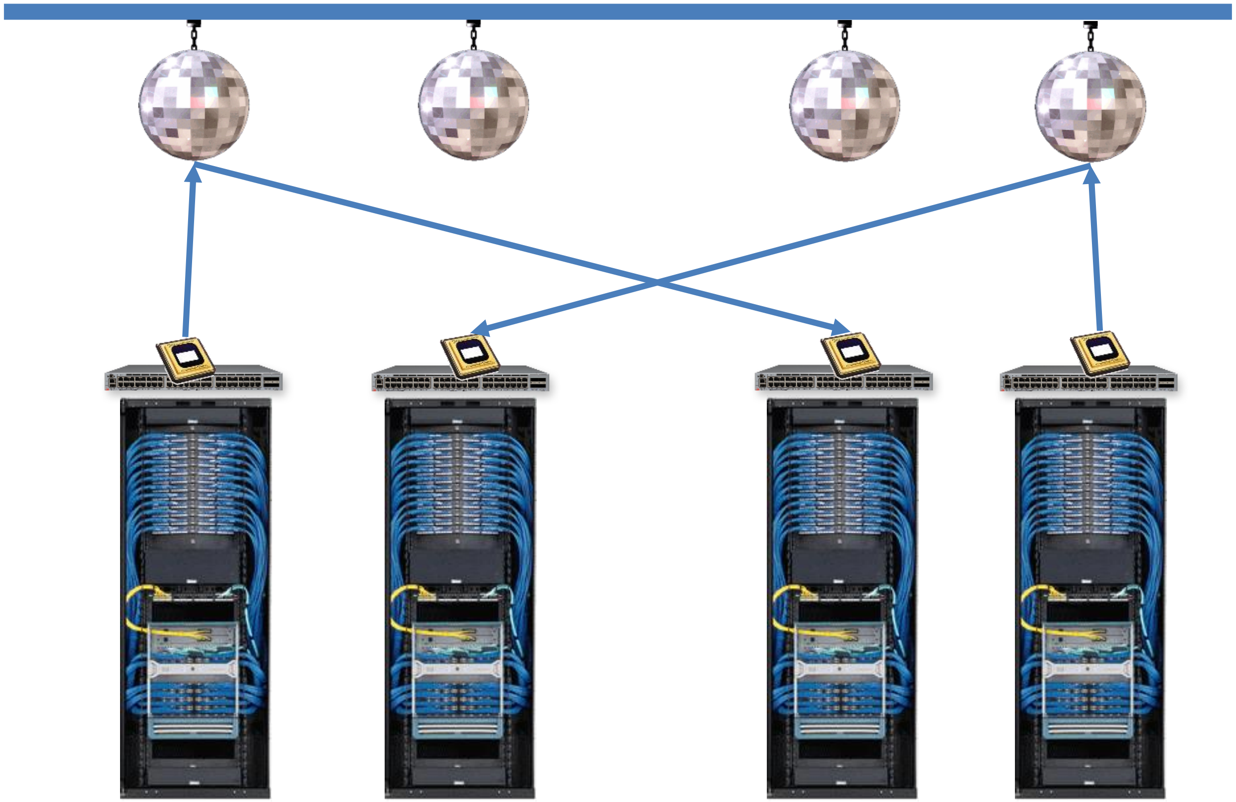
Networks in the Disco:
Algorithms for Demand-Aware and Self-Adjusting Networks

Stefan Schmid (University of Vienna)

Motivation: Free-Space Optics

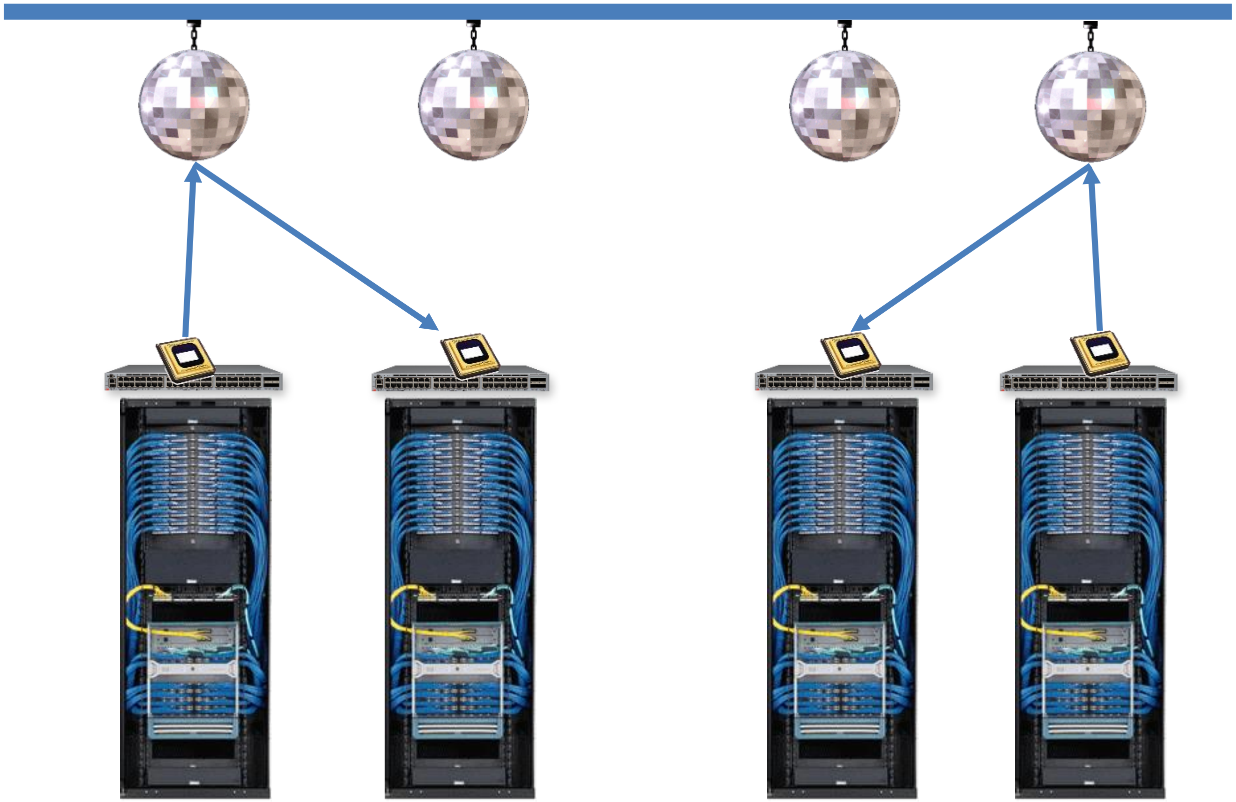
(*ProjecToR*)



t=1

Motivation: Free-Space Optics

(*ProjecToR*)



t=2

ProjecToR @ SIGCOMM 2016

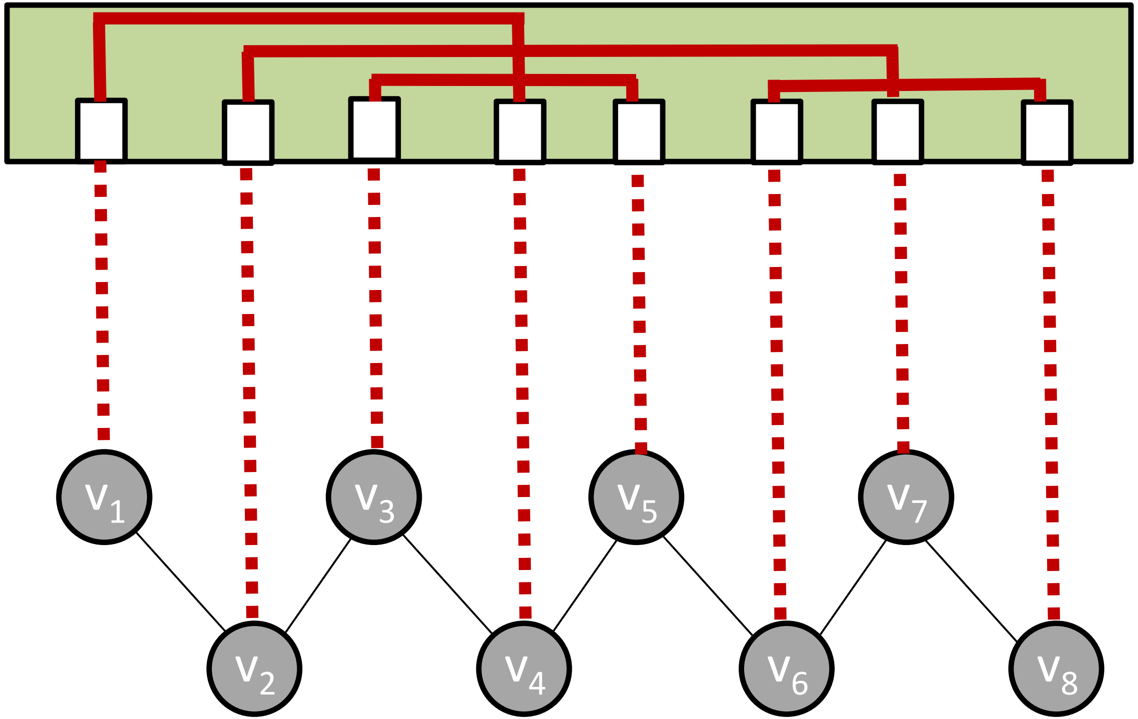
Also: Reconfigurable Optical Switches

(*Helios*, *c-Through*, etc.)

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

t=1

Static topology:
electric



Matching!

Also: Reconfigurable Optical Switches

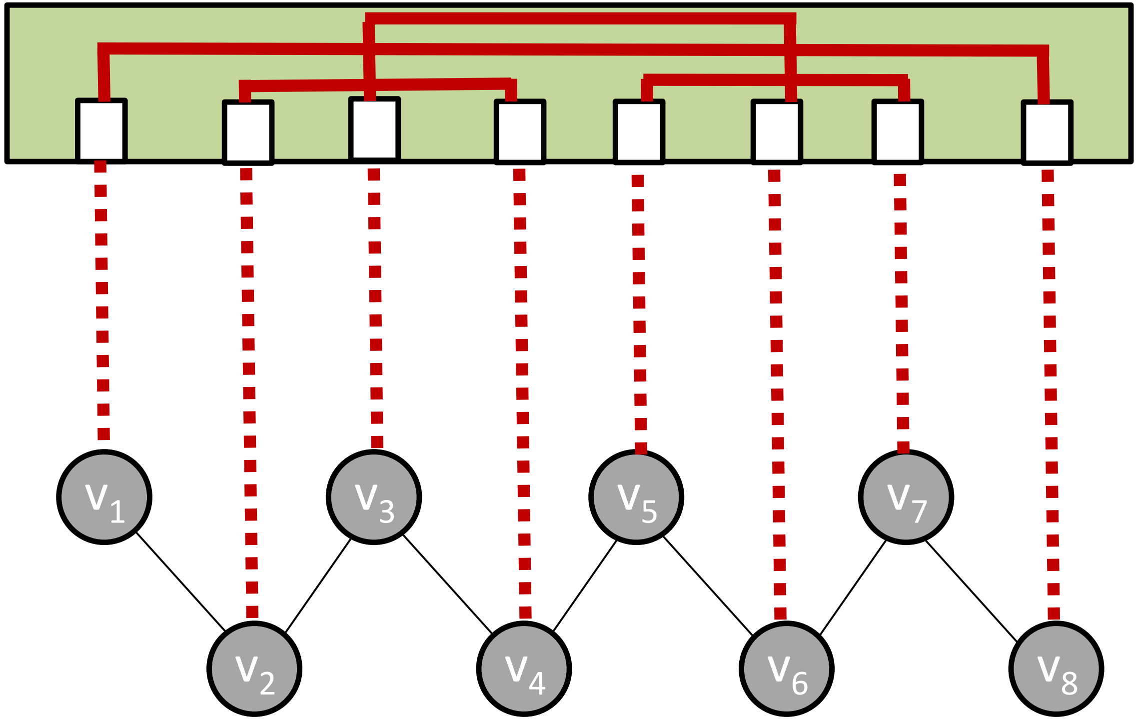
(*Helios*, *c-Through*, etc.)

Matching!

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

t=2

Static topology:
electric



Emerging Technologies

Movable Antennas

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

60GHz Wireless Communication

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

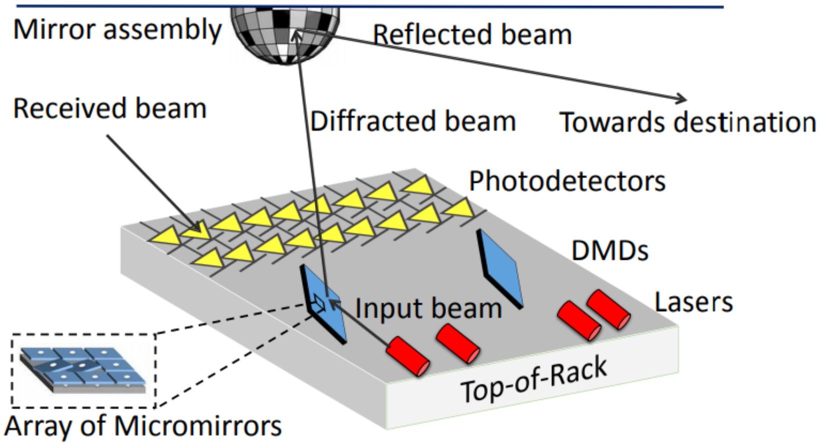
Free-Space Optics

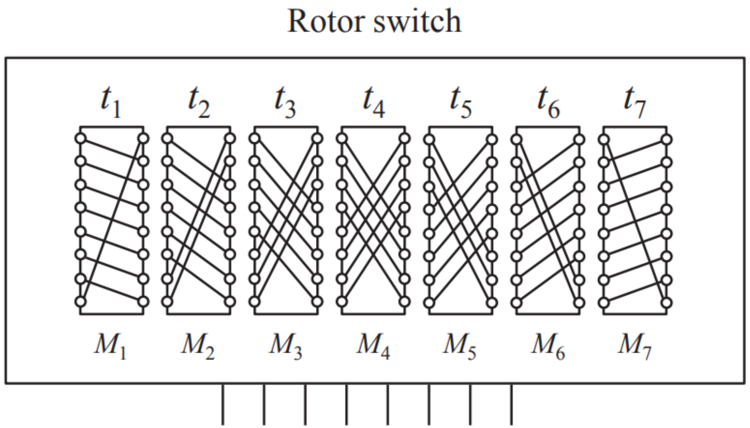
- Ghobadi et al., “***Projector***: Agile reconfigurable data center interconnect,” SIGCOMM 2016.
- Hamedazimi et al. “***Firefly***: A reconfigurable wireless data center fabric using free-space optics,” CCR 2014.

Optical Circuit Switches

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

Etc.!

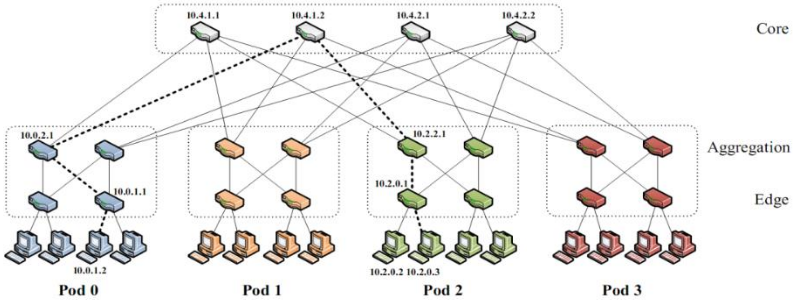




Observation: Technology Enables “Demand-Aware Networks”

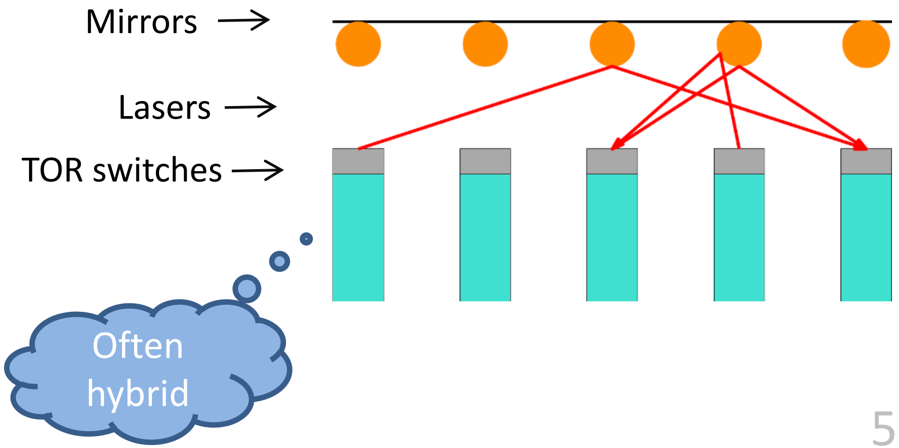
Traditional Networks

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



DANs

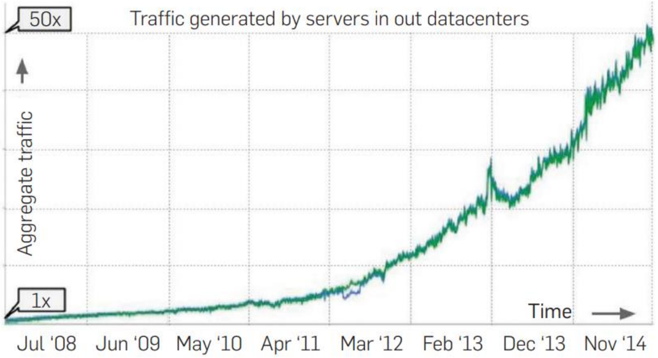
- **Demand-Aware Network (DAN)**
  - Optimized ***toward the workload*** it serves (e.g., **route length**)
  - Statically or ***even dynamically***



Why…?

Growing Traffic and Cost…

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

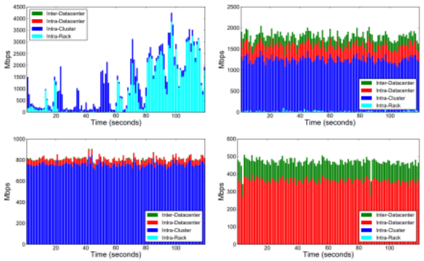


Source: Jupiter Rising. SIGCOMM 2015.

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

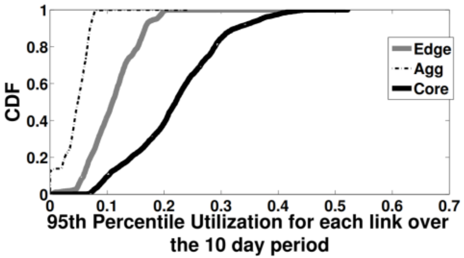
… But Much Structure!

Facebook

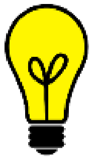


Inside the Social Network's (Datacenter) Network @ SIGCOMM 2015

Benson et al.

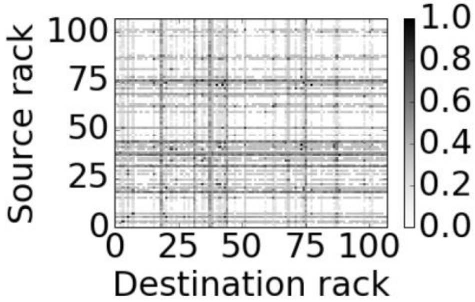


Understanding Data Center Traffic Characteristics @ WREN 2009



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

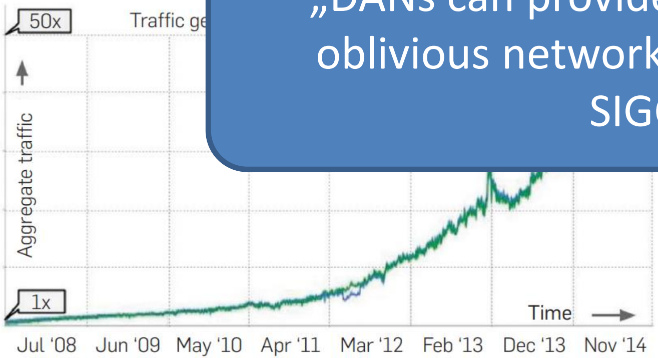
Microsoft



ProjecToR @ SIGCOMM 2016

Growing Traffic and Cost…

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



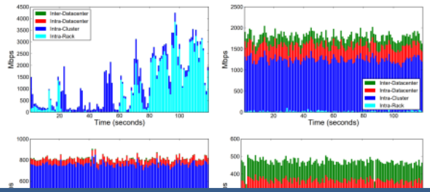
Source: Jupiter Rising. SIGCOMM 2015.

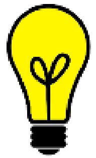
Aggregate server traffic in **Google's datacenter fleet**

„DANs can provide same performance as demand-oblivious networks at 25-40% lower costs." Firefly, SIGCOMM CCR, 2014.

… But Much Structure!

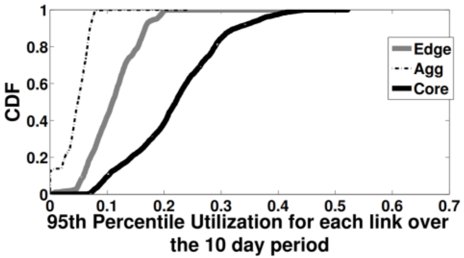
Facebook





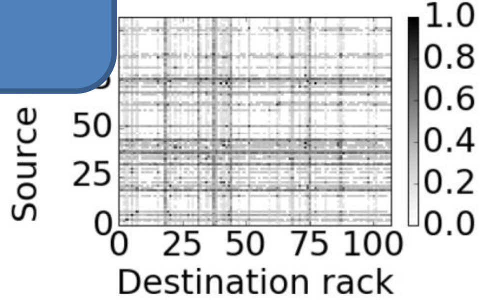
Spatial (***sparse!***) and emporal **locality**

Benson et al.



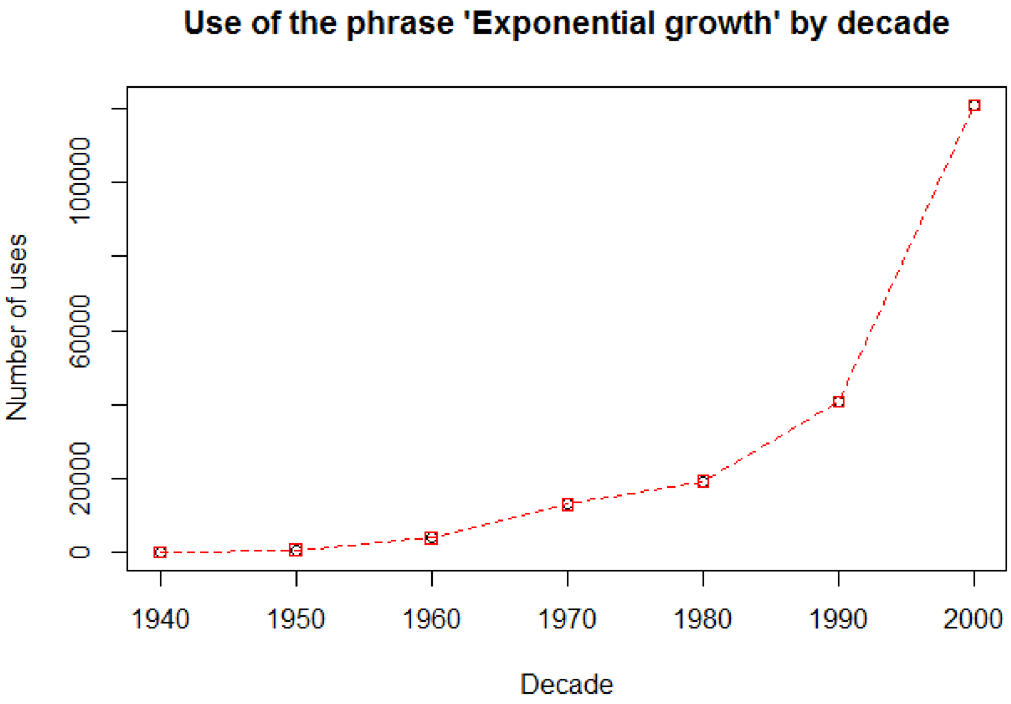
Understanding Data Center Traffic Characteristics @ WREN 2009

Microsoft



ProjecToR @ SIGCOMM 2016

Fun Fact



Data from Google Scholar

Roadmap

- Motivation: Demand-Aware Networks ✓
- **Principles of Static Demand-Aware Network Designs**
- Principles of Dynamic Demand-Aware Network Designs
- Principles of Decentralized Approaches

A “Simple” DAN Design Problem

Input: Workload

Destinations

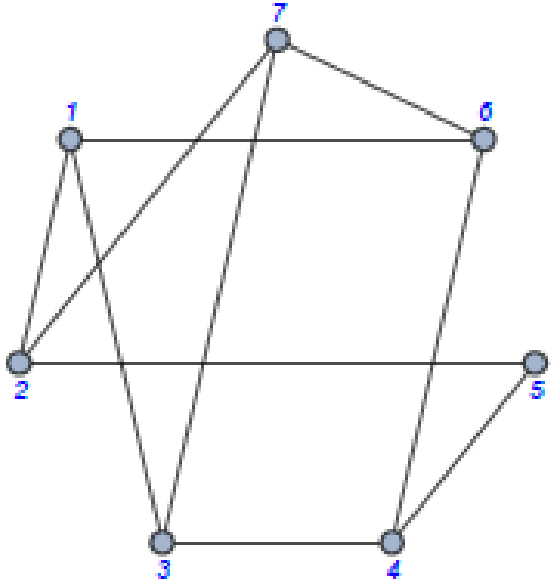
Sources

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

Demand matrix: joint distribution

design

Output: DAN



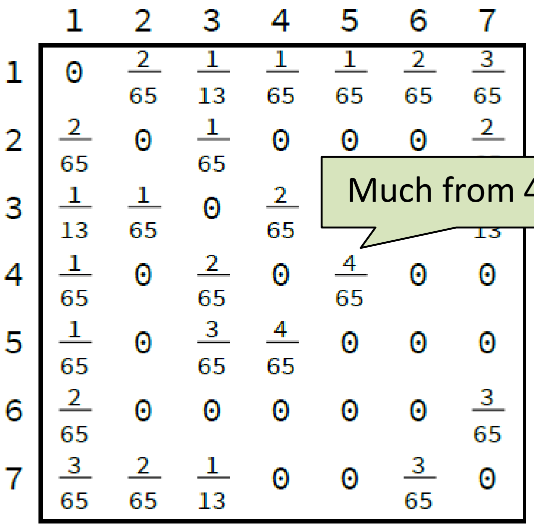
… of ***constant degree*** (**scalability**)

A “Simple” DAN Design Problem

Input: Workload

Destinations

Sources

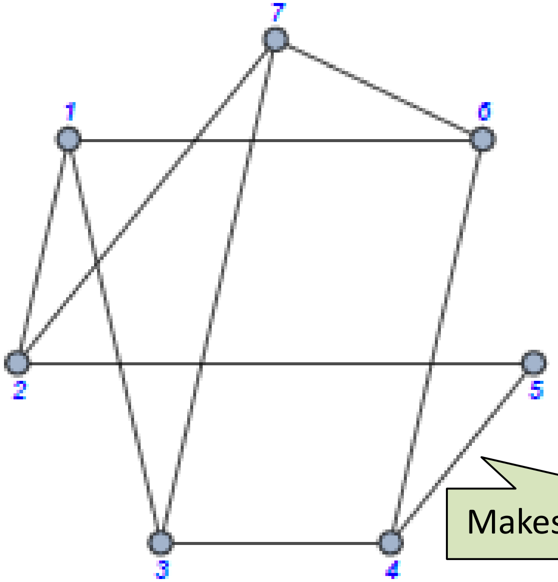


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	2 / [illegible]
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	[illegible]	[illegible]	[illegible] / 13
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$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

Much from 4 to 5.

design

Output: DAN



Makes sense to add link!

Demand matrix: joint distribution

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

A “Simple” DAN Design Problem

Input: Workload

Destinations

1 communicates to many.

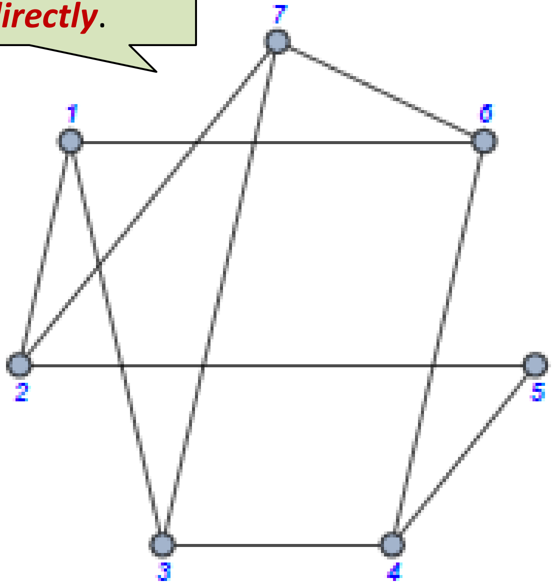
Sources

						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**)

A "Simple" DAN Design Problem

Input: Workload

Destinations

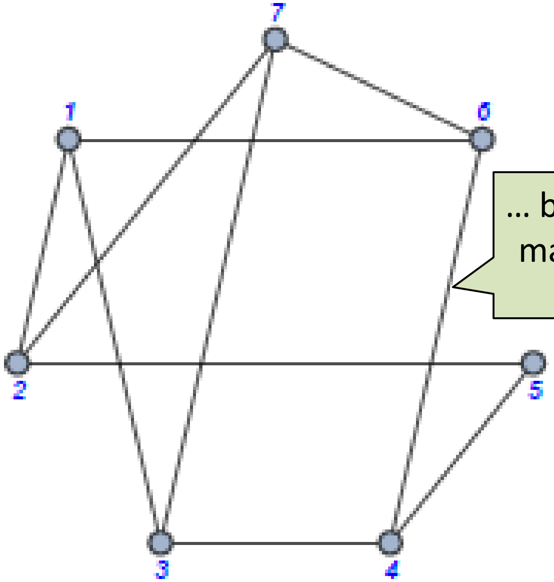
Sources

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

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**)

Case Study: DAN for Short Routes

Shorter routes: smaller bandwidth footprint, lower latency, less energy, …

More Formally: DAN Design Problem

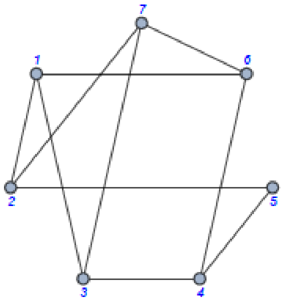
Input:

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

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

Output:

N: DAN



Bounded degree
$\Delta=3$

Objective:

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

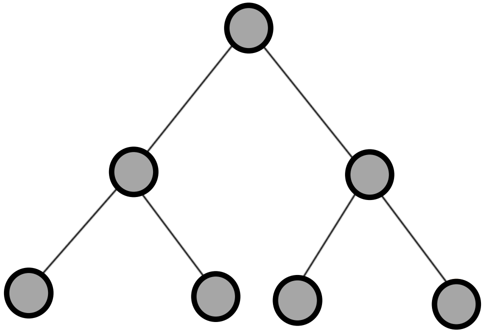
Path length ***on DAN N***.

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

Frequency

Some Examples

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

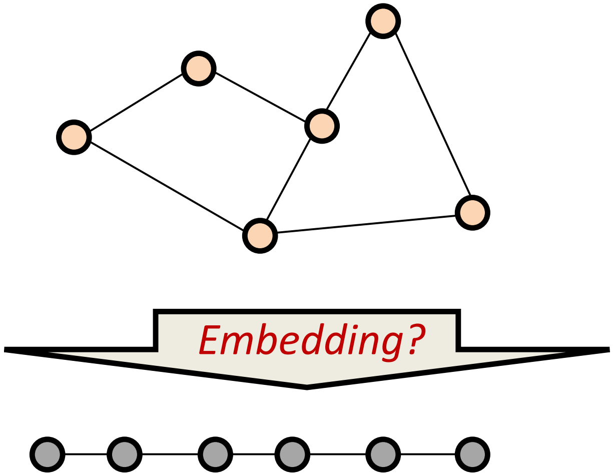


- DANs of Δ = 2:
  - E.g., set of **lines** and **cycles**

How hard is it to design a DAN?

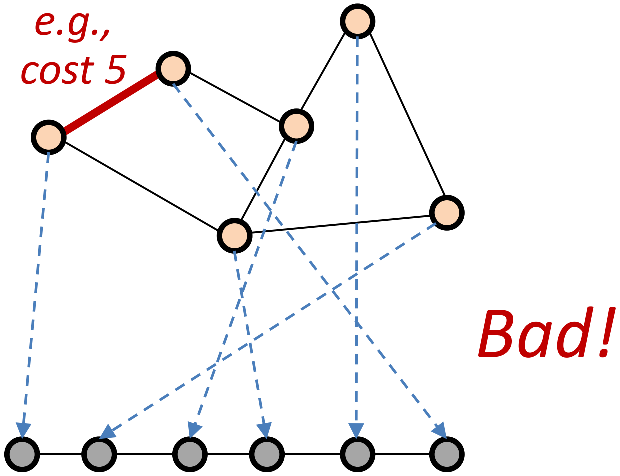
DAN design can be NP-hard

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



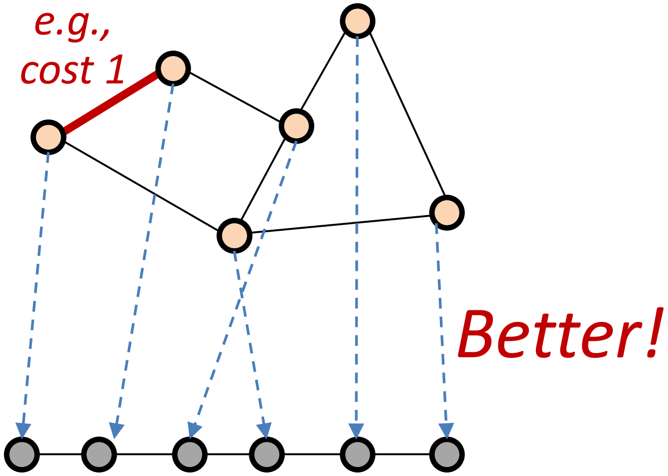
DAN design can be NP-hard

- **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

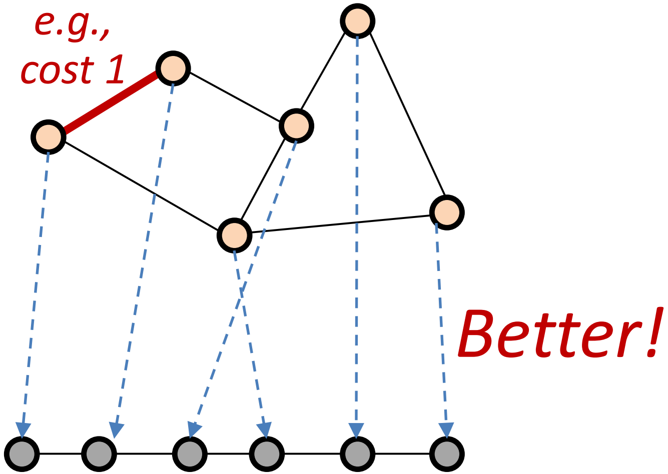
- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A "Virtual Network Embedding Problem", VNEP
  - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

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

NP-hard, and so is DAN design.



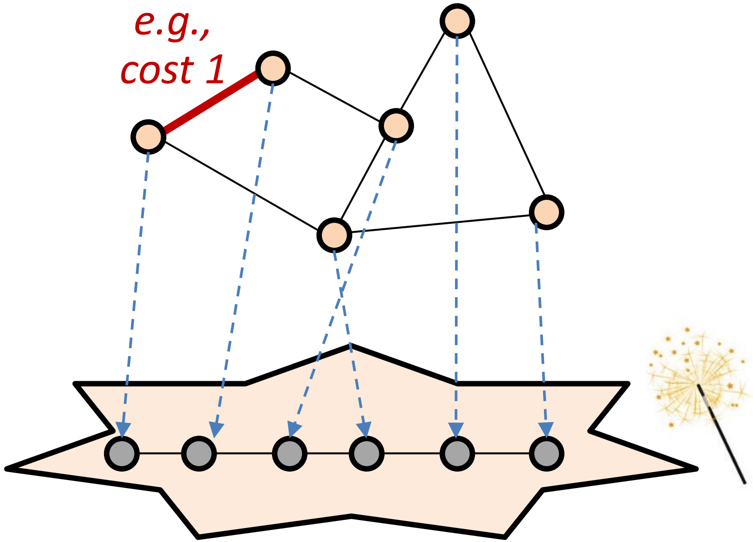
DAN design can be NP-hard

- **Example Δ = 2:** A Mini inear Arrangement (M blem
  - A “Virtual N bedding Problem”, VNEP
  - *Mini* engths 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!

So: How useful are DANs?

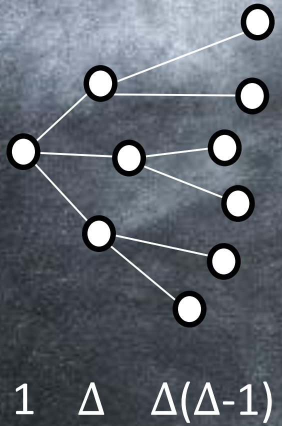
As always in computer science (e.g., also in coding, in self-adjusting datastructures, etc.): ***it depends!*** ☺

Expected Path Length in Traditional Networks?

Theorem (Traditional Networks):

Constant-degree networks have at least logarithmic diameter.

Proof.



In *k* steps, reach at most $1+ \Sigma \Delta(\Delta -1)^i$

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

□

Example: Clos, Bcube, Xpander.

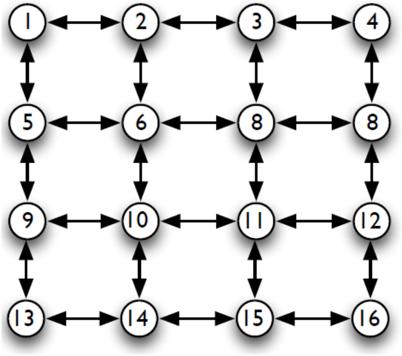
Can DANs do better?

Can DANs do better?

In general not really, e.g. in all-to-all communication (*clique*): ***logarithmic diameter unavoidable***.

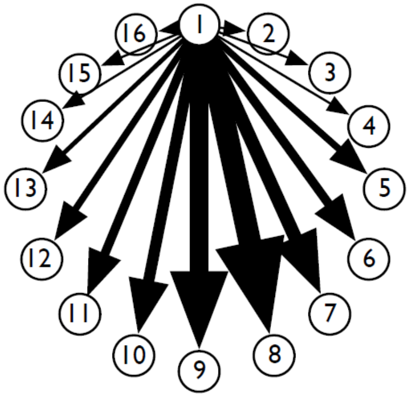
But sometimes, DANs can be much better!

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?

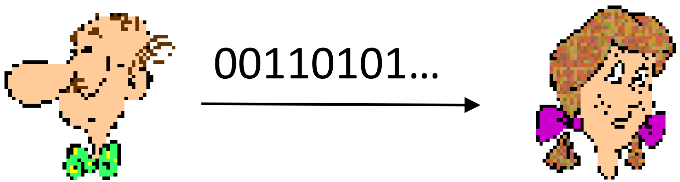
So on what does it depend?

We argue: on the **“entropy” of the demand**!

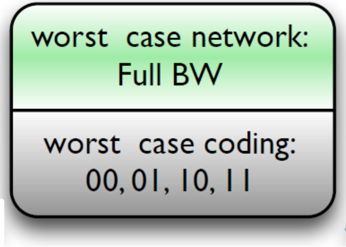
?

An Analogy to Coding

„Coming to Wroclaw?“



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


worst case network:
Full BW

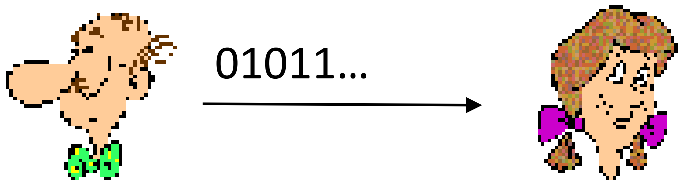
worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

„Coming to Wroclaw?“

An Analogy to Coding



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?

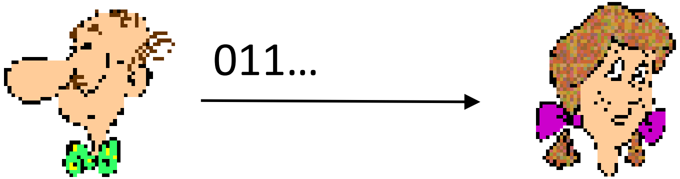
static
Demand-Aware Nets

entropy / symbol

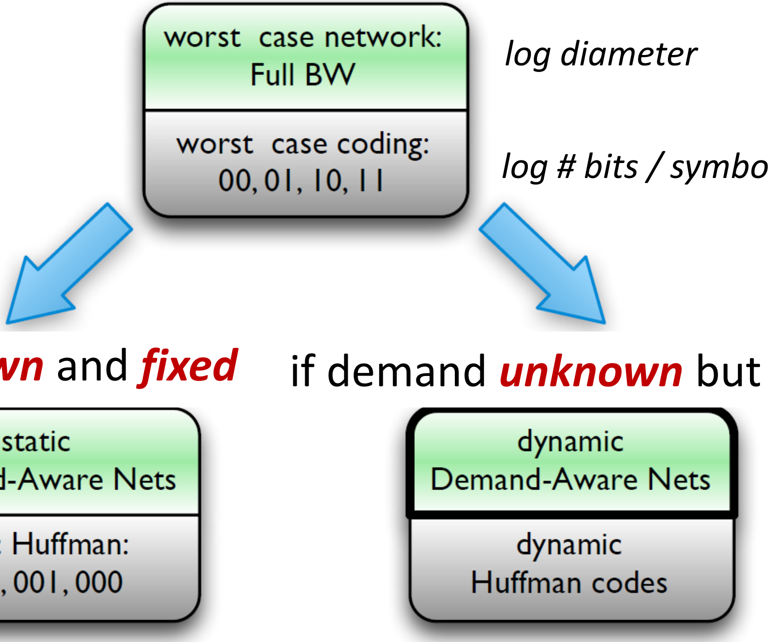
static Huffman:
1, 01, 001, 000

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

DAN!

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

SAN!

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

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

entropy?

entropy / symbol

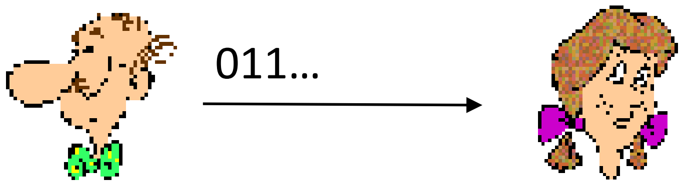
if demand ***unknown*** but ***reconfigurable***

dynamic
Demand-Aware Nets

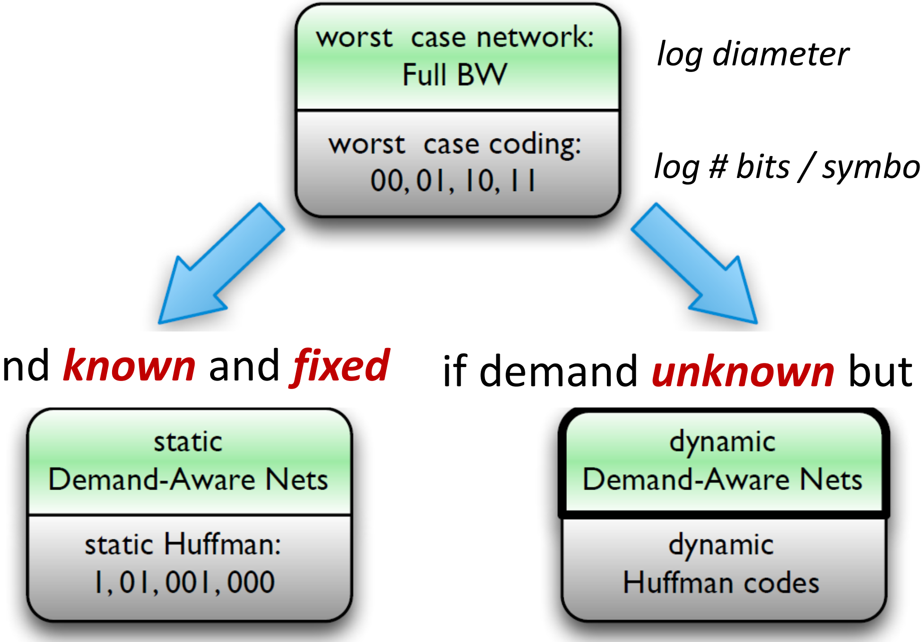
dynamic
Huffman codes

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

DAN!

SAN!

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

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

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

Can exploit ***spatial locality***!

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

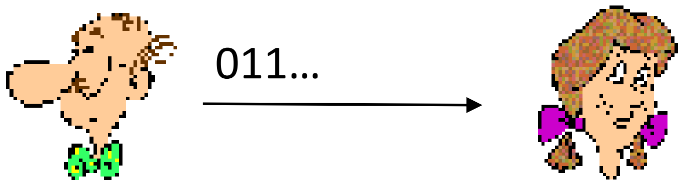
dynamic
Demand-Aware Nets

dynamic
Huffman codes

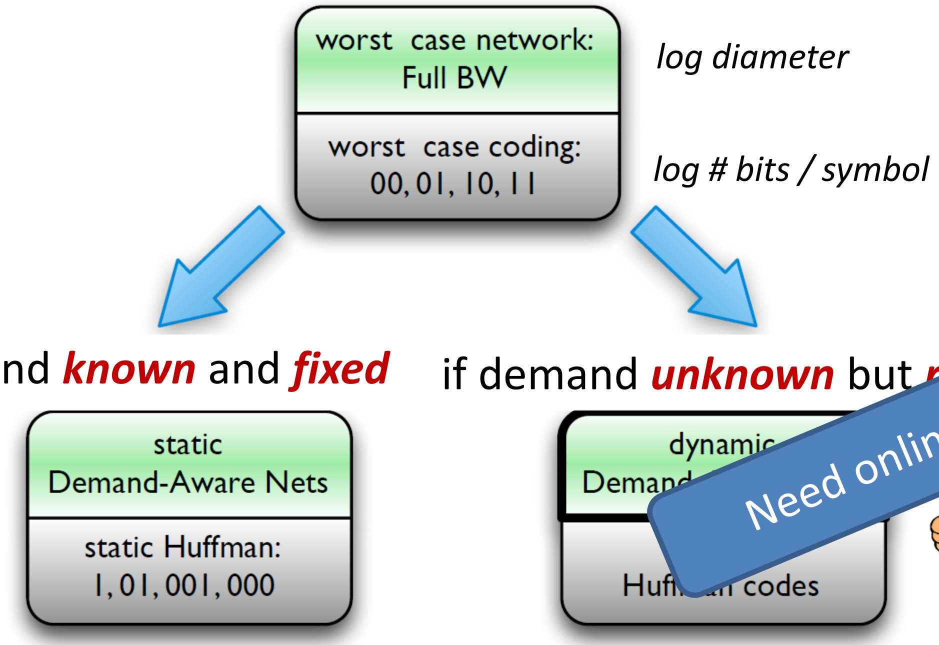
Additionally exploit ***temporal locality***!

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***

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

if demand ***unknown*** but r

dynamic
Demand

Huf an codes

Need online algorithms!

Can exploit
spatial locality!

Additionally exploit
temporal locality!

Analogous to *Datastructures*: Oblivious…

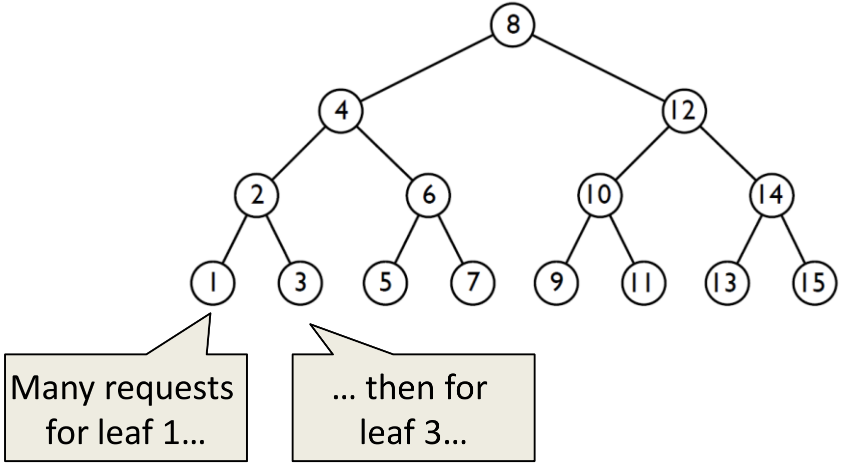
- Traditional, **fixed** BSTs do not rely on any assumptions on the demand
- Optimize for the **worst-case**
- Example **demand**:

  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*!

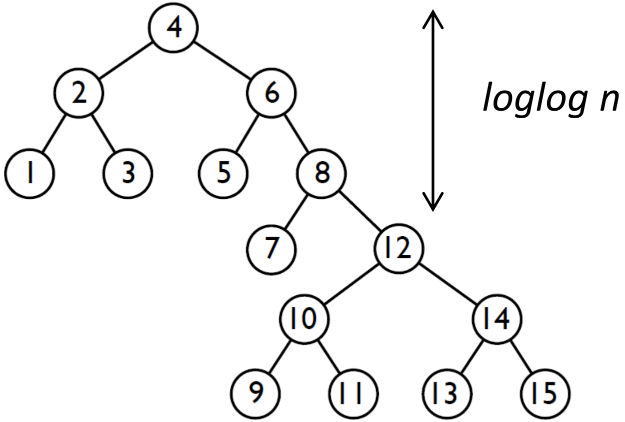


Many requests for leaf 1…

… then for leaf 3…

… Demand-Aware …

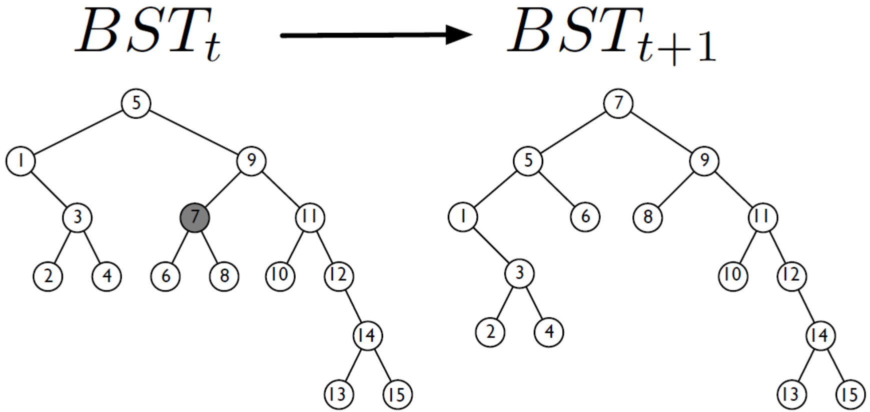
- **Demand-aware fixed** BSTs can take advantage of ***spatial locality*** of the demand
- E.g.: place frequently accessed elements close to the root
- E.g., **Knuth/Mehlhorn/Tarjan** trees
- Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
  - Amortized cost ***O(loglog n)***



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

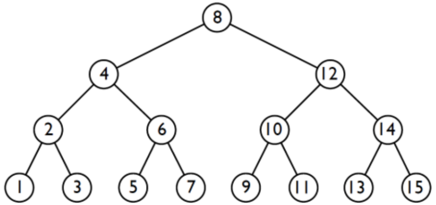
… Self-Adjusting!

- **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

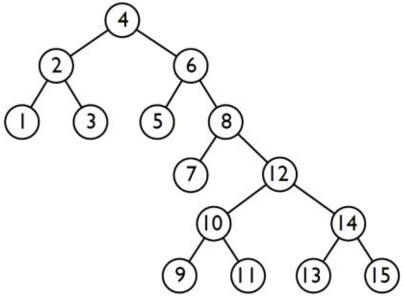
Oblivious



Lookup

O(log n)

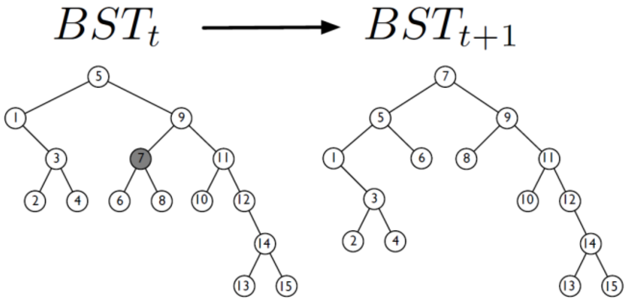
Demand-Aware



Exploit **spatial locality**:

empirical entropy O(loglog n)

Self-Adjusting

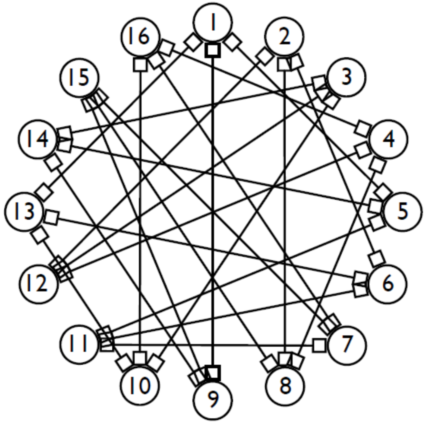


Exploit **temporal locality** as well:

O(1)

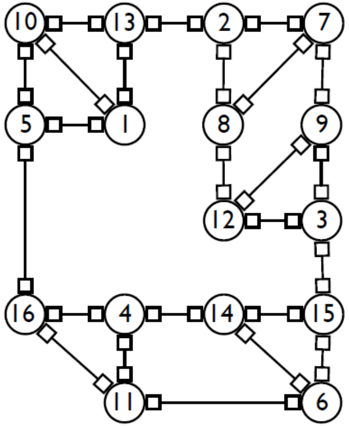
Analogously for Networks

Oblivious



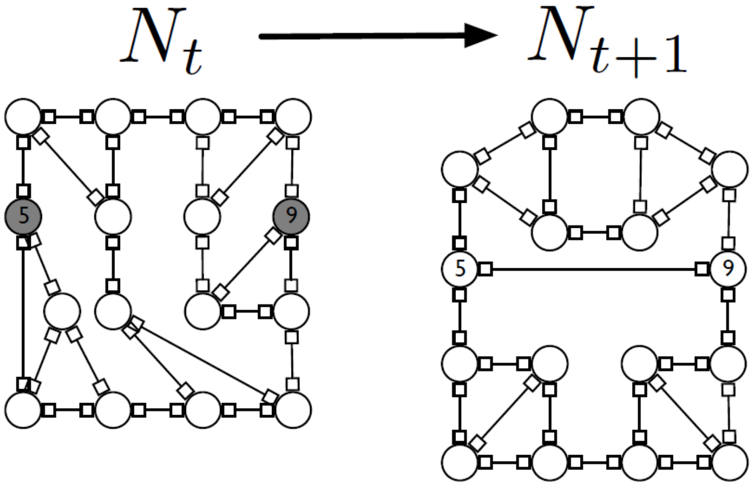
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

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.

Limitations of (Static) DANs: Entropy-Based Lower Bounds

Indeed!

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

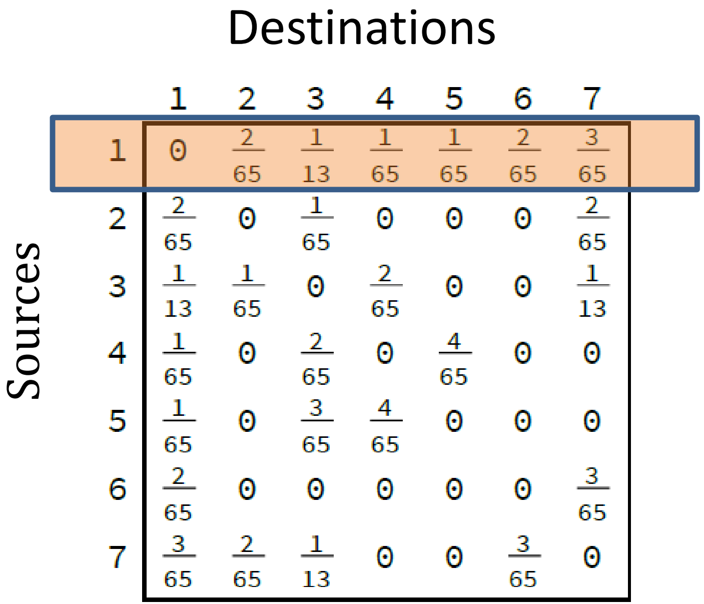
- DAN just for a ***single (source) node 1***: cannot do better than Δ-ary **Huffman tree** for its destinations [0,1/65,1/13,1/65,1/65,2/65,3/65]
  - resp. **Knuth/Mehlhorn/Tarjan** tree if search property required

- 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 Da

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 [0,1/65,1/13,1/65,1/65,2/65,3/65]
  - resp. **Knuth/Mehlhorn/Tarjan** tree if search property required
- 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 [0,1/65,1/13,1/65,1/65,2/65,3/65]
  - resp. **Knuth/Mehlhorn/Tarjan** tree if search property required
- How good can this tree be?

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

So: what is the entropy of the whole demand?

Lower Bound & Entropy of the Demand

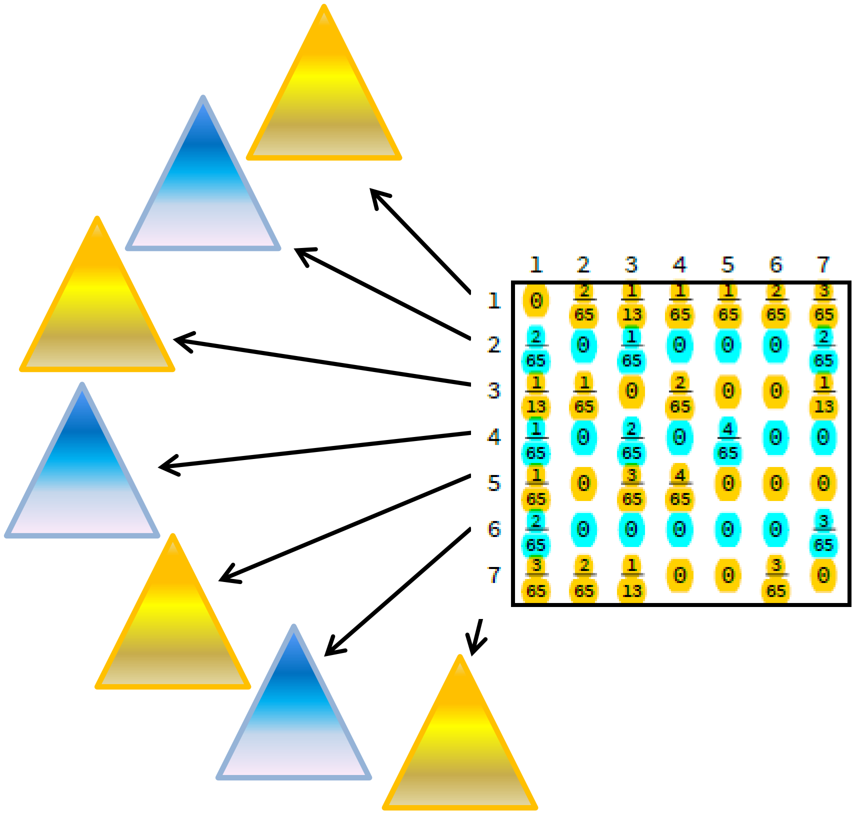
sources

destinations

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

entropy

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



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

Lower Bound & Entropy of the Demand: Sources + Destinations

Do this in **both dimensions**:

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

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

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

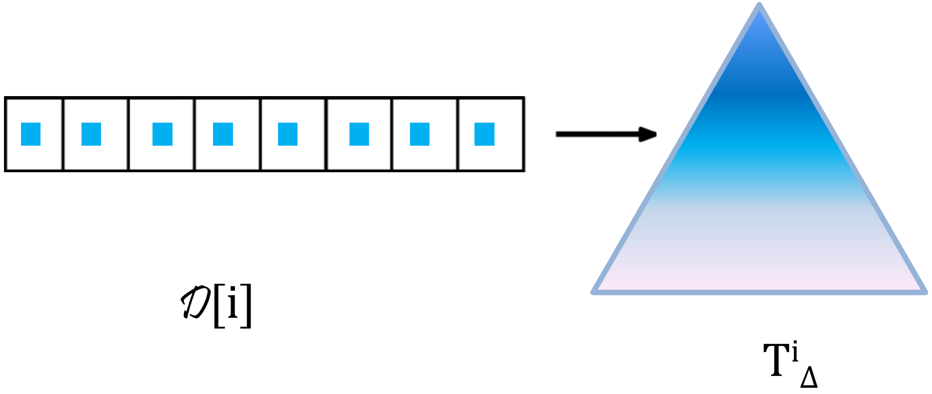
$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13}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

$\mathcal{D}$

Can DANs Match The Entropy Speed Limit?
Upper Bounds

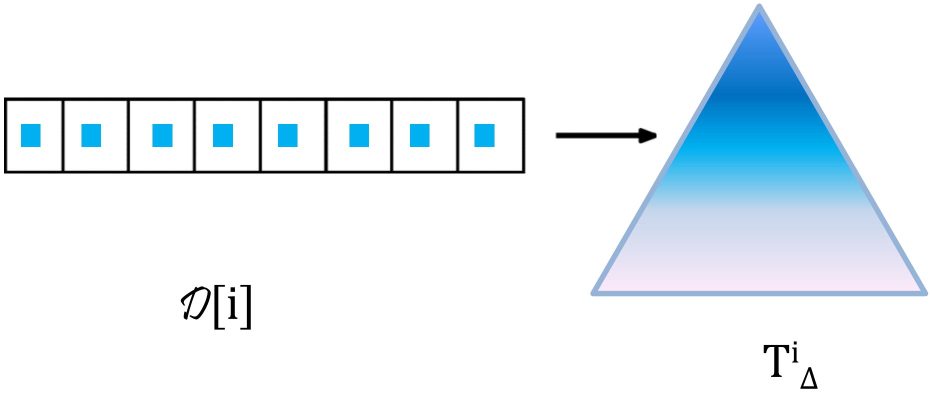
Ego-Trees Revisited

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



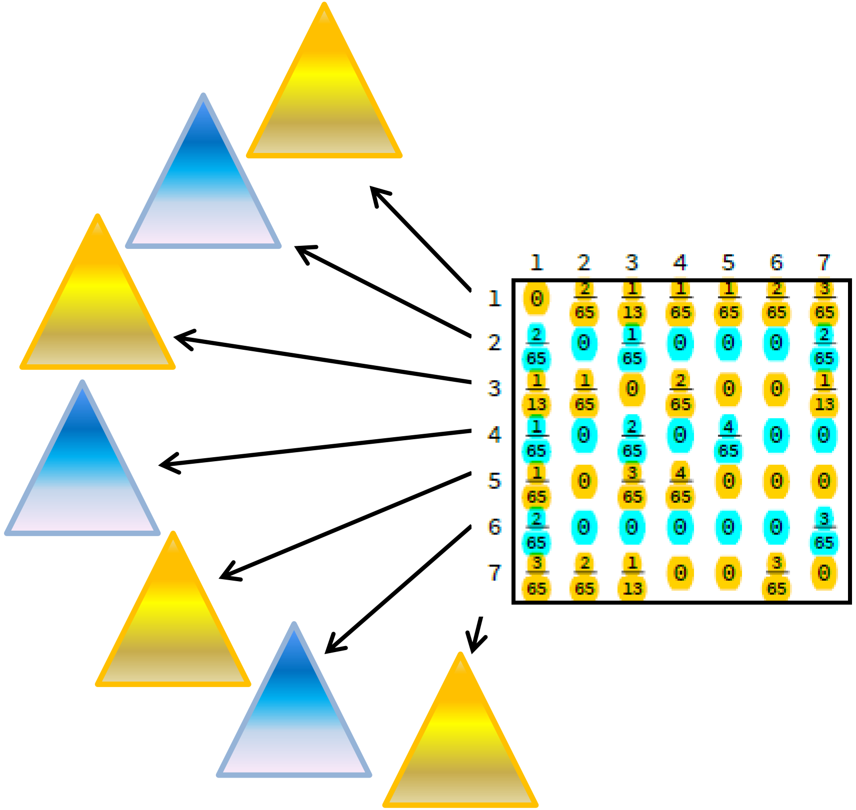
Ego-Trees Revisited

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



$\mathcal{D}[i]$

T^i_Δ

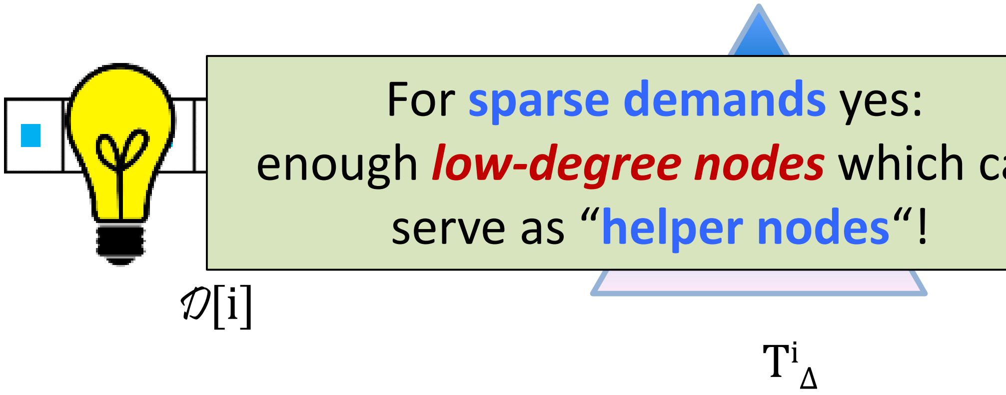


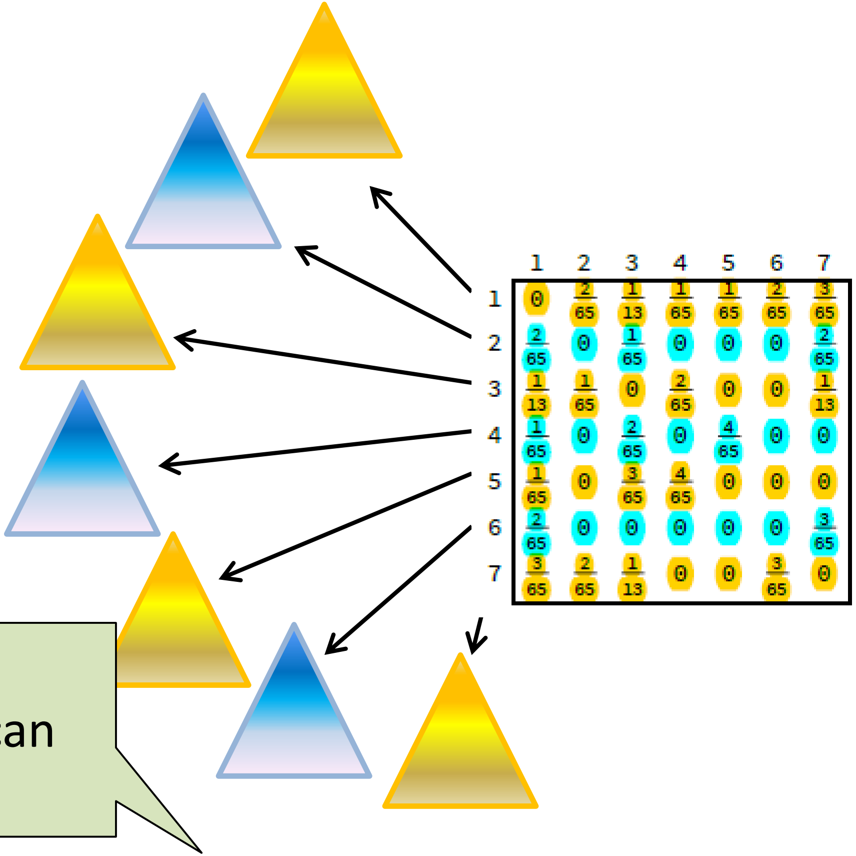
$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13}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

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

Ego-Trees Revisited

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





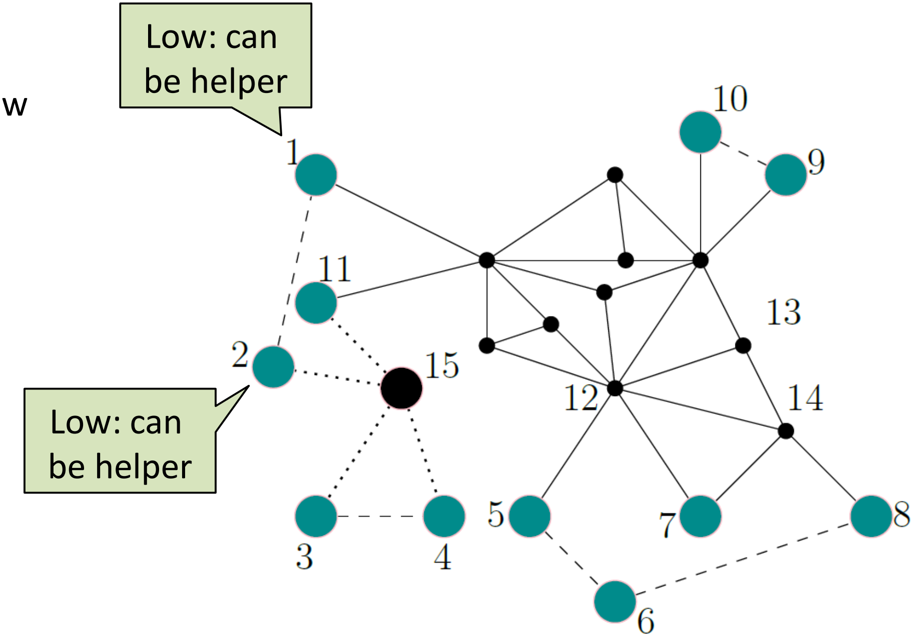
	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$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

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***?

DAN for Sparse Demand

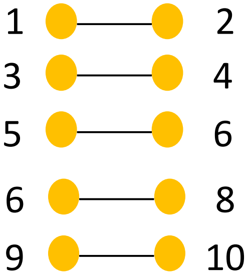
- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”

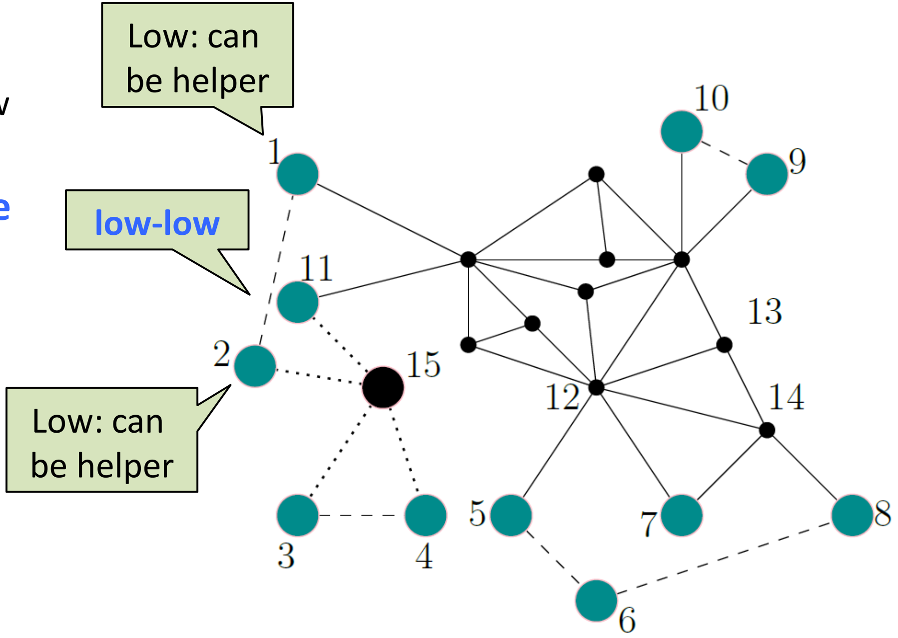

Low: can be helper

Low: can be helper

DAN for Sparse Demand

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges and the binary **tree** into DAN and remove from demand



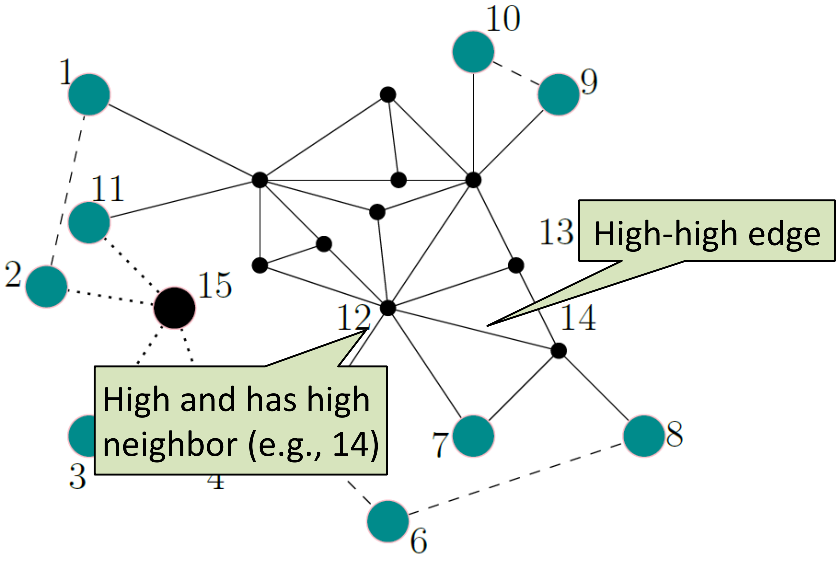

Low: can be helper

low-low

Low: can be helper

DAN for Sparse Demand

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges and the binary **tree** into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distanced increased by **+1**

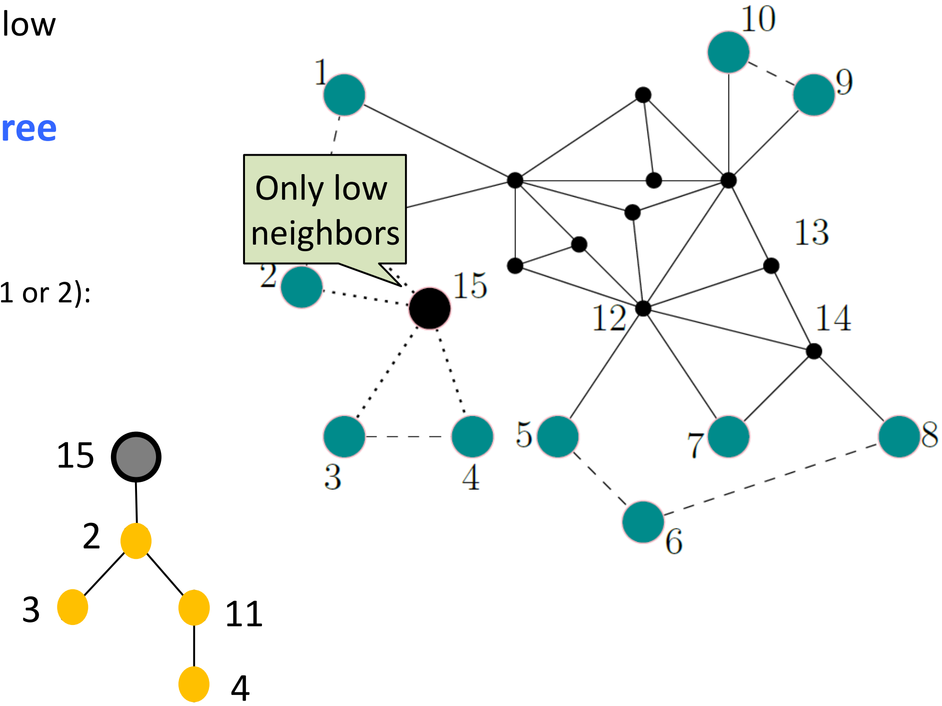


High-high edge

High and has high neighbor (e.g., 14)

DAN for Sparse Demand

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges and the binary **tree** into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distanced increased by **+1**
- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors



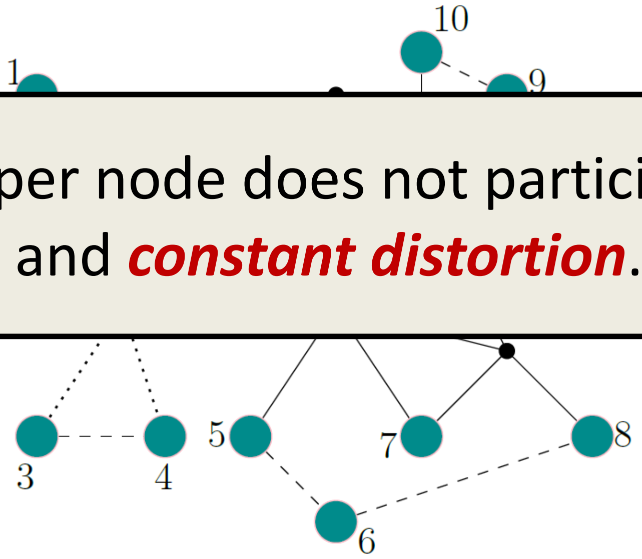
Only low neighbors

DAN for Sparse Demand

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”

Theorem [Asymptotic Optimality]: Helper node does not participate in many trees, so ***constant degree***, and ***constant distortion***.

- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors



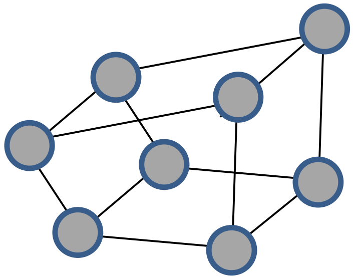
Remark: The Problem is Related To Spanners

- Sparse, distance-preserving (low-distortion) **spanners**
- But:
  - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
  - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
  - We require ***constant degree***

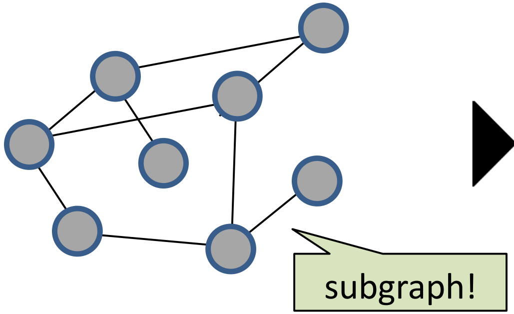
Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant**, **sparse**) spanner for this request graph: then we can design a constant degree DAN providing an ***optimal EPL*** (i.e., O(H(X|Y)+H(Y|X)).

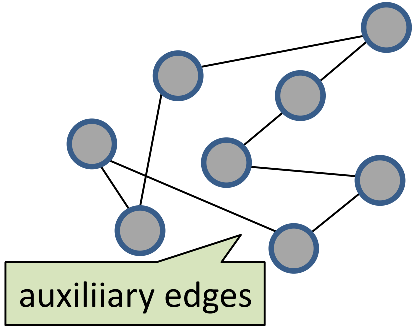
r-regular *and* ***uniform*** **demand***:*



Sparse, ***irregular*** ***(constant)*** **spanner***:*



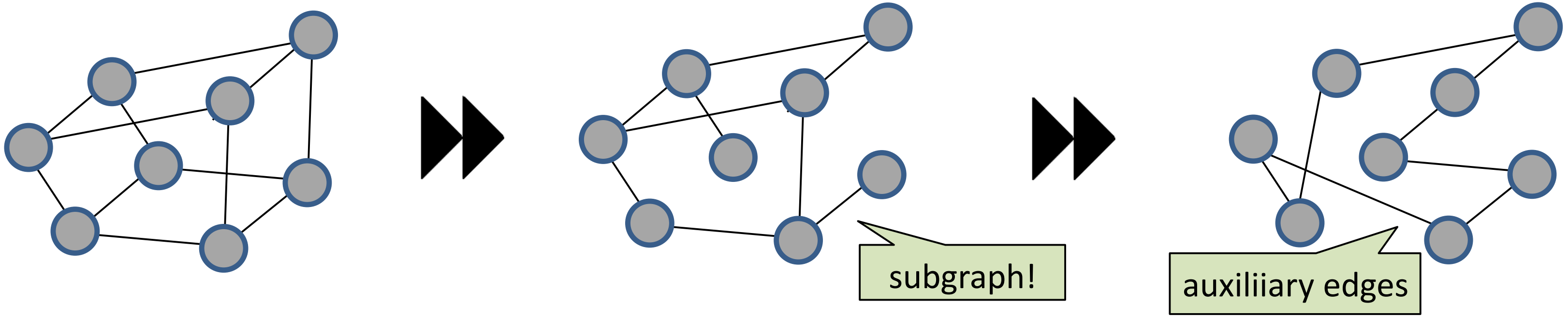
Constant *degree* ***optimal*** **DAN** (EPL at most ***log r***):



Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant**, **sparse**) spanner for this request graph: then we can design a constant degree DAN provi (Y|X)).

Optimal: in r-regular graphs, conditional entropy is log r.


r-regular and *uniform* **demand**:

Sparse, irregular ***(constant)*** **spanner**:

Constant *degree* ***optimal*** **DAN** (EPL at most ***log r***):

Proof Idea

- **Degree reduction** again, this time ***from sparse spanner*** (before: from sparse demand graph)

Corollaries

- Optimal DAN designs for
  - **Hypercubes** (with n log n edges)
  - **Chordal** graphs
  - Trivial: graphs with polynomial degree (**dense graphs**)
  - Graphs of **locally bounded doubling dimension**

Has sparse 3-spanner.

Has sparse O(1)-spanner.

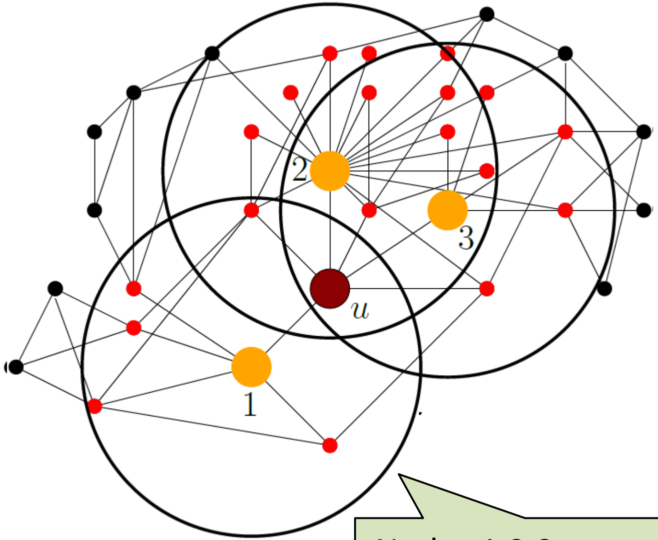
We also know some more algos, e.g., for BSTs.

Another Example: Demands of Locally-Bounded Doubling Dimension

- **LDD**: $G_{\mathcal{D}}$ has a **Locally-bounded Doubling Dimension** (LDD) iff all 2-hop neighbors are covered by 1-hop neighbors of just ***λ*** nodes
  - Note: care only about ***2-neighborhood***

We only consider 2 hops!

- Formally, $B(u, 2) \subseteq \cup_{i=1}^{\lambda} B(v_i, 1)$
- Challenge: can be of ***high degree***!



Nodes 1,2,3 cover 2-hop neighborhood of u.

DAN for Locally-Bounded Doubling Dimension

Lemma: There exists a sparse 9-(subgraph)spanner for LDD.

This ***implies optimal DAN***: still focus on regular and uniform!

Def. (ε-net): A subset V' of V is a **ε-net** for a graph $G = (V,E)$ if

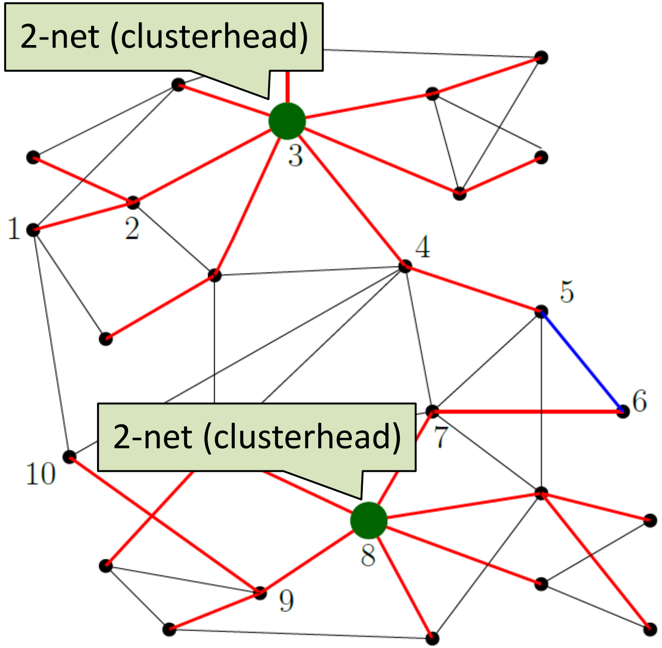
- V' sufficiently "independent": for every $u, v \in V'$, $d_G(u, v) > \boldsymbol{\varepsilon}$
- "dominating" V: for each $w \in V$, $\exists$ at least one $u \in V'$ such that, $d_G(u,w) \leq \boldsymbol{\varepsilon}$

9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

Easy: Select nodes into 2-net **one-by-one** in decreasing (remaining) degrees, **remove 2-neighborhood**. Iterate.

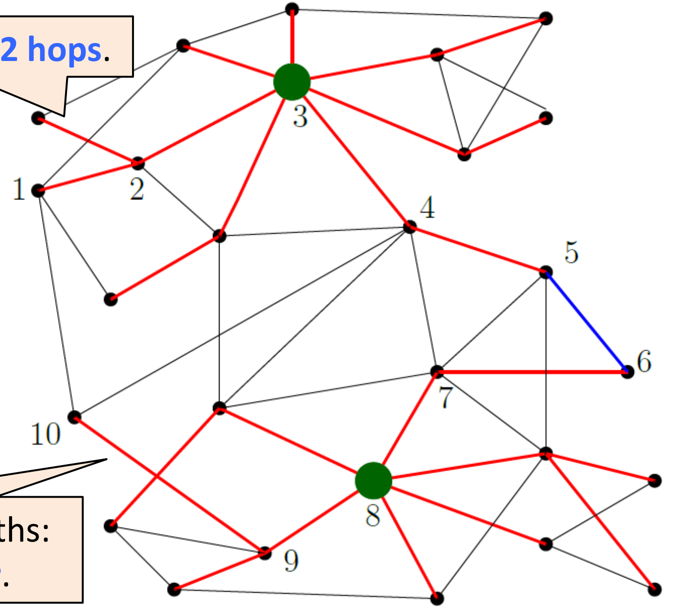


9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Add nodes to one of the closest 2-net nodes



Assign: at most **2 hops**.

Union of these shortest paths:
a **forest**. Add to **spanner**.

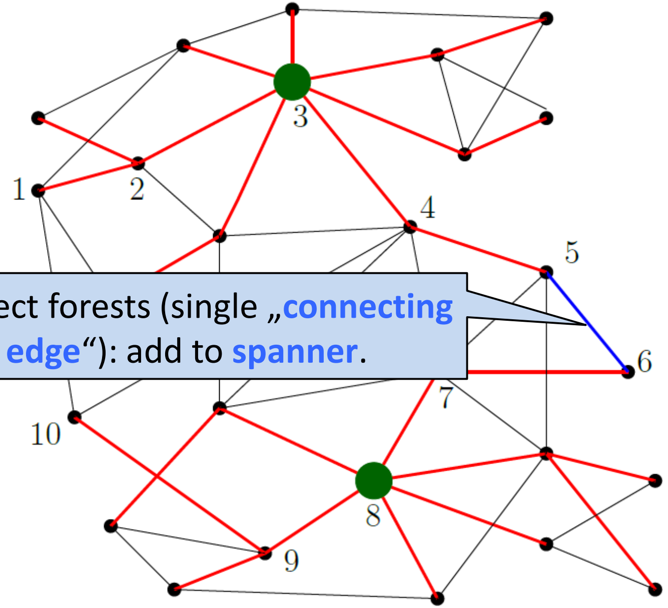
9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Add nodes to one of the closest 2-net nodes

3. Join two clusters if there are edges in between



Connect forests (single „**connecting edge**“): add to **spanner**.

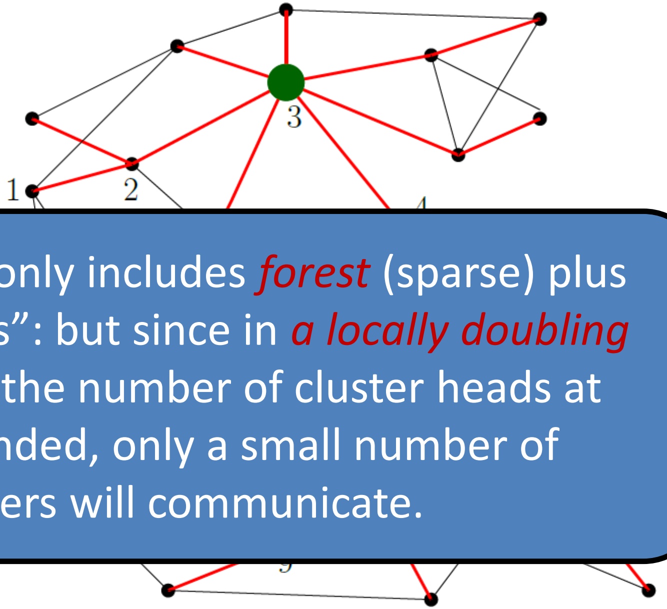
9-Spanner for LDD (= optimal DAN)

Distortion 9: *Short detour* via clusterheads: u,ch(u),x,y,ch(v),v

1. Find a 2-net

2. Add nodes to one of the closest 2-net node

3. Join two clusters edges in between



Sparse: Spanner only includes *forest* (sparse) plus "connecting edges": but since in *a locally doubling dimension graph* the number of cluster heads at distance 5 is bounded, only a small number of neighboring clusters will communicate.

Further Reading

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

Roadmap

- Motivation: Demand-Aware Networks ✓
- Principles of Static Demand-Aware Network Designs ✓
- **Principles of Dynamic Demand-Aware Network Designs**
- Principles of Decentralized Approaches

Objectives and Metrics for Dynamic DANs, i.e. SANs?

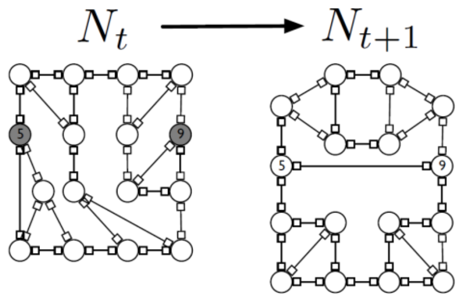
Input for Dynamic DANs

A **sequence σ** = (u1,v1), (u2,v2), (u3,v3)….

chosen arbitrarily

Chosen i.i.d. from initially unknown distribution

A Cost-Benefit Tradeoff



Basic question:

How often to reconfigure?

Short routes

High reconfiguration cost

Tradeoff

Low reconfiguration cost

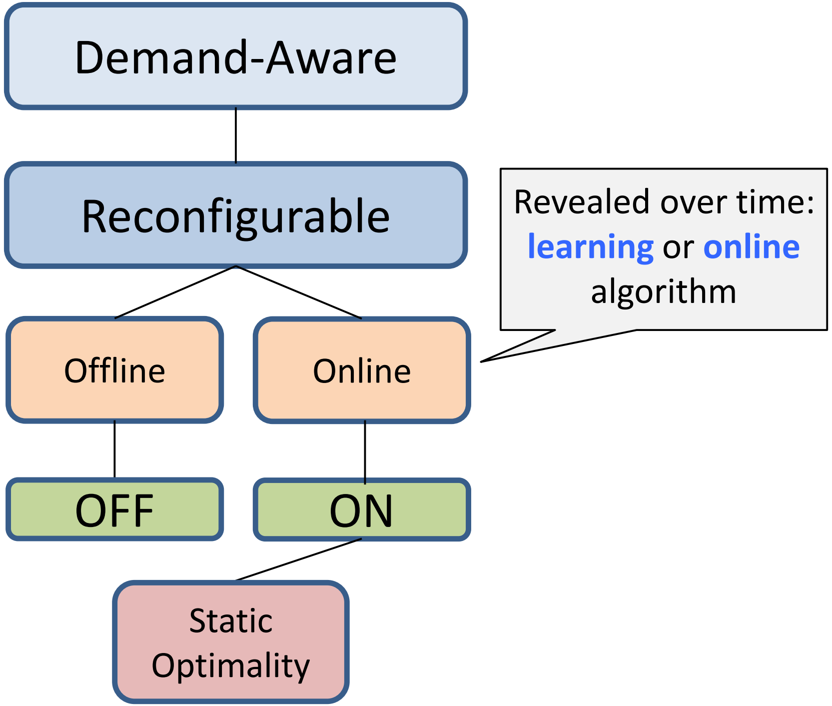
Long routes

A Taxonomy: Reconfigurable Networks

Static Optimality:

“***Not worse than static*** which knows demand ahead of time!”

ρ = Cost(ON)/Cost(STAT*) is constant.



Revealed over time: **learning** or **online** algorithm

Awareness

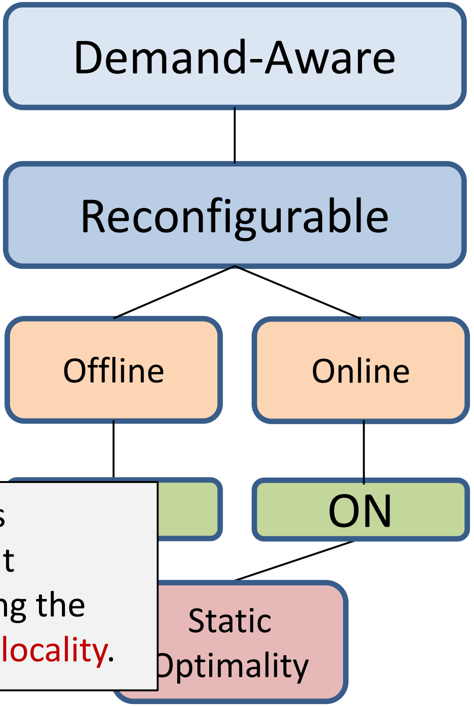
Topology

Input

Algorithm

Property

A Taxonomy: Reconfigurable Networks



Revealed over time: **learning** or **online** algorithm

Awareness

Topology

Input

Algorithm

Property

Static Optimality:

"***Not worse than static*** which knows demand ahead of time!"

ρ = Cost(ON)/Cost(STAT*****) is constant.

Note: may be ***<<1***. ON has advantage of adjusting, but the disadvantage of not knowing the workload. E.g. if much temporal locality.

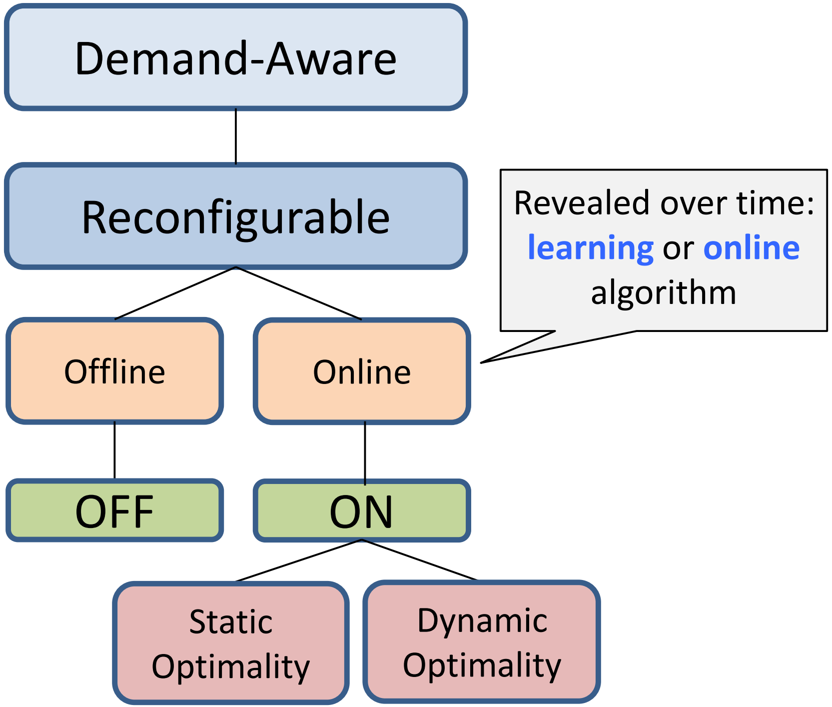
A Taxonomy: Reconfigurable Networks

Dynamic Optimality:

“***No worse than an offline*** algorithm which ***knows the sequence***!”

ρ = Cost(ON)/Cost(OFF*) is constant.

Always >=1.



Demand-Aware

Reconfigurable

Offline

Online

Revealed over time: **learning** or **online** algorithm

OFF

ON

Static Optimality

Dynamic Optimality

Awareness

Topology

Input

Algorithm

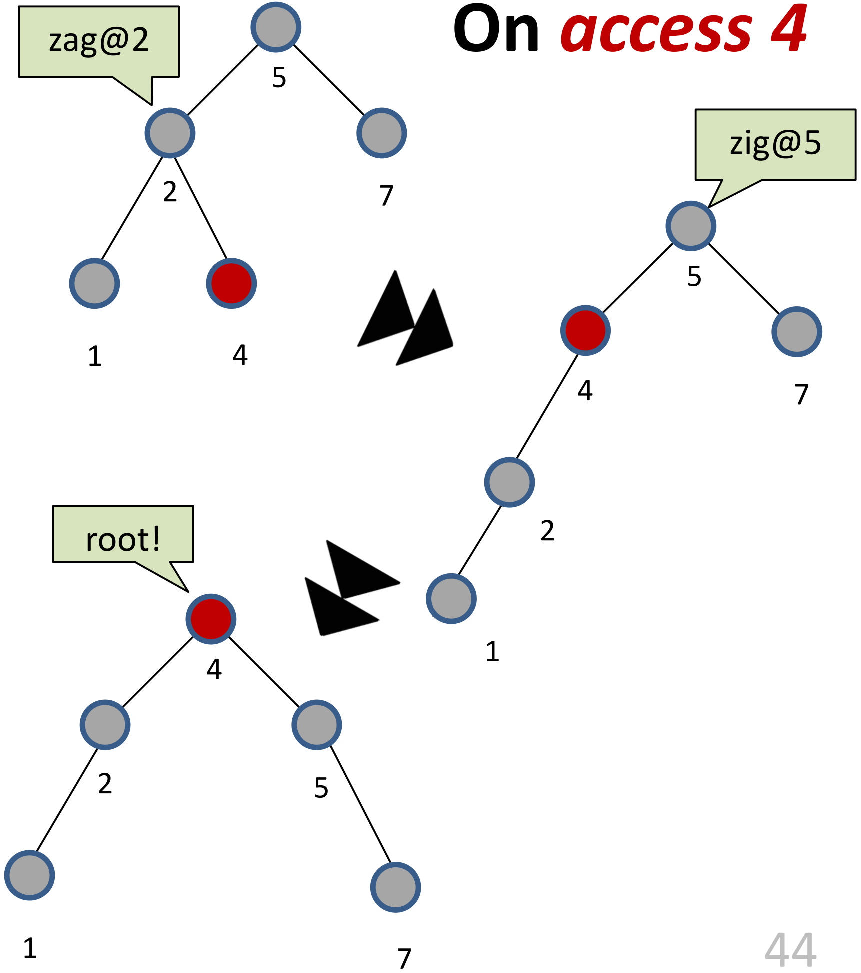
Property

How to Design SANs?

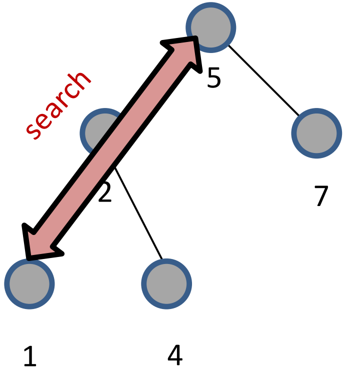
Inspiration from **self-adjusting datastructures** again!

Recall: Splay Tree

- A Binary Search Tree (**BST**)
- Inspired by "**move-to-front**": move ***to root***!
- Self-adjustment: **zig**, **zigzig**, **zigzag**
  - Maintains ***search property***
- Many nice properties
  - **Static optimality**, **working set**, (static,dynamic) **fingers**, …



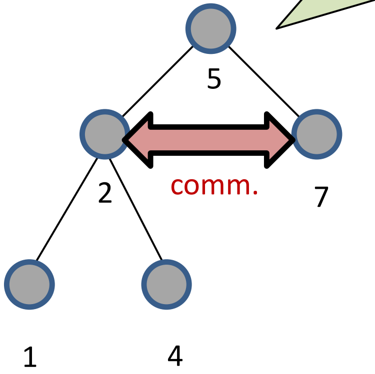
A Simple Idea:
Generalize Splay Tree To *SplayNet*



Splay Tree

vs

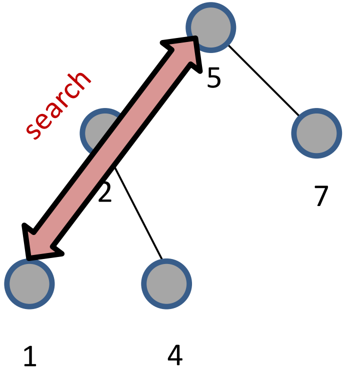
BST is nice for networks:
local (greedy) search!



SplayNet

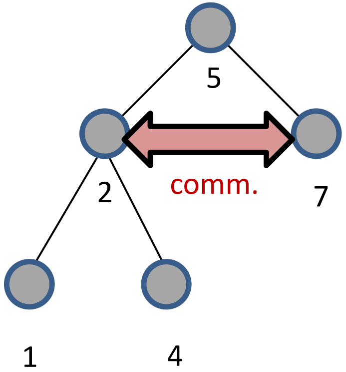
A Simple Idea:
Generalize Splay Tree To *SplayNet*

But how?



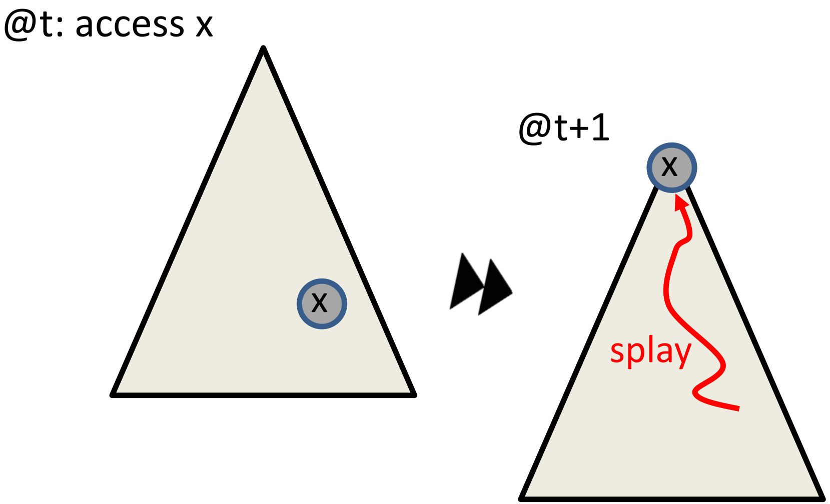
Splay Tree

vs

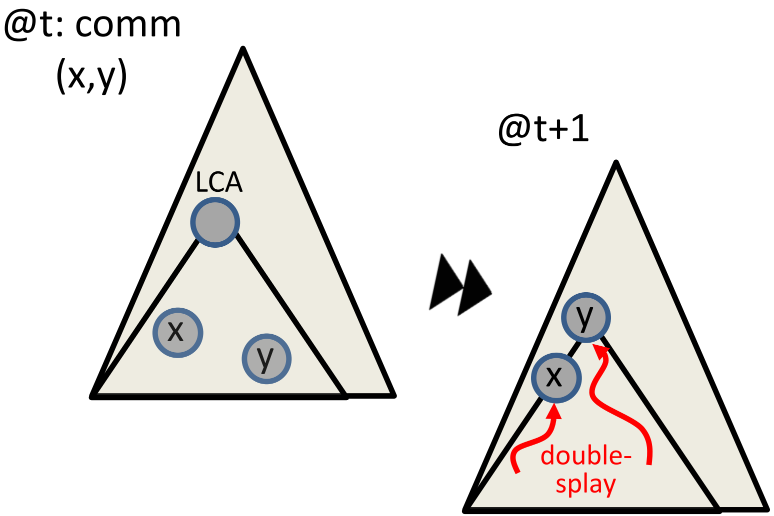


SplayNet

SplayNet: A Simple Idea

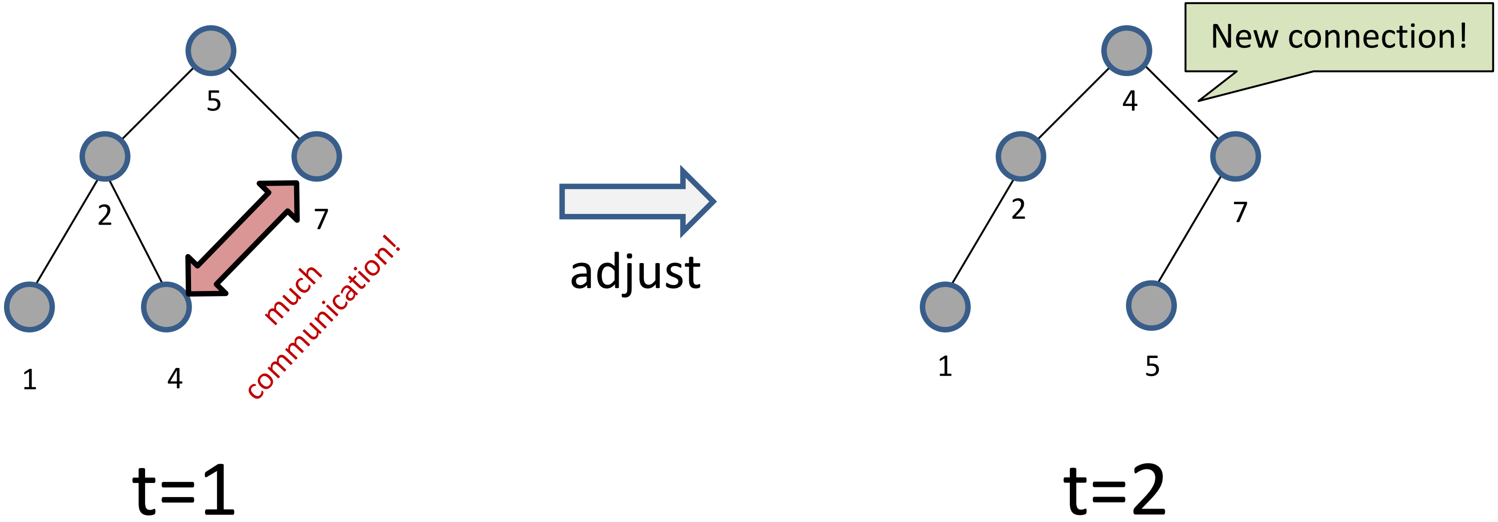


Splay Tree



SplayNet

Example



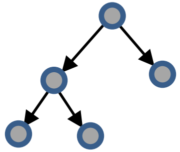
Challenges: How to **minimize reconfigurations**?
How to keep network **locally** routable?

Properties of SplayNets

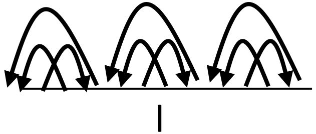
- **Statically optimal** if demand comes from a ***product distribution***
  - Product distribution: entropy equals conditional entropy, i.e., H(X)+H(Y)=H(X|Y)+H(X|Y)

- Converges to optimal static topology in
  - **Multicast scenario**: requests come from a BST as well
  - **Cluster scenario**: communication only within interval
  - **Laminated scenario** : communication is „non-crossing matching“

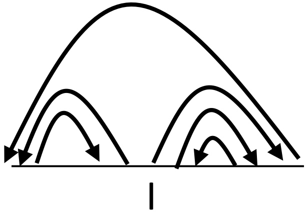
Multicast Scenario



Cluster Scenario



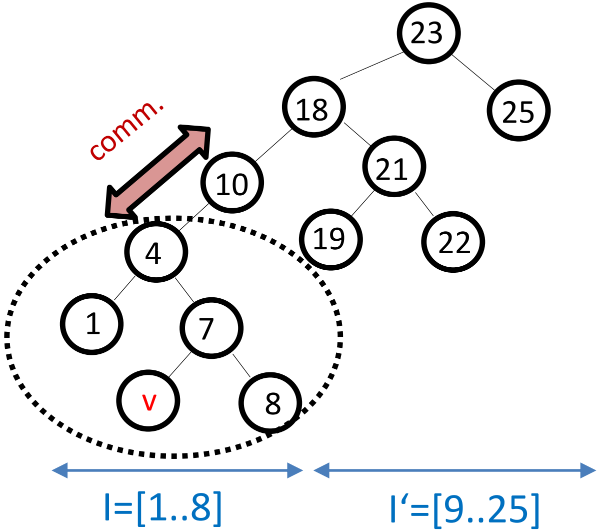
Laminated Scenario



Remark: Static SplayNet

Theorem: Optimal static SplayNet can be computed in polynomial-time (dynamic programming)

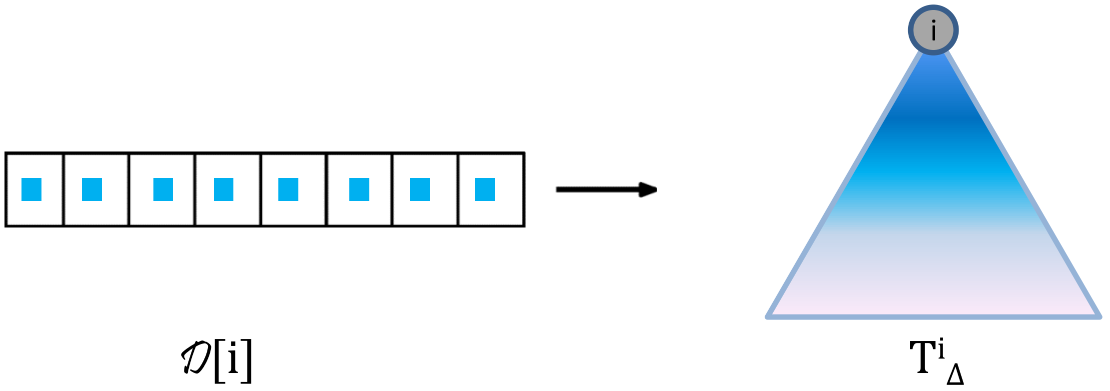
- **Unlike unordered tree?**



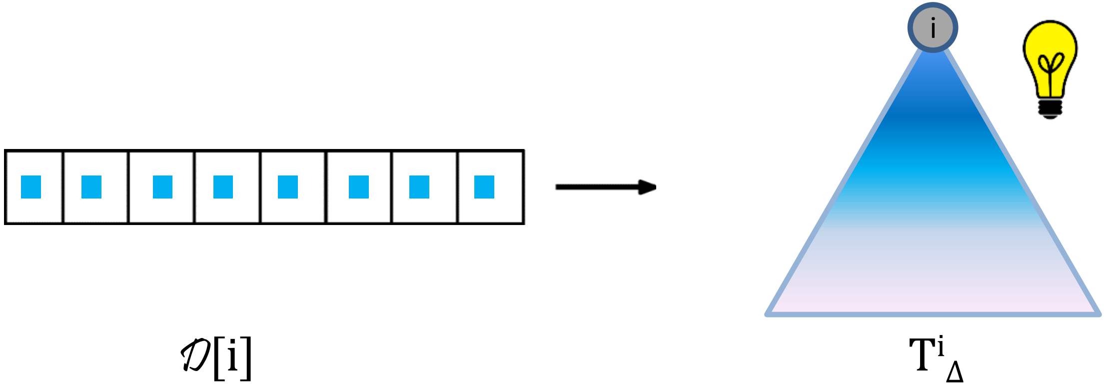
Further Reading

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.

Better Idea: Back to Ego-Trees!



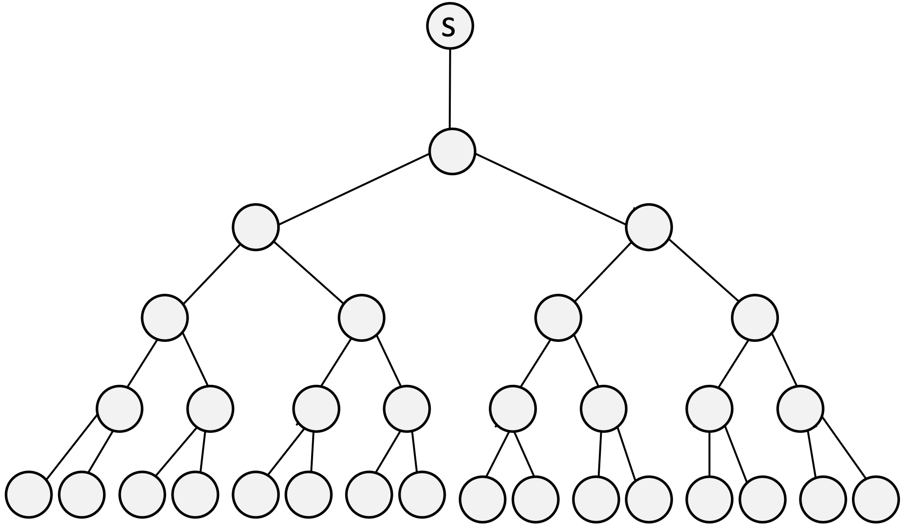
Better Idea: Back to Ego-Trees!



Idea: let each node ***adjust its ego-tree***!

A Balanced Self-Adjusting Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

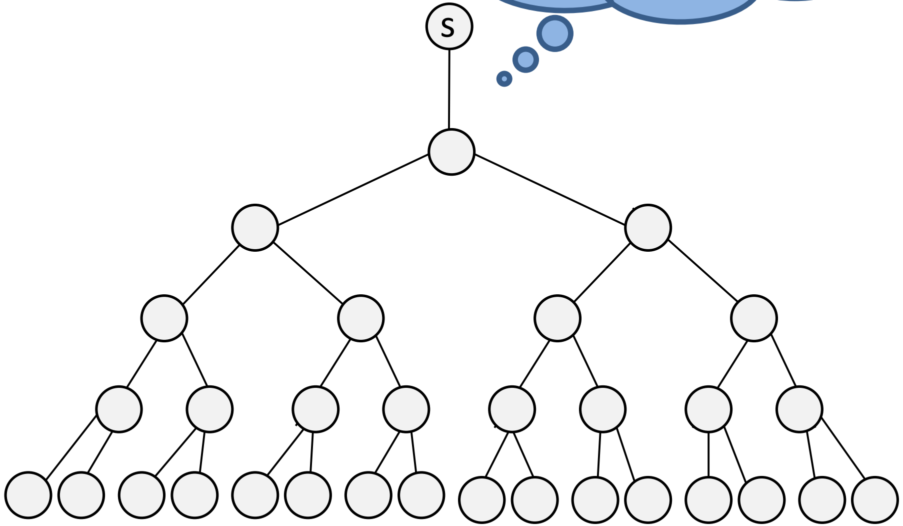


A Balanced Self-Adjusting Tree: Push-Down Tree

Unordered!

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

Equivalent: structure fix, moving nodes, not edges

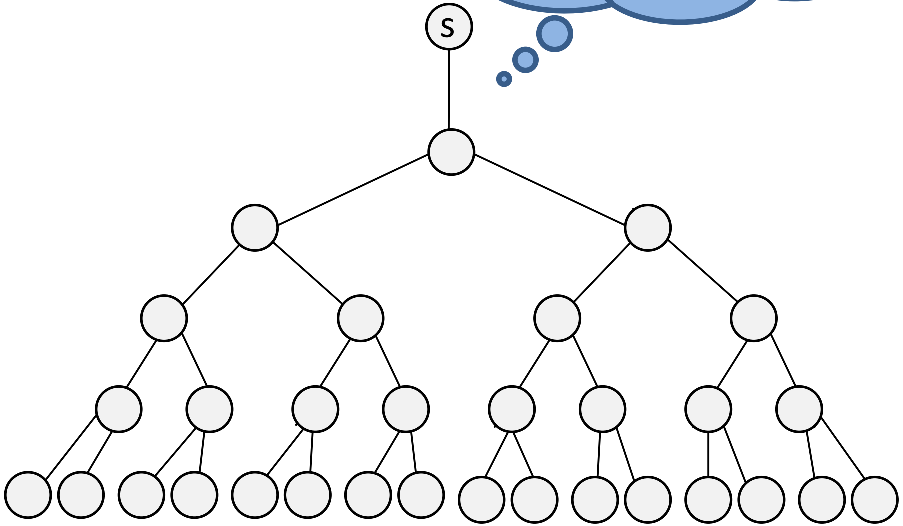


A Balanced Self-Adjusting Tree: Push-Down Tree

Unordered!

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

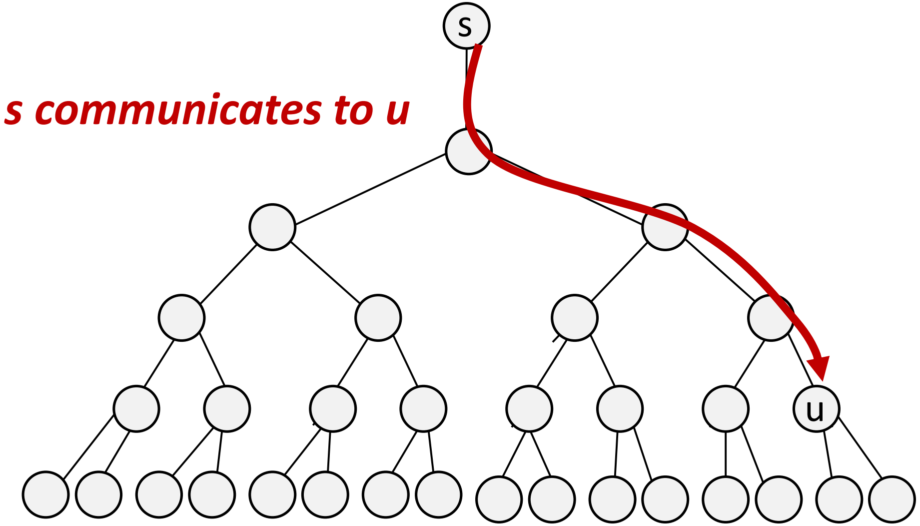
Equivalent: structure fix, moving nodes, not edges



A useful dynamic property: **Most-Recently Used (MRU)**!
Similar to **Working Set Property**: more recent communication Partners closer to source.

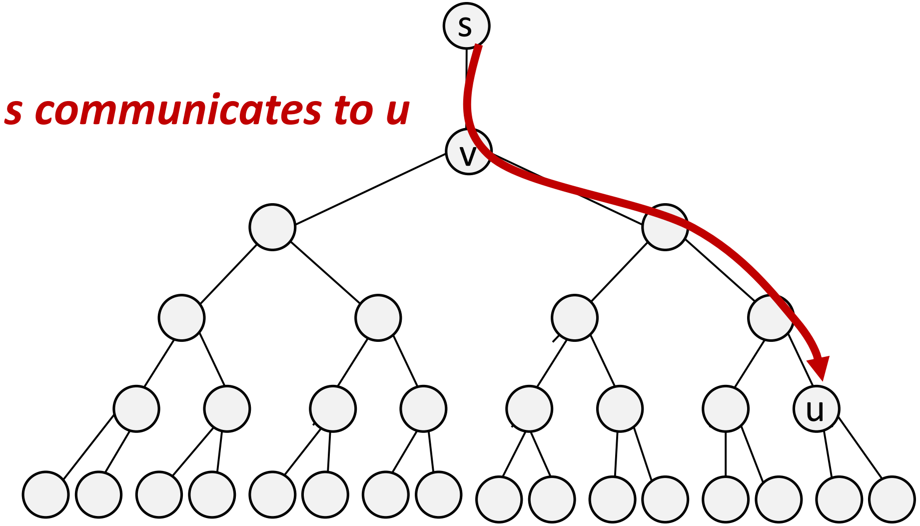
A Balanced Self-Adjusting Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



A Balanced Self-Adjusting Tree: Push-Down Tree

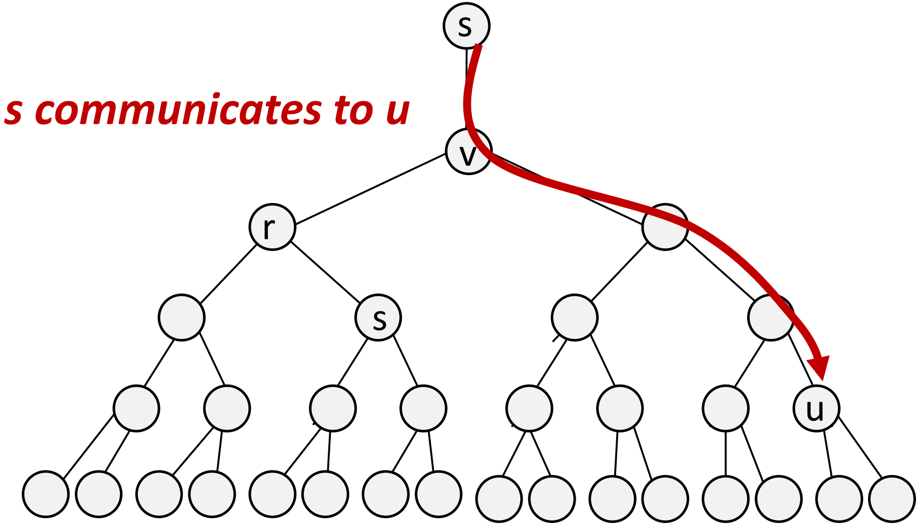
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

A Balanced Self-Adjusting Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

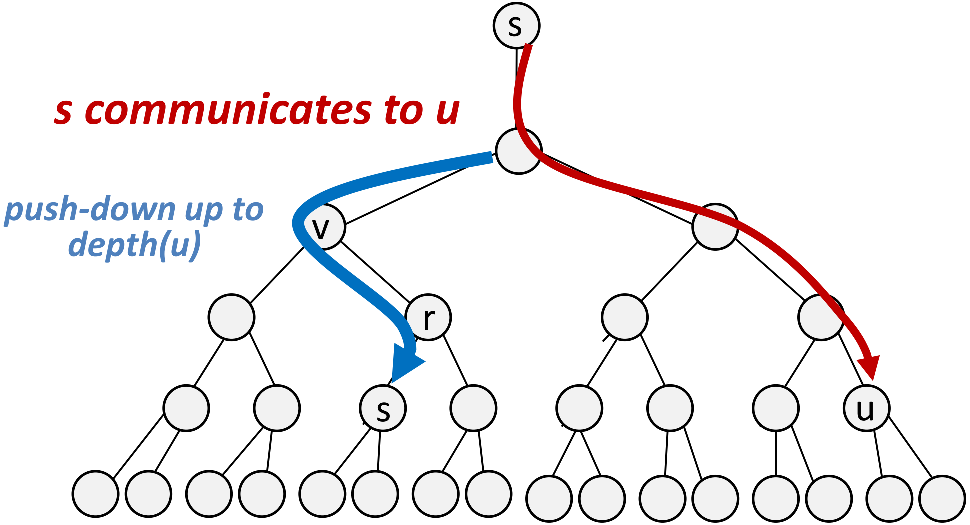
Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push“)

A Balanced Self-Adjusting Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree

- ***Dynamically optimal***

- Not ordered: requires **a map**

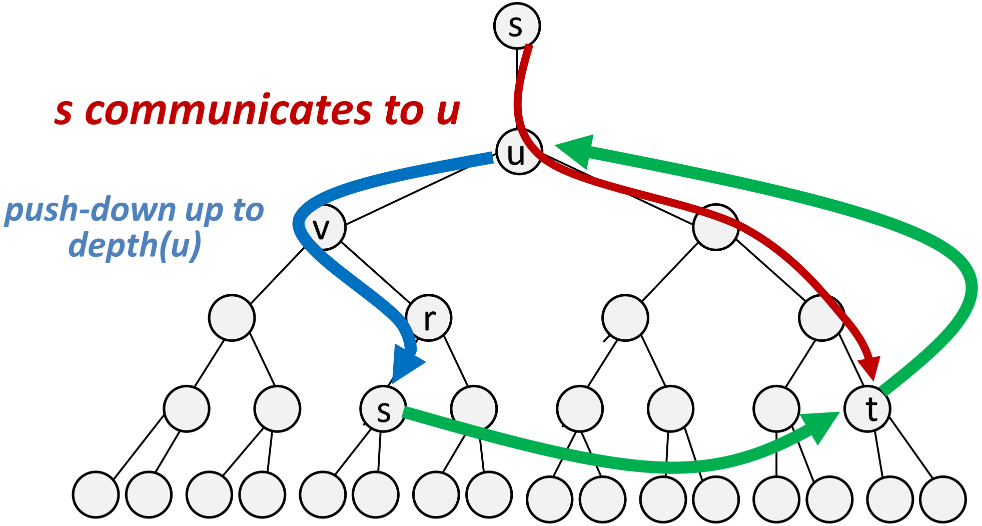


Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push“)

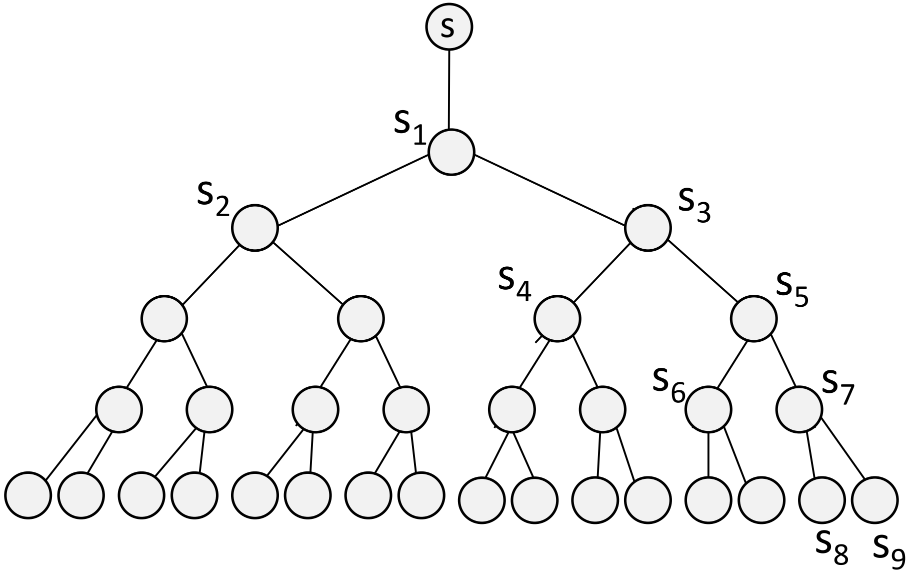
A Balanced Self-Adjusting Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



Then: promote u to available root, and t to u: at original depth!

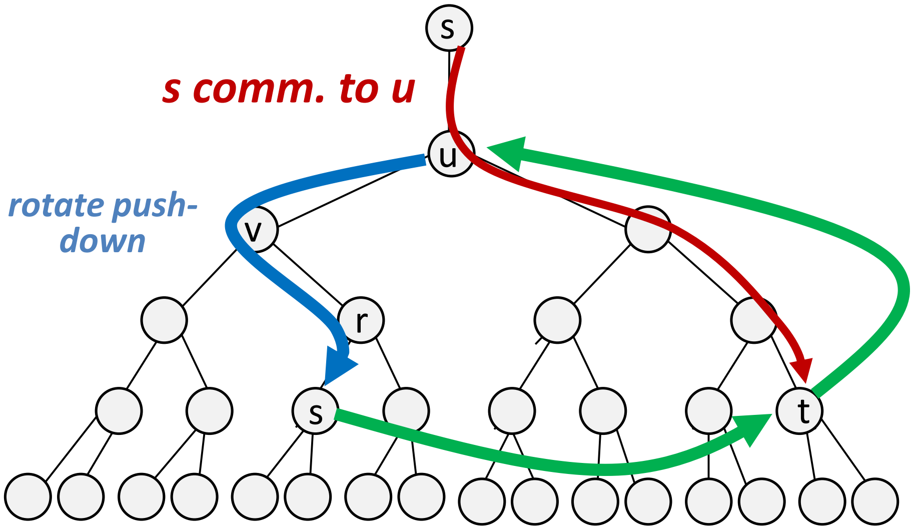
Remarks

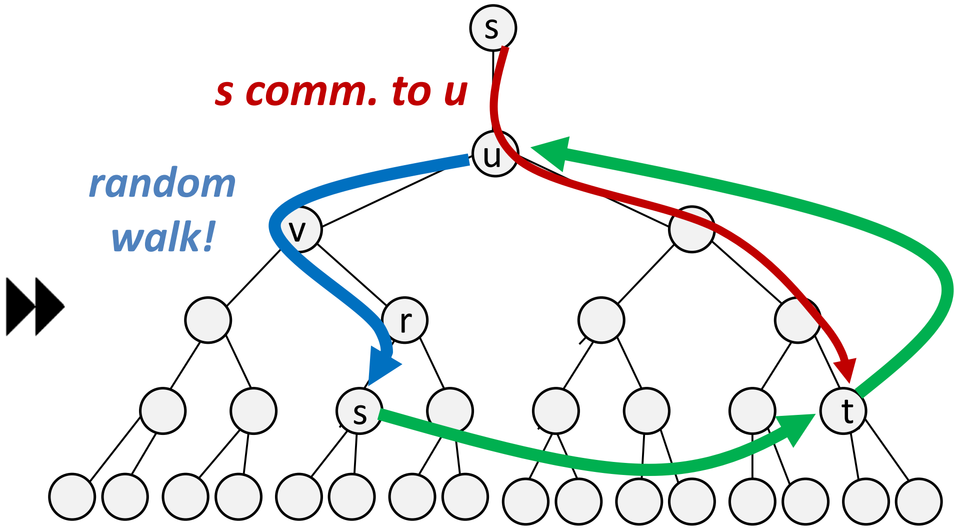


- Unfortunately, alternating push-down does ***not maintain MRU*** (working set) property

- Tree can ***degrade***, e.g.: sequence of requests from level 4,1,2,1,3,1,4,1

Solution: Random Walk





At least maintains approximate working set / MRU!

Further Reading

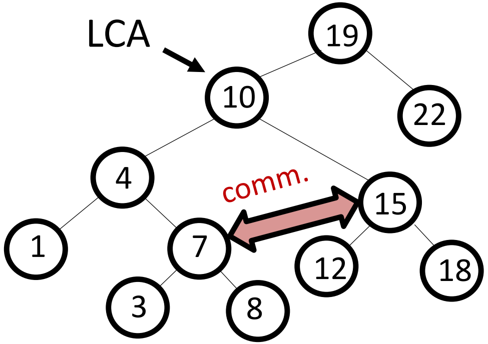
Push-Down Trees: Optimal Self-Adjusting Complete Trees
Chen Avin, Kaushik Mondal, and Stefan Schmid.
ArXiv Technical Report, July 2018.

Roadmap

- Motivation: Demand-Aware Networks ✓
- Principles of Static Demand-Aware Network Designs ✓
- Principles of Dynamic Demand-Aware Network Designs ✓
- **Principles of Decentralized Approaches**

A “Simple” Decentralized Solution: Distributed SplayNet (*DiSplayNet*)

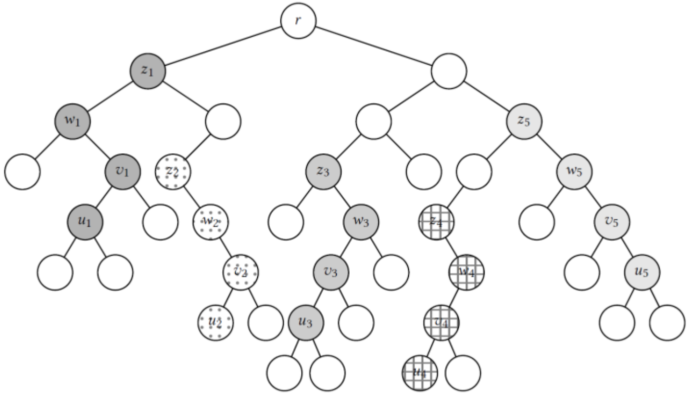
- SplayNet attractive: ordered BST supports **local routing**
  - Nodes ***maintain three ranges***: interval of left subtree, right subtree, upward
- If communicate (frequently): **double-splay** toward LCA
- Challenge: **concurrency**!
  - Access Lemma of splay trees no longer works: ***potential function*** does not „***telescope***“ anymore: a concurrently rising node may push down another rising node again

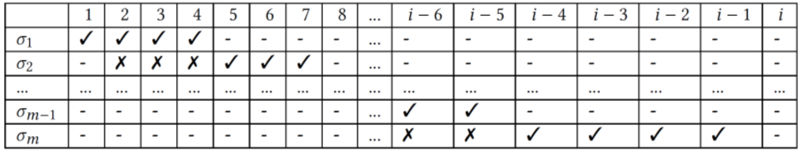


SplayNet

DiSplayNet: Challenges

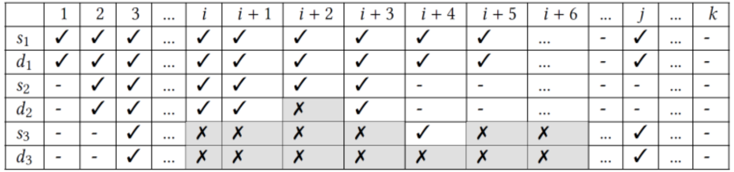
- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:





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

Sequential SplayNet: requests ***one after another***



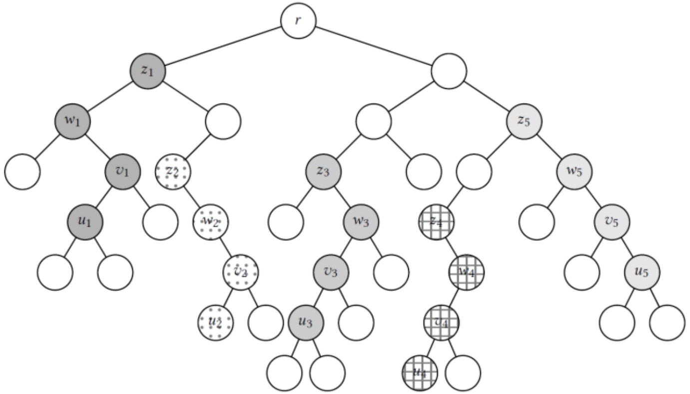
	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

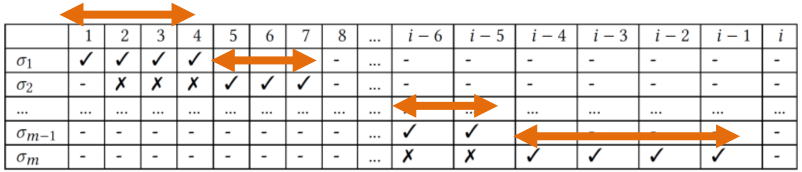
DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can "push-down" another.

DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:

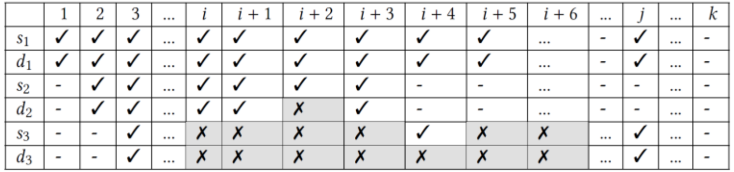
Telescopic: max potential drop





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

Sequential SplayNet: requests ***one after another***



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

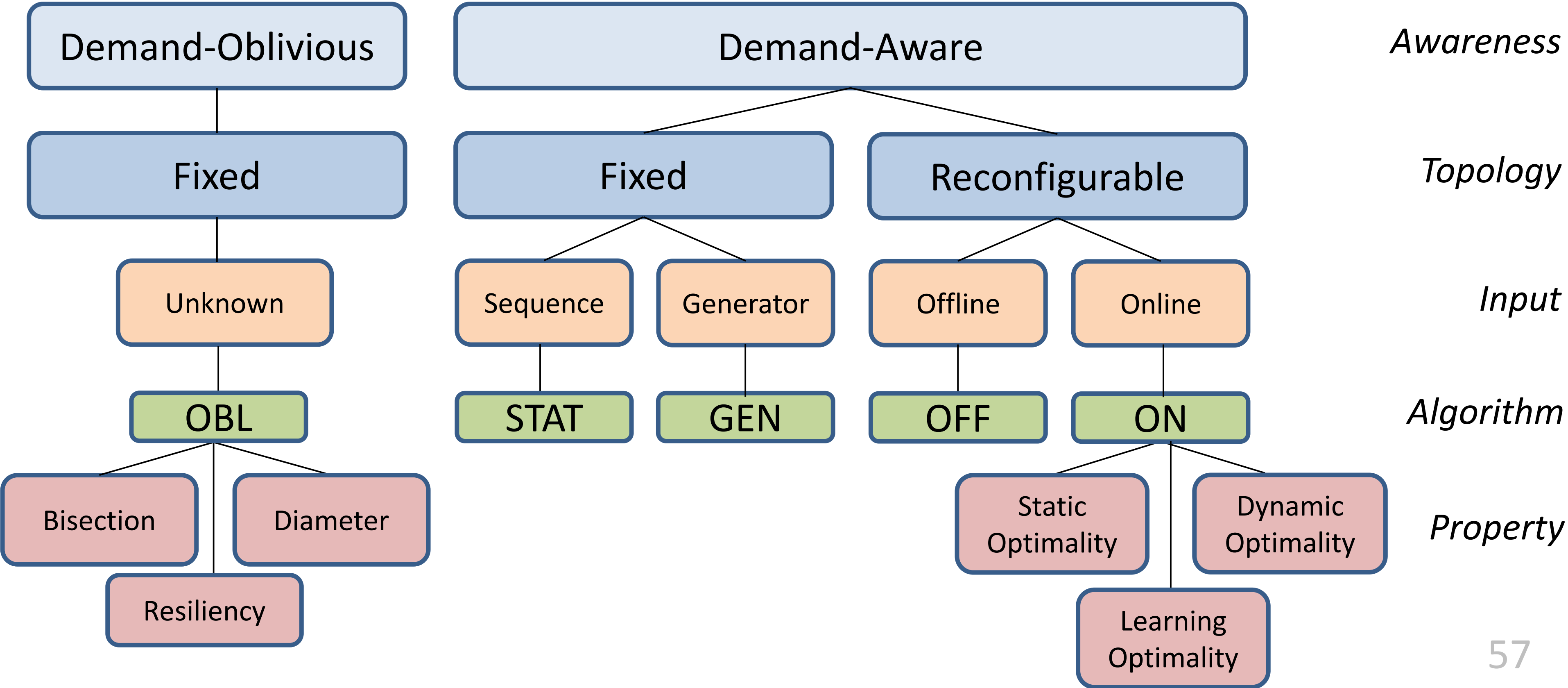
DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can "push-down" another.

Further Reading

Brief Announcement: Distributed SplayNets
Bruna Peres, Olga Goussevskaia, Stefan Schmid, and Chen Avin.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

Uncharted Landscape!

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



Conclusion

- Reconfigurable switches: ***Yoga for Networks?***
- New metrics needed: e.g., entropy?
- New algorithms needed: static, offline and online!
- ***Let’s chat!***

Thank you!
Question?

Further Reading

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks
Chen Avin and Stefan Schmid.
SIGCOMM CCR, October 2018.
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.
Push-Down Trees: Optimal Self-Adjusting Complete Trees
Chen Avin, Kaushik Mondal, and Stefan Schmid.
ArXiv Technical Report, July 2018.
Online Balanced Repartitioning
Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid.
30th International Symposium on Distributed Computing (**DISC**), Paris, France, September 2016.
rDAN: Toward Robust Demand-Aware Network Designs
Chen Avin, Alexandr Hercules, Andreas Loukas, and Stefan Schmid.
Information Processing Letters (**IPL**), Elsevier, 2018.
SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.
Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.
Charting the Complexity Landscape of Virtual Network Embeddings
Matthias Rost and Stefan Schmid. **IFIP Networking**, Zurich, Switzerland, May 2018.