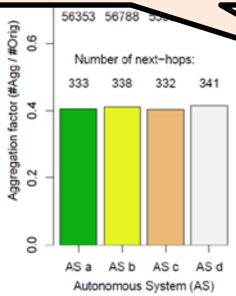
update cost 2:
remove + add

update cost 3:
remove + add
subtree

… and compress it!



But may introduce
***update churn*!**



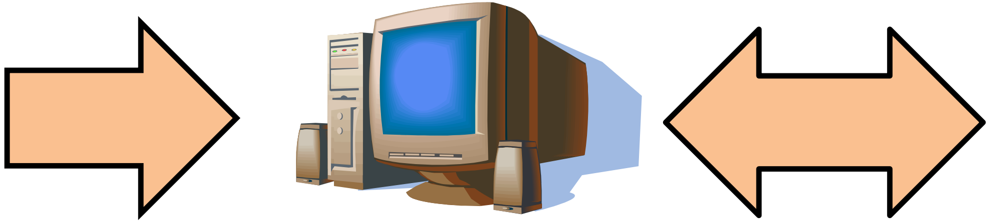
An Optimization Problem

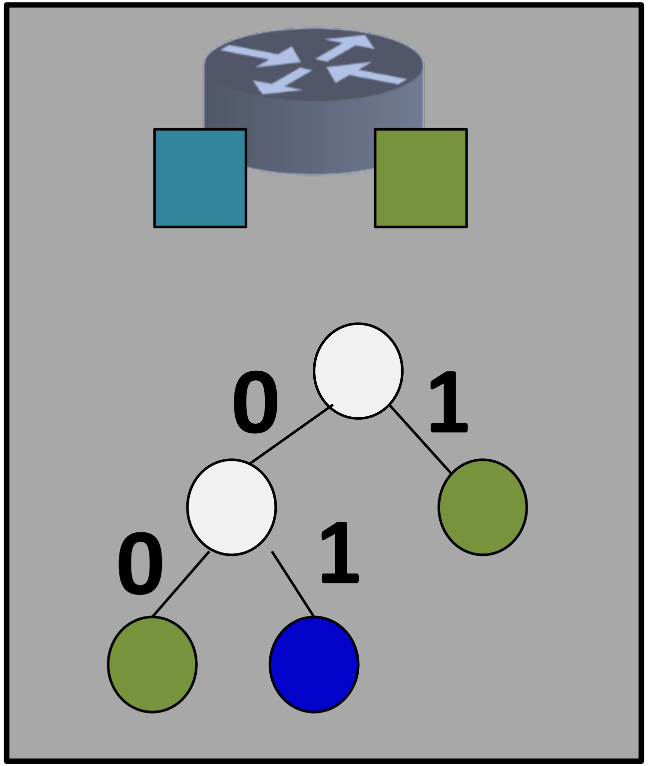
Route Processor or SDN Controller

FIB or SDN Switch

BGP Events

update!





An online problem:

1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size

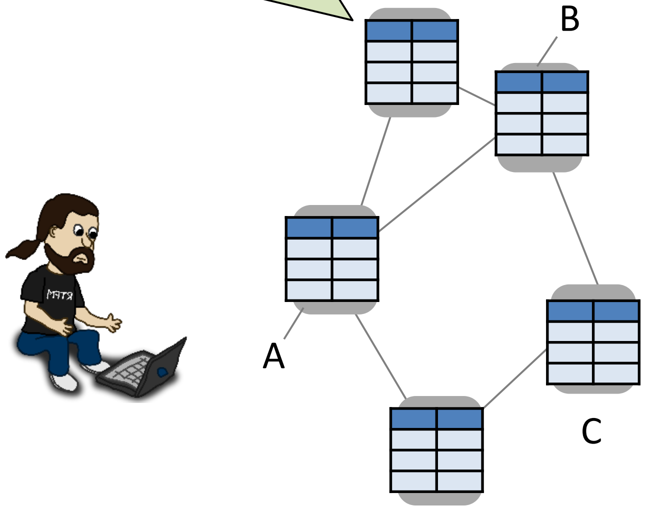
Optimization of Local Fast Failover

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. ACM **CoNEXT**, Heraklion/Crete, Greece, December 2018.

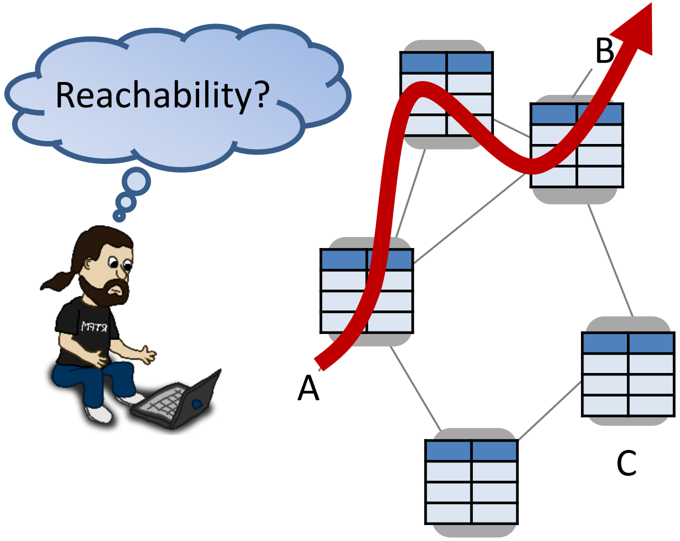
Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks
Stefan Schmid and Jiri Srba. IEEE **INFOCOM**, Honolulu, Hawaii, USA, April 2018.

Responsibilities of a Sysadmin

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



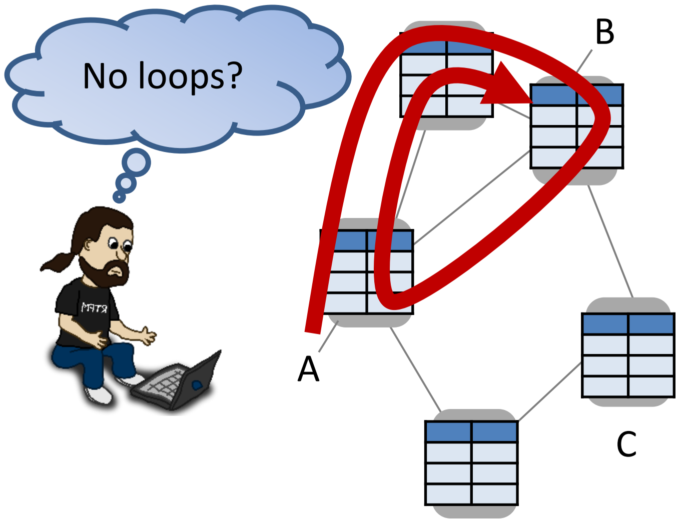
Responsibilities of a Sysadmin



Sysadmin responsible for:

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

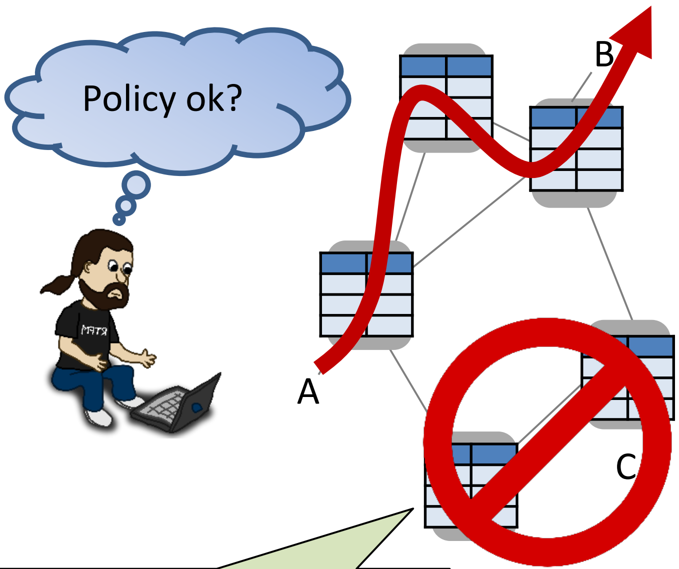
Responsibilities of a Sysadmin



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

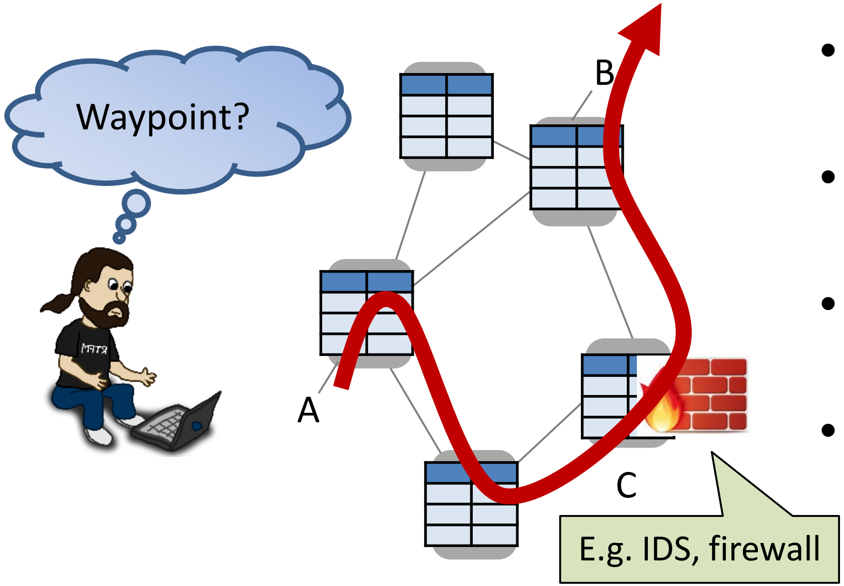


E.g. ***NORDUnet***: no traffic via Iceland (expensive!). Or no traffic through ***route reflectors***.

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?

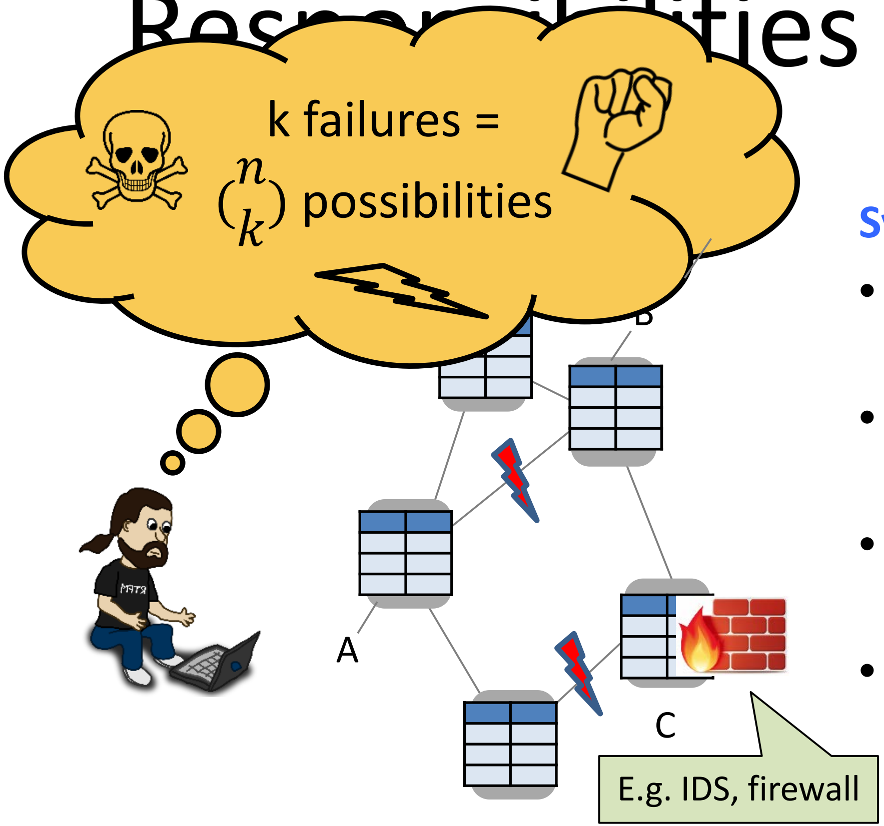
Responsibilities of a Sysadmin



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?

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?

… and everything even under multiple failures?!

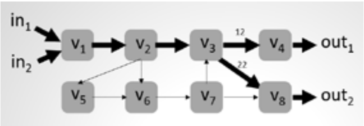
Can we automate such tests
or even self-repair?

Can we automate such tests or even self-repair?

Yes! Encouraging: sometimes even ***fast***:
What-if Analysis Tool for MPLS and SR

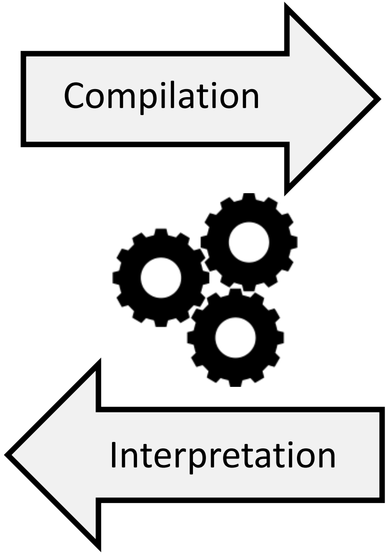
Leveraging Automata-Theoretic Approach

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\bot$	(v_1, v_2)	$push(10)$
	in_2	$\bot$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

MPLS **configurations**, Segment Routing etc.

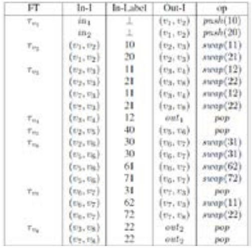


$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

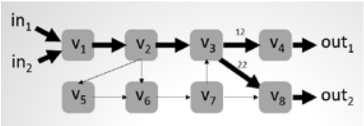
Pushdown Automaton and **Prefix Rewriting Systems** Theory

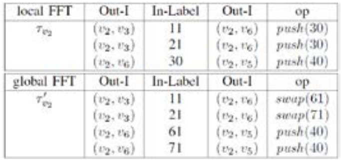
Leveraging Autom… …oach

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



FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop





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

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

Compilation

Interpretation

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

MPLS **configurations**, Segment Routing etc.

Pushdown Automaton and **Prefix Rewriting Systems** Theory

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

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

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

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

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

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

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

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

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

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

Time for Automata Theory (from Switzerland)!

- Classic result by **Büchi** 1964: the set of all reachable configurations of a pushdown automaton a is regular set
- Hence, we can operate only on Nondeterministic Finite Automata (NFAs) when reasoning about the pushdown automata
- The resulting **regular operations** are all polynomial time
  - Important result of **model checking**

Julius Richard Büchi

1924-1984

Swiss logician

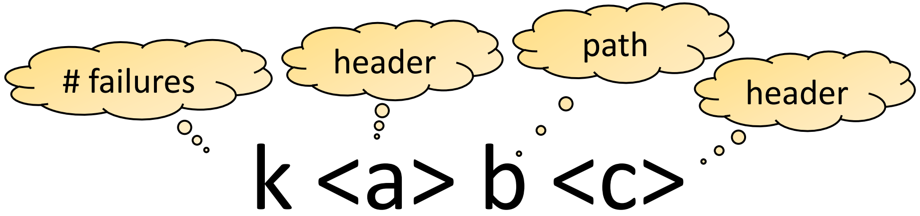
Tool and Query Language

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

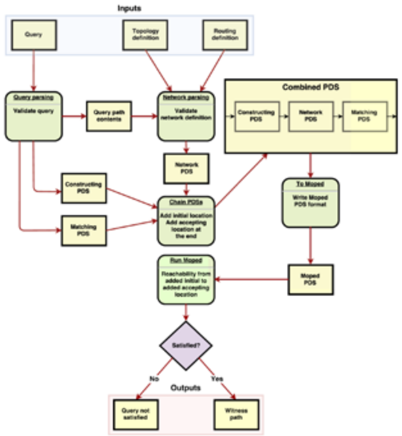
- In Python 3

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

- Using ***Moped*** tool



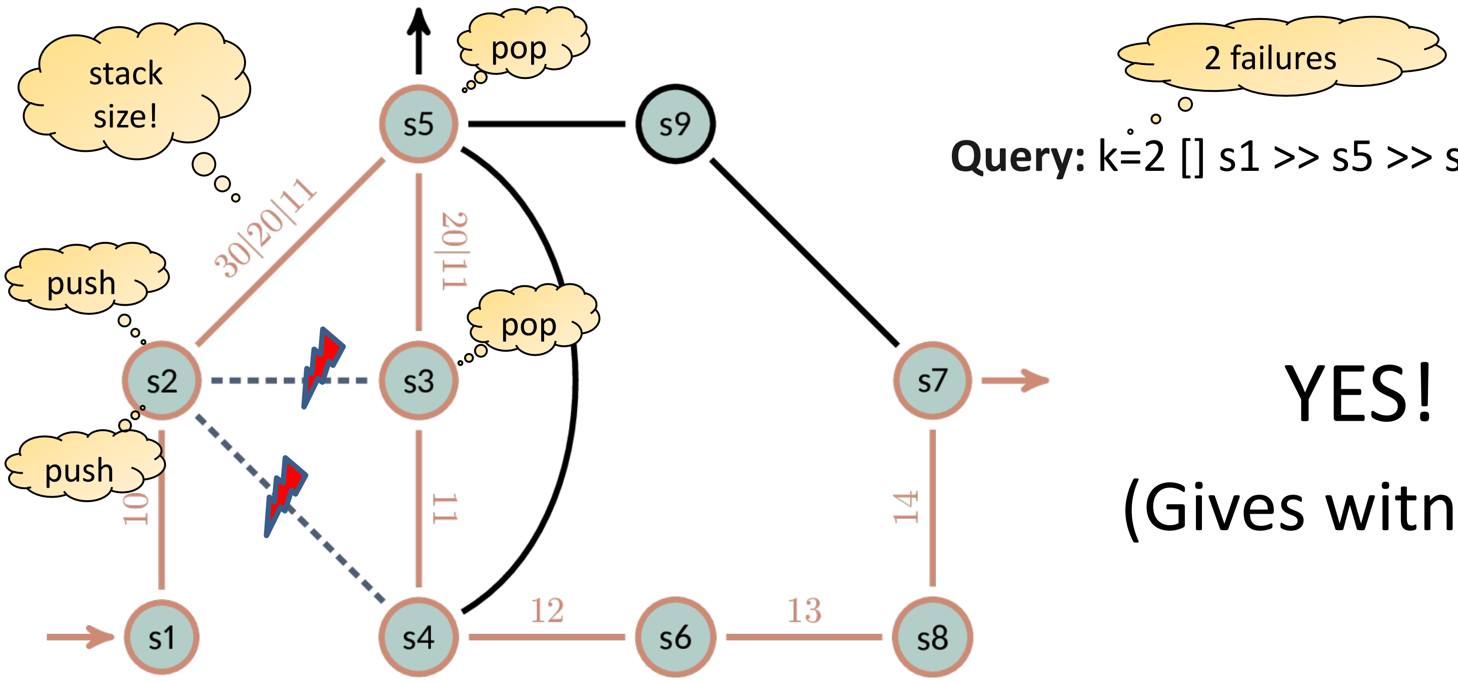
Regular query language



query processing flow

Example: Traversal Testing With 2 Failures

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



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

YES!

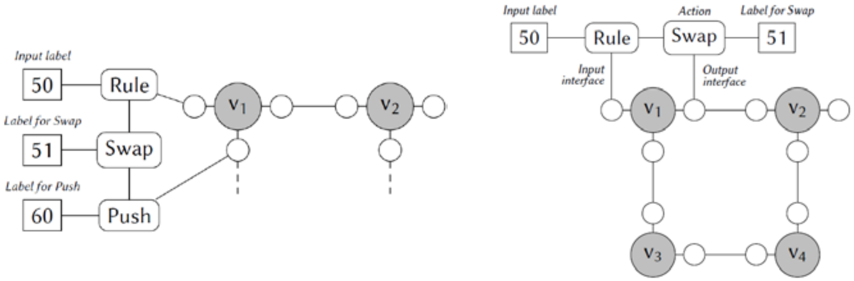
(Gives witness!)

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

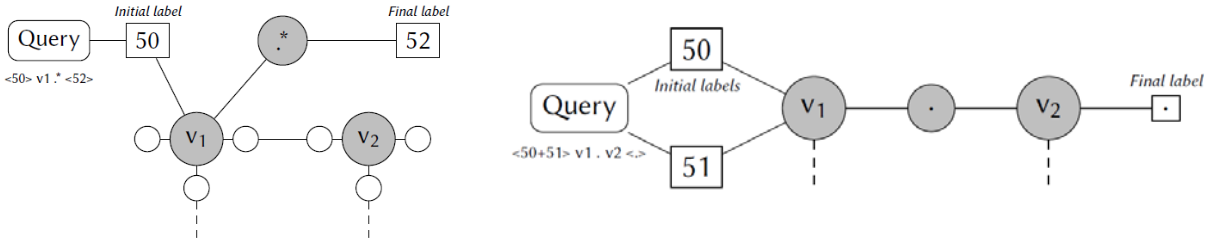
DeepMPLS: Fast Analysis of MPLS Configurations Using Deep Learning. Fabien Geyer and Stefan Schmid. **IFIP Networking**, Warsaw, Poland, May 2019.

Speed Up Further and Synthesize: Deep Learning (s. talk by Fabien Geyer)

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



Network topologies and MPLS rules



Network topologies and query

Challenges of Self-* Networks

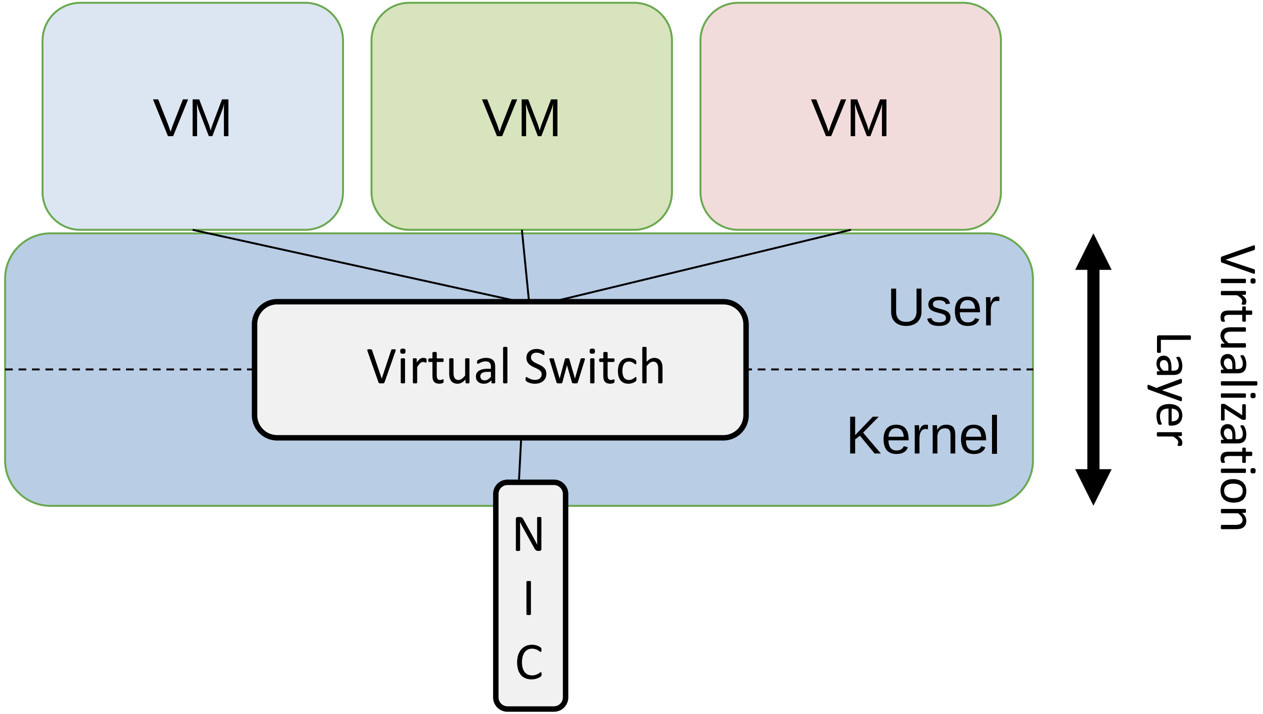
- Can a self-* network realize its **limits**?
- E.g., when quality of **input data** is not good enough?
- When to hand over to human? Or **fall back** to „safe/oblivious mode“?
- Can we learn from self-driving **cars**?

Security Challenges of More Flexible Networks

MTS: Bringing Multi-Tenancy to Virtual Switches. Kashyap Thimmaraju, Saad Hermak, Gabor Retvari, and Stefan Schmid. USENIX **ATC**, 2019.

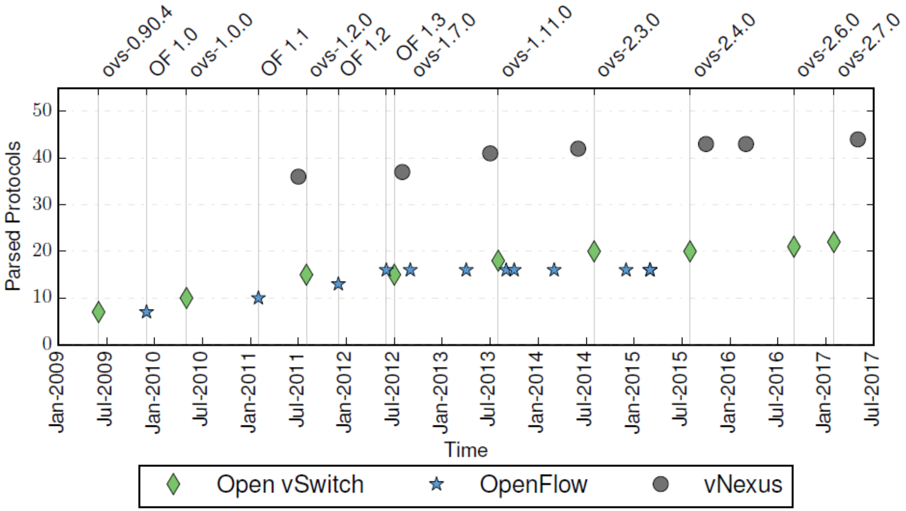
Taking Control of SDN-based Cloud Systems via the Data Plane. Kashyap Thimmaraju et al. ACM **SOSR**, USA, March 2018.

Security of vSwitch



Virtual switches reside in the **server's virtualization layer** (e.g., Xen's Dom0). Goal: provide connectivity and isolation.

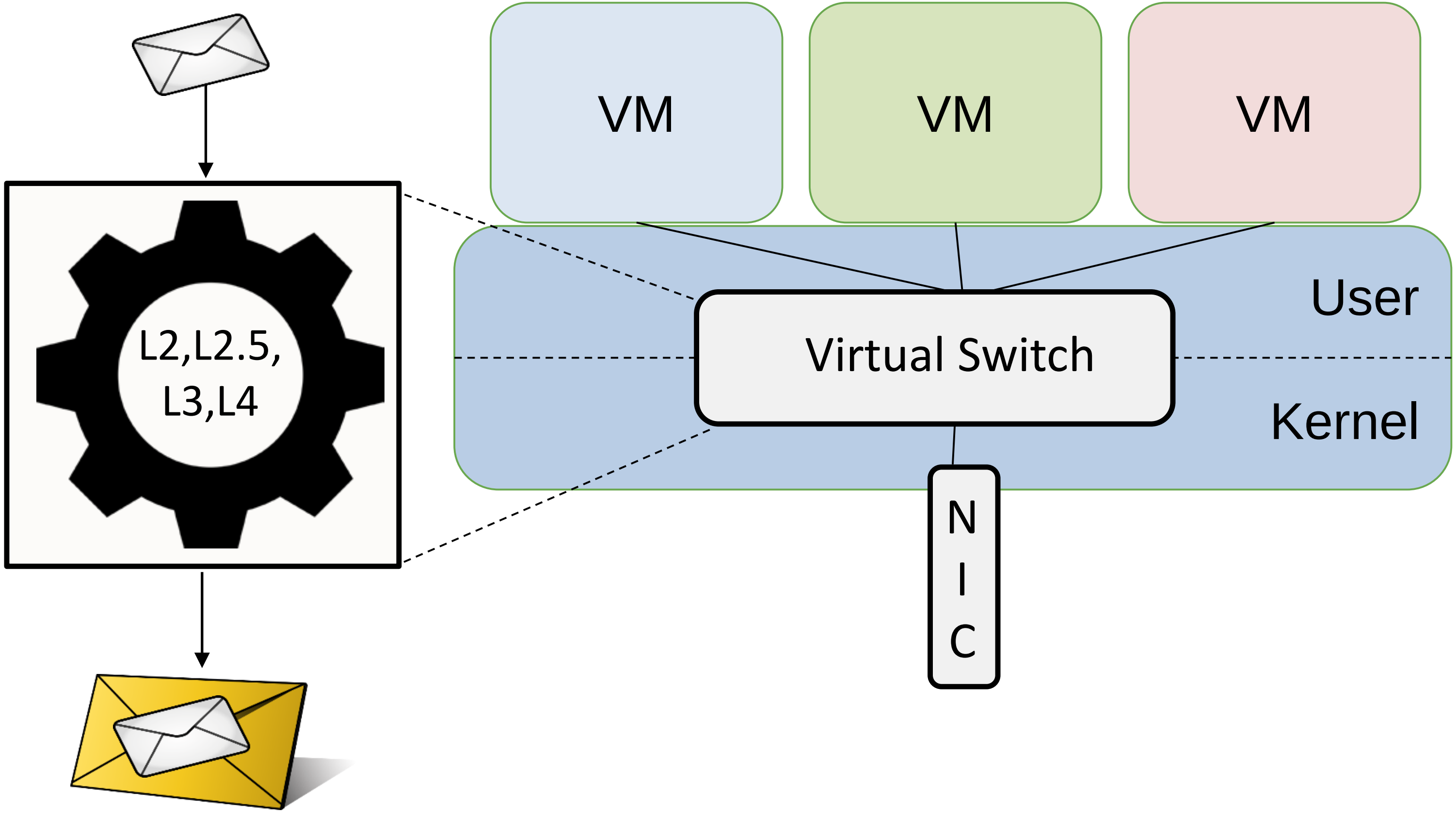
Security of vSwitch



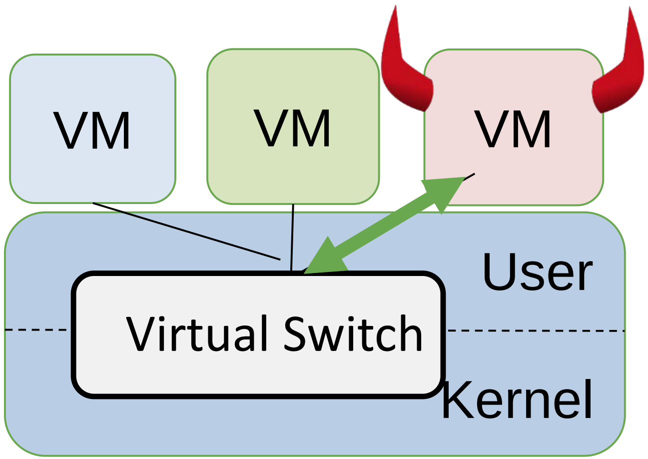
Number of parsed high-level protocols constantly increases…

Security of vSwitch

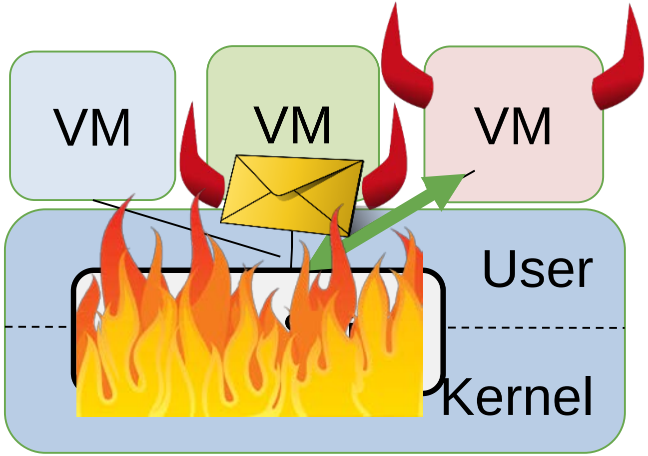
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



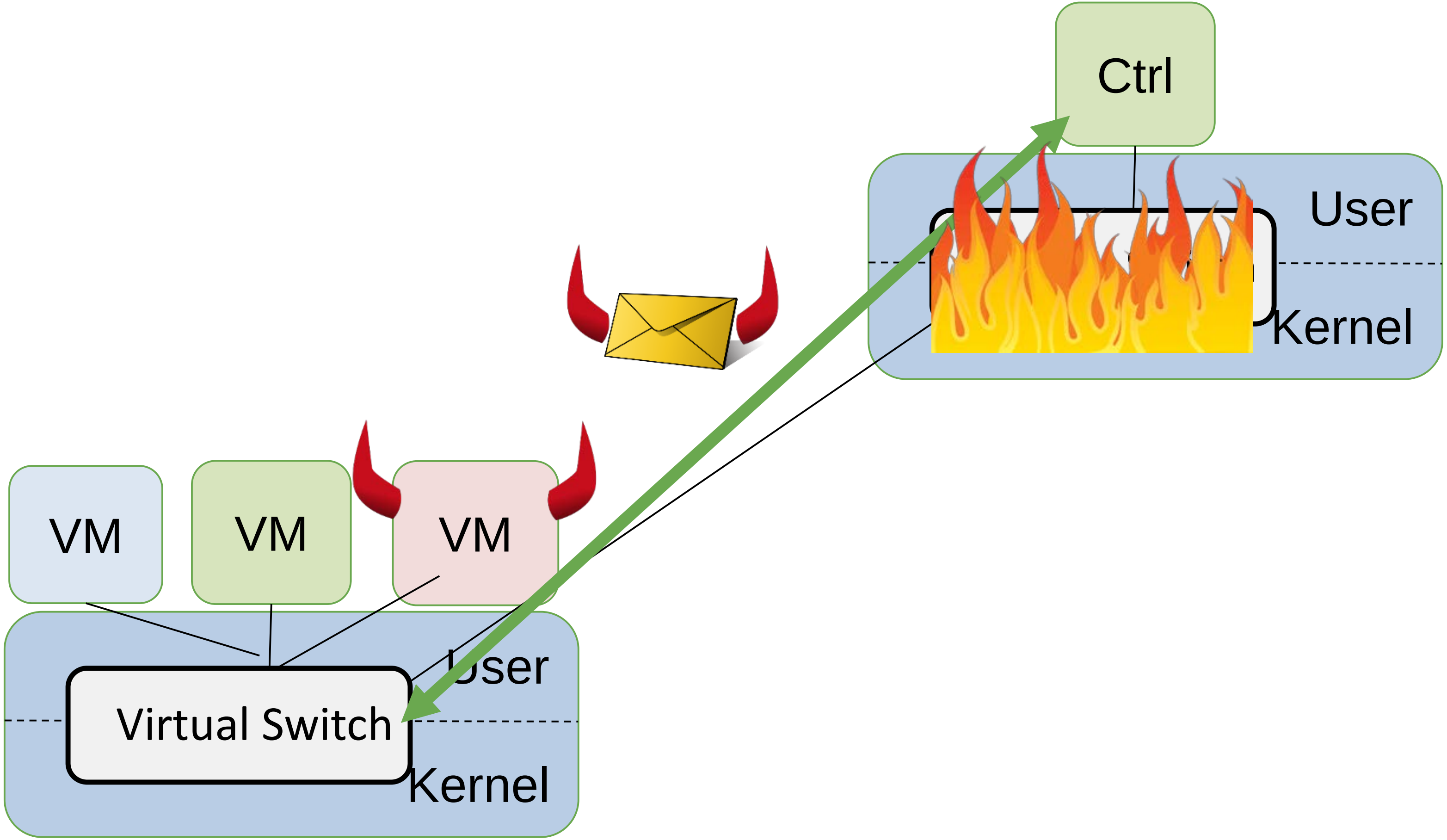
Security of vSwitch



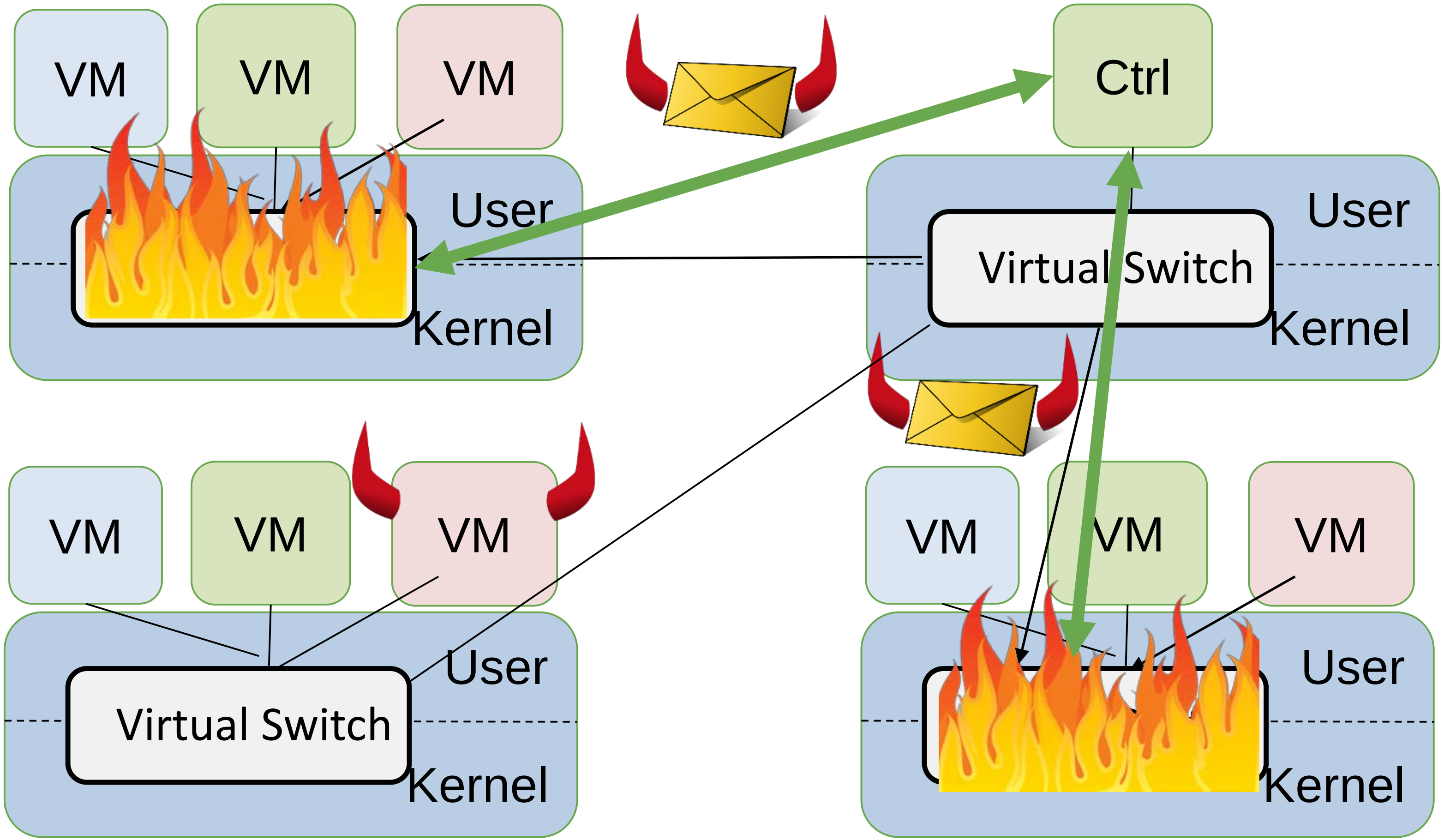
Security of vSwitch

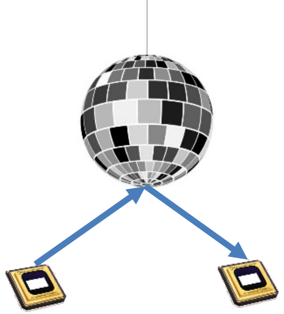


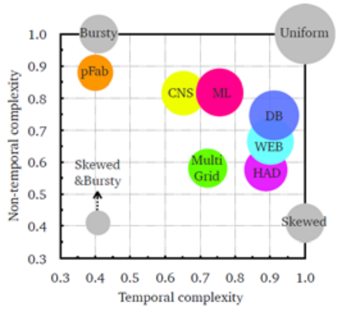
Security of vSwitch



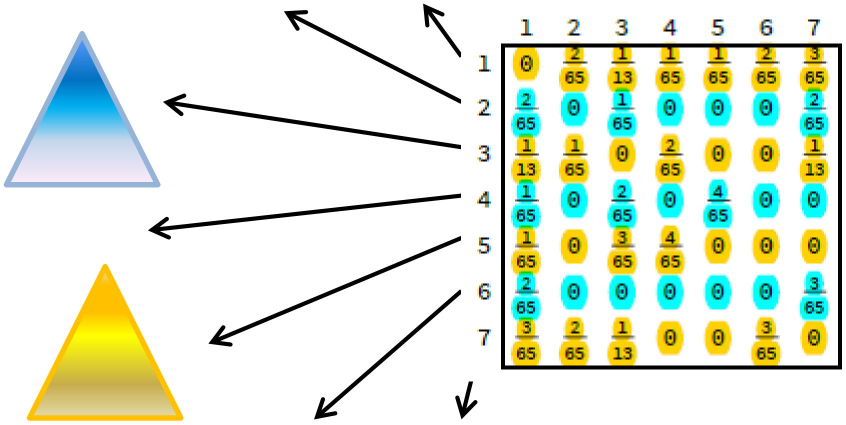
Security of vSwitch







Thank you! Questions?



Further Reading

Demand-aware networks

A survey!

Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity
Klaus-Tycho Foerster and Stefan Schmid.
SIGACT News, June 2019.

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks (Editorial)
Chen Avin and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), October 2018.

Measuring the Complexity of Network Traffic Traces
Chen Griner, Chen Avin, Manya Ghobadi, and Stefan Schmid.
arXiv, 2019.

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.

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.

DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks
Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu.
IEEE/ACM International Symposium on Quality of Service (**IWQoS**), Phoenix, Arizona, USA, June 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

What-if analysis

P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures
Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen.
14th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Heraklion/Crete, 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.

Secure sampling and dataplane

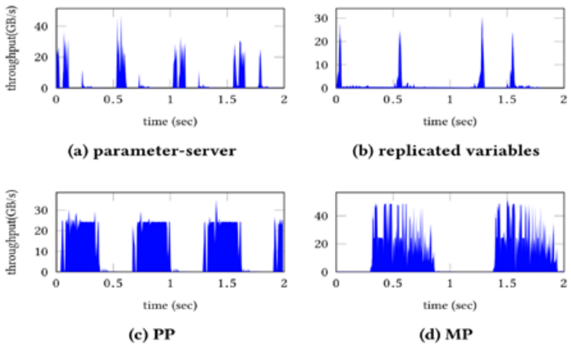
Preacher: Network Policy Checker for Adversarial Environments
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
38th International Symposium on Reliable Distributed Systems (**SRDS**), Lyon, France, October 2019.
MTS: Bringing Multi-Tenancy to Virtual Switches
Kashyap Thimmaraju, Saad Hermak, Gabor Retvari, and Stefan Schmid.
USENIX Annual Technical Conference (**ATC**), Renton, Washington, USA, July 2019.
Taking Control of SDN-based Cloud Systems via the Data Plane (Best Paper Award)
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
ACM Symposium on SDN Research (**SOSR**), Los Angeles, California, USA, March 2018.

Backup Slides

How Predictable is Traffic?

Even if reconfiguration fast, control plane (e.g., data collection) can become a bottleneck. However, many good examples:

- Machine learning applications
- Trend to disaggregation (specialized racks)
- Datacenter communication dominated by elephant flows
- Etc.



ML workload (GPU to GPU):
deep convolutional neural network
Predictable from their dataflow graph