Principles of Distributed Network Design

Stefan Schmid *et al.*, mainly Chen Avin (BGU, Israel)

GIF German-Israeli Foundation for Scientific Research and Development

Flexible Distributed Systems:
A Great Time To Be a Researcher!

Flexible distributed systems: virtualized and programmable

Algorithms: opportunities and challanges

Research & Innovation

Rhone and Arve Rivers, Switzerland

Credits: George Varghese.

The new frontier!

Routing, e.g., SDN

Flexible **placement**, e.g., VMs

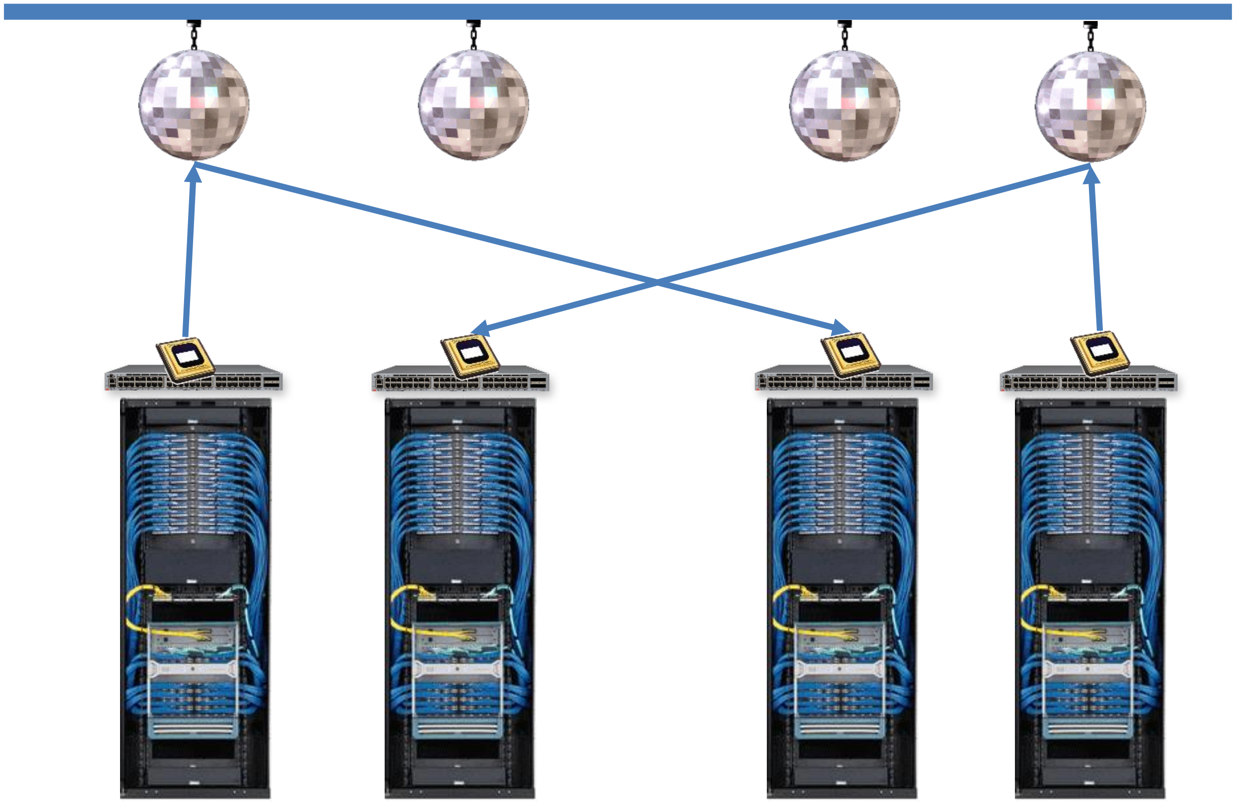
Flexible **migration**, e.g., VMs

Topology **reconfiguration**

Flexibility of communication networks

Technological Motivation

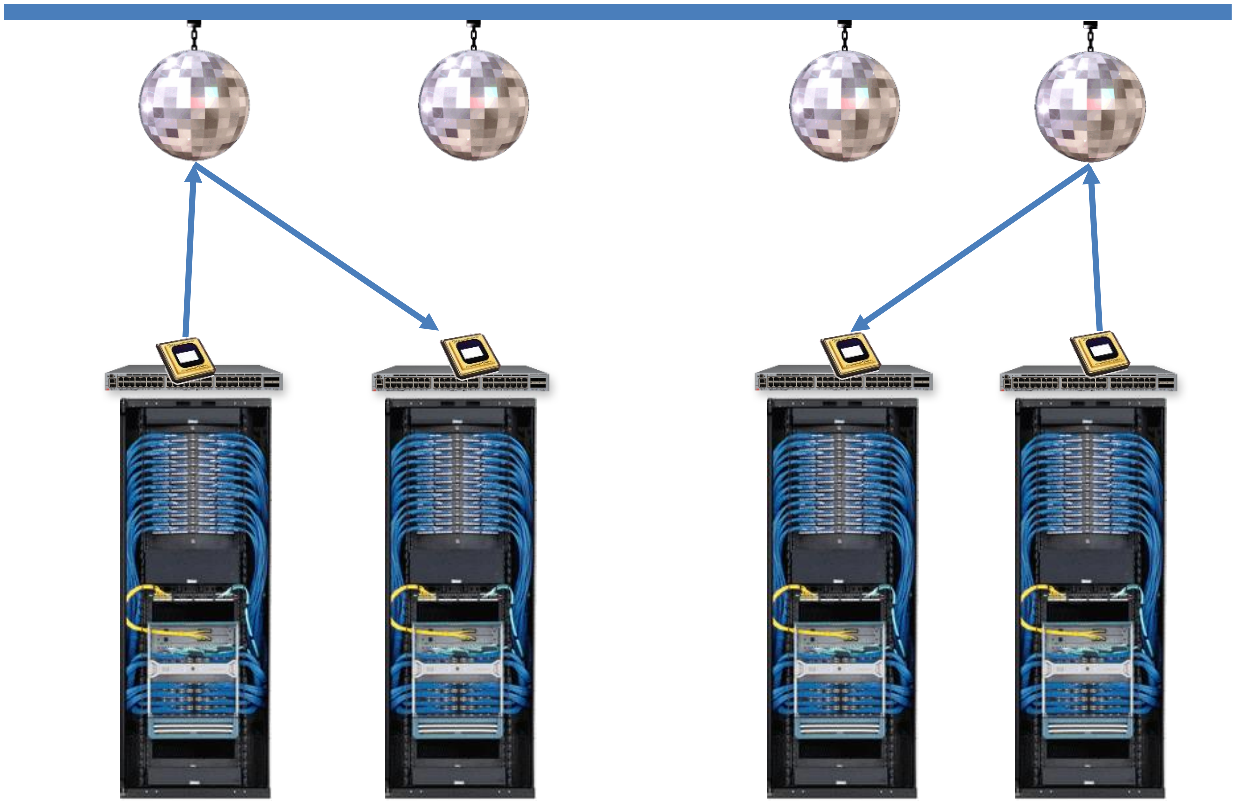
Example: Free-Space Optics (*ProjecToR*)



t=1

Example: Free-Space Optics (*ProjecToR*)

t=2



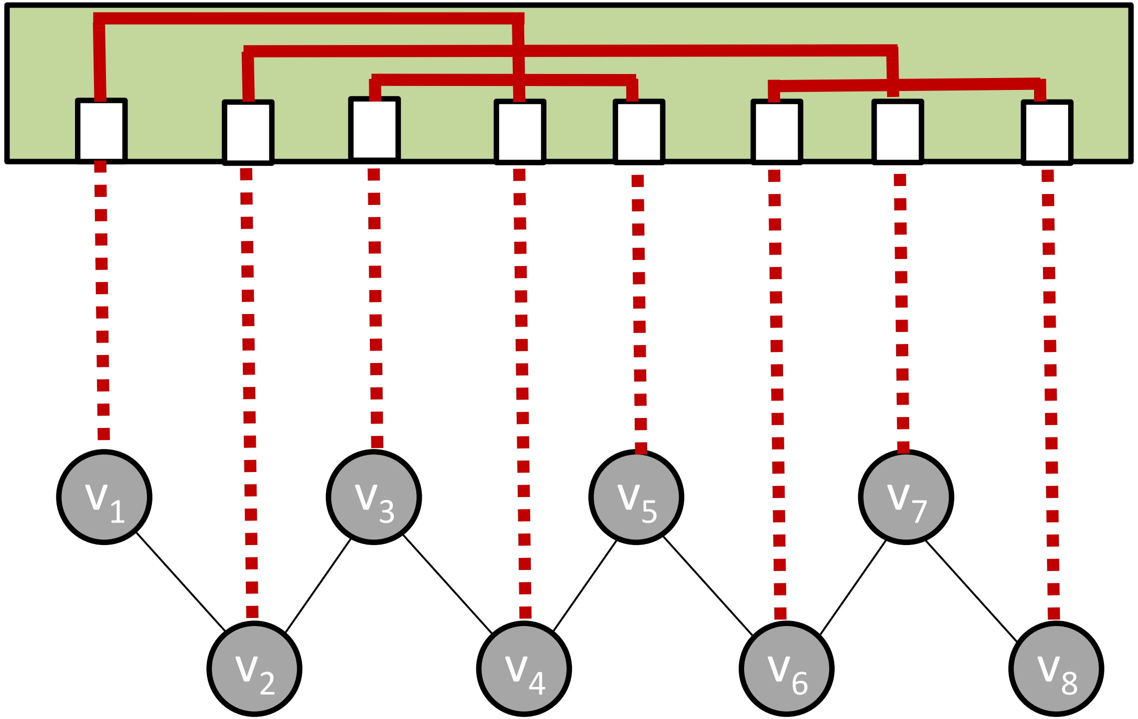
Example: Reconfigurable Optical Switches (*Helios*, *c-Through*, etc.)

Matching!

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

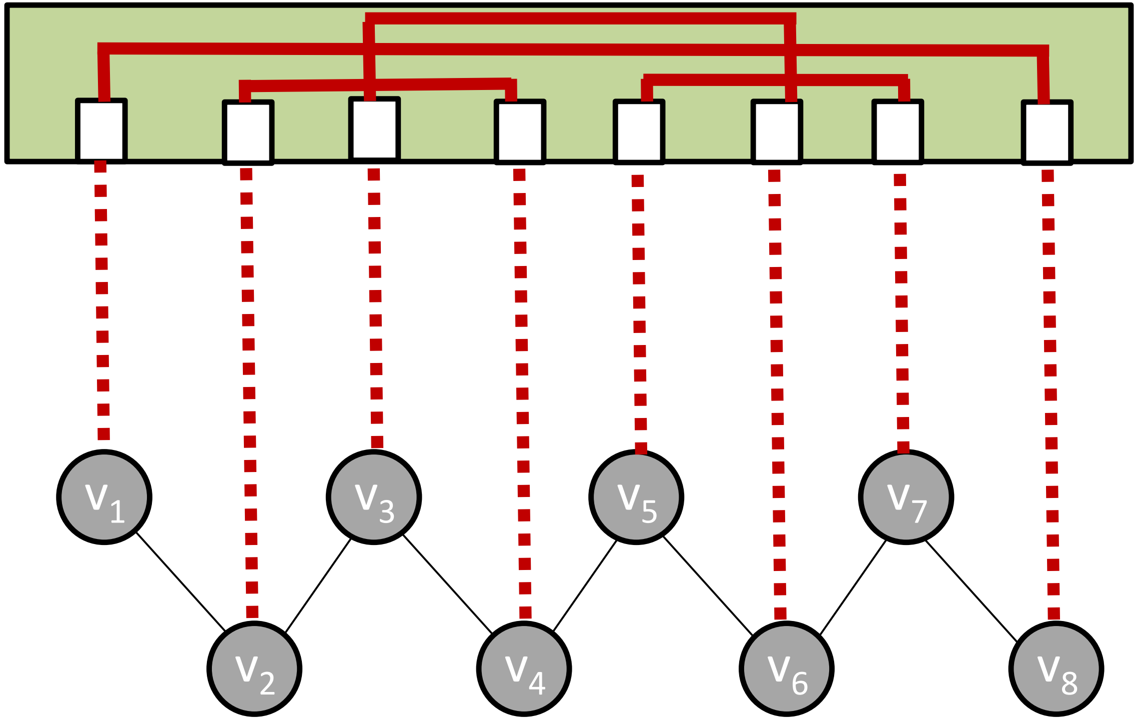
t=1

Static topology:
electric



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

Matching!



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

t=2

Static topology:
electric

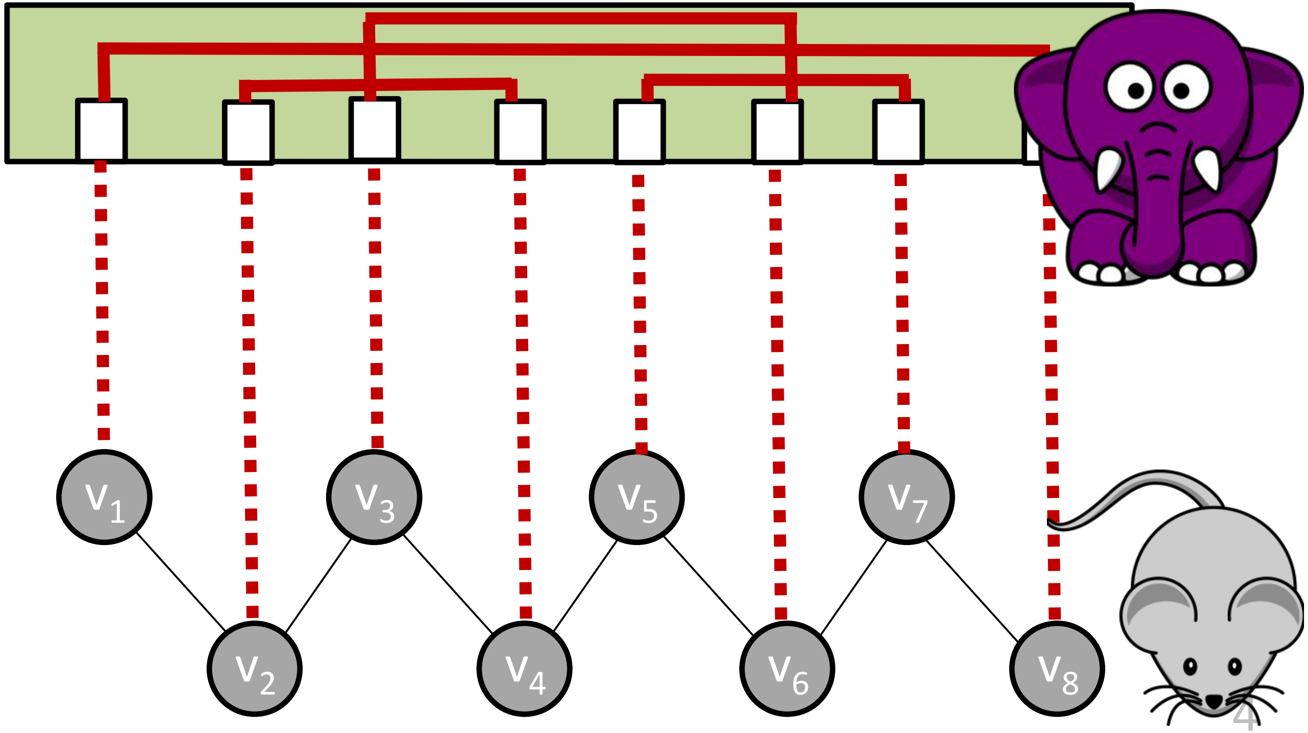
Example: Reconfigurable Optical Switches (*Helios*, *c-Through*, etc.)

Matching!

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

t=2

Static topology:
electric



Further Reading

Free-Space Optics

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

Optical Circuit Switches

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

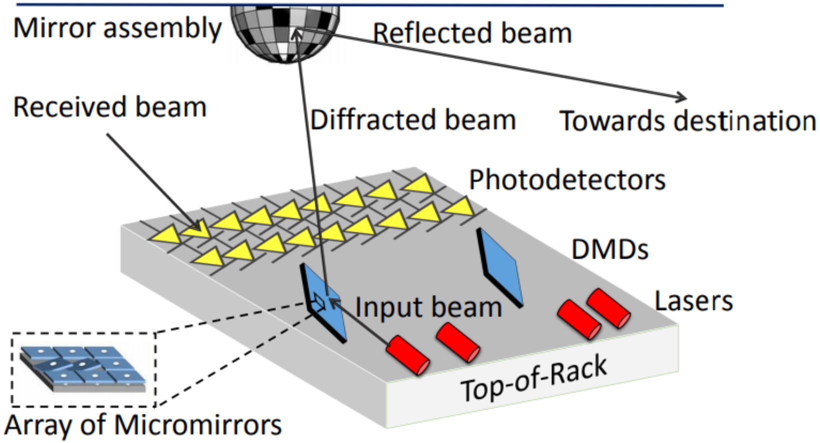
Movable Antennas

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

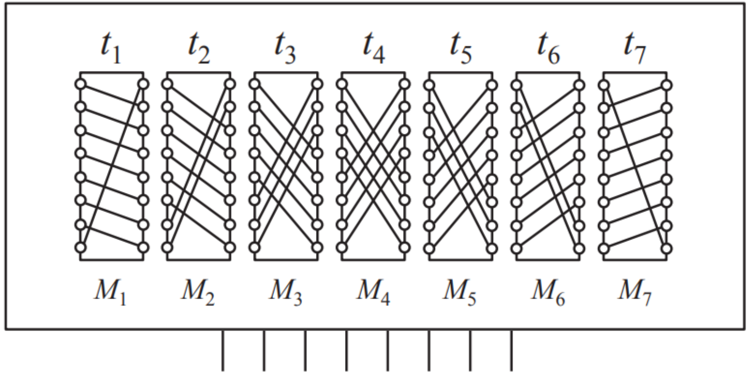
60GHz Wireless Communication

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

Etc.!

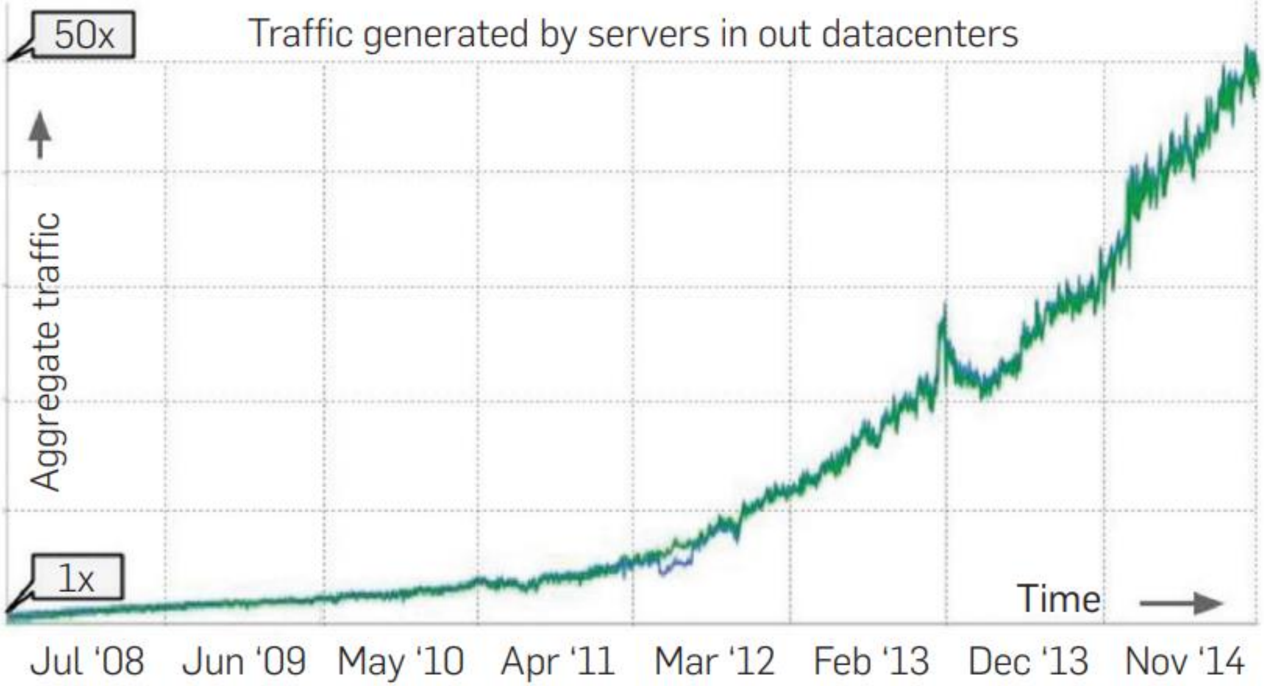


Rotor switch



Empirical Motivation

Data-Centric Applications: Growing Traffic…



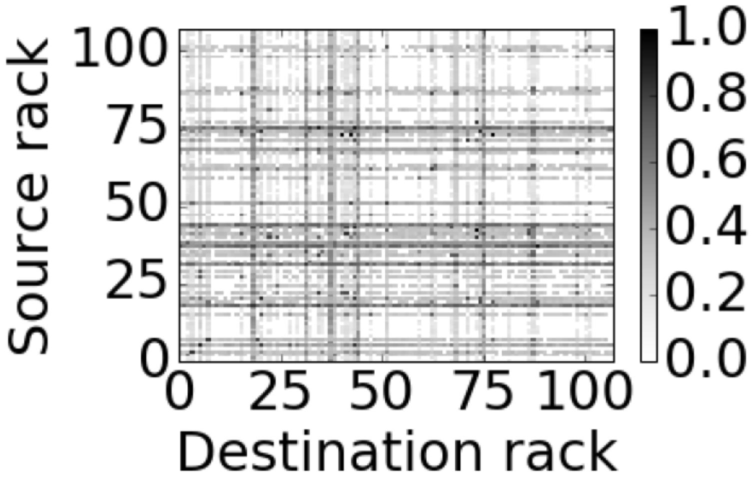
Aggregate server traffic in
Google's datacenter fleet

Source: Jupiter Rising.

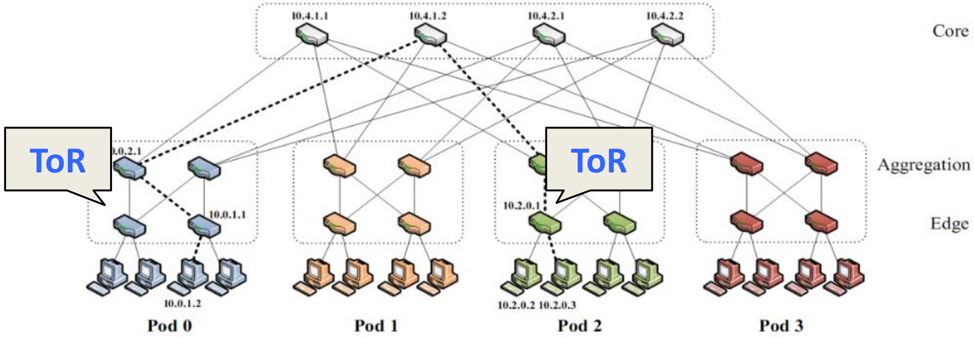
SIGCOMM 2015.

… But Much Structure!

Heatmap rack-to-rack traffic:



Clos topology and ToR switches:



ProjecToR @ SIGCOMM 2016

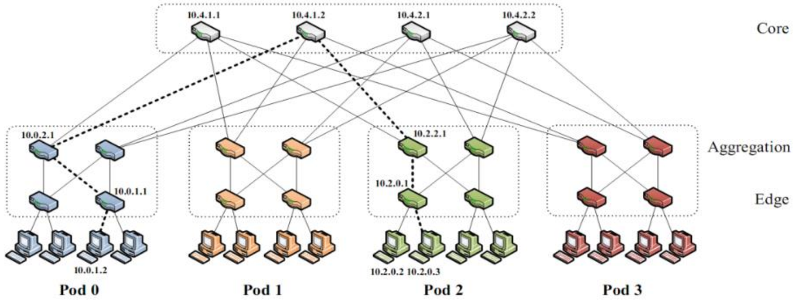
The Idea 💡

Traditional Networks

Demand-Oblivious

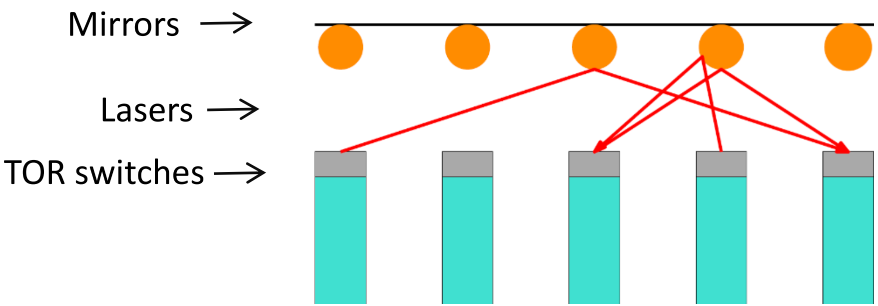
Fixed

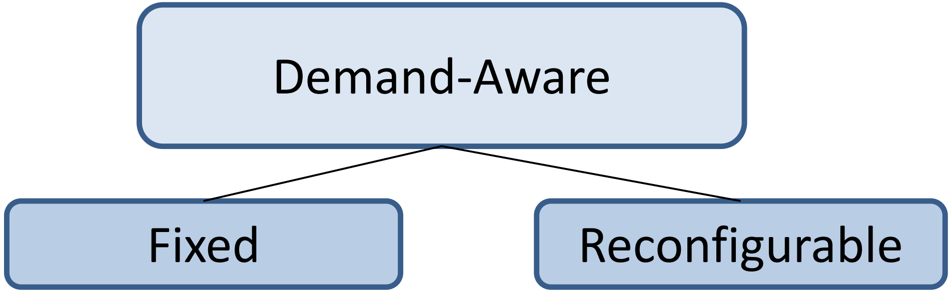
- Usually optimized for the "worst-case" (**all-to-all** communication)
- Example, fat-tree topologies: provide **full bisection bandwidth**
- Lower bounds and hard **trade-offs**, e.g., degree vs diameter



Demand-Aware Networks

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





Demand-Aware

Fixed

Reconfigurable

Roadmap

- Vision and Motivation
- An analogy: coding and datastructures
- Principles of Demand-Aware Network (DAN) Designs
- Principles of Self-Adjusting Network (SAN) Designs
- Principles of Decentralized Approaches

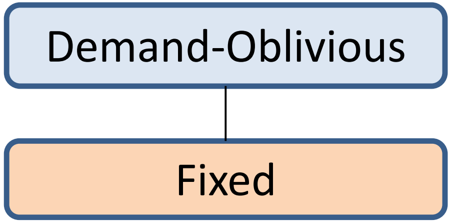
Roadmap

- Vision and Motivation ✓
- **An analogy: coding and datastructures**
- Principles of Demand-Aware Network (DAN) Designs
- Principles of Self-Adjusting Network (SAN) Designs
- Principles of Decentralized Approaches

Analogous to *Datastructures*: Oblivious…

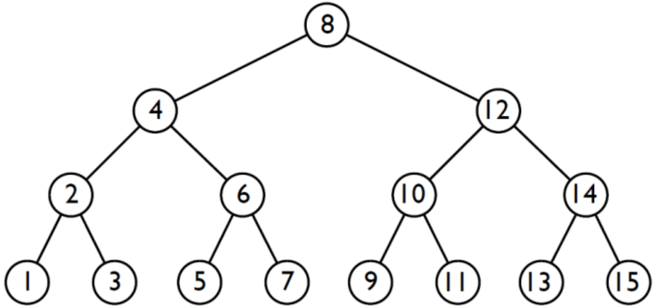
- Traditional, **fixed** BSTs do not rely on any assumptions on the demand

- Optimize for the **worst-case**



Demand-Oblivious

Fixed



Analogous to *Datastructures*: Oblivious…

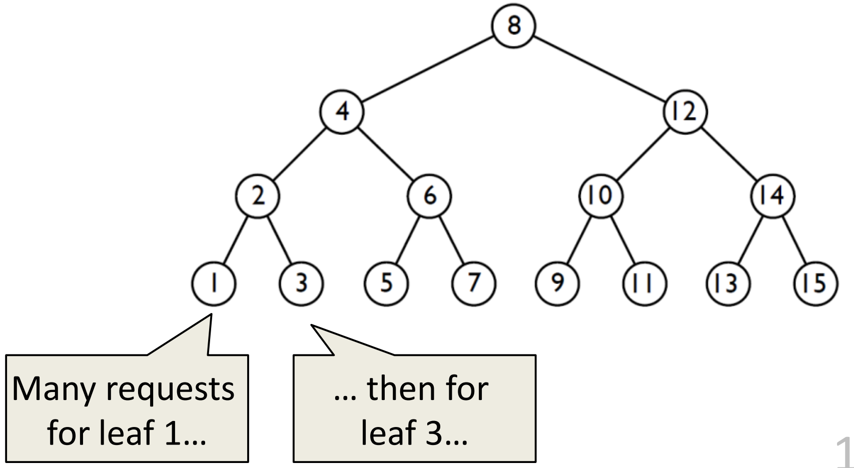
- Traditional, **fixed** BSTs do not rely on any assumptions on the demand
- Optimize for the **worst-case**
- Example **demand**:

  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

Demand-Oblivious

Fixed



Many requests for leaf 1…

… then for leaf 3…

Analogous to *Datastructures*: Oblivious…

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

  1,…,1,3,…,3,5,…,5,7,…,7

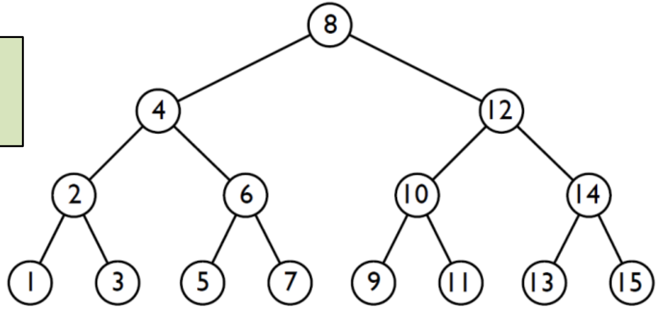
  ⟷ ⟷ ⟷ ⟷

  many many many many

  Amortized cost corresponds to ***max entropy of demand***!
- Items stored at ***O(log n)*** from the root, **uniformly** and **independently** of their frequency

Demand-Oblivious

Fixed

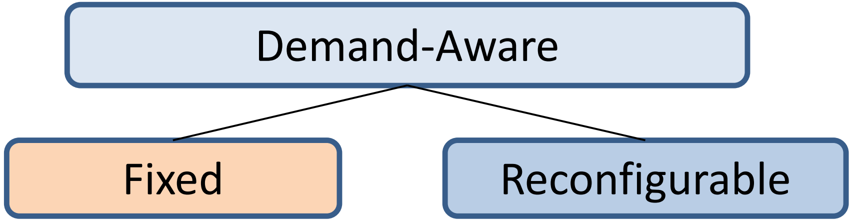


Many requests for leaf 1…

… 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



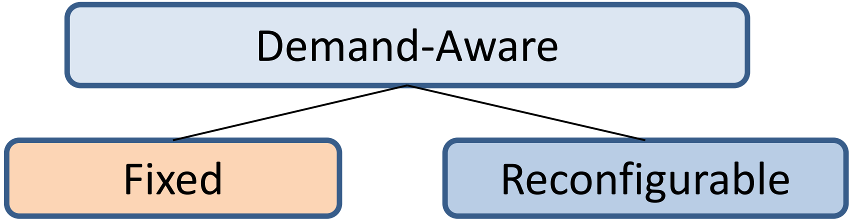
Demand-Aware

Fixed

Reconfigurable

… Demand-Aware …

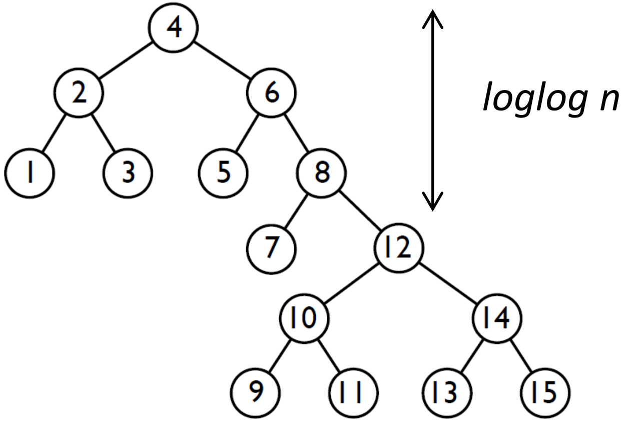
- **Demand-aware fixed** BSTs can take advantage of ***spatial locality*** of the demand
- E.g.: place frequently accessed elements close to the root
- E.g., **Knuth/Mehlhorn/Tarjan** trees
- Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
  - Amortized cost ***O(loglog n)***



Demand-Aware

Fixed

Reconfigurable



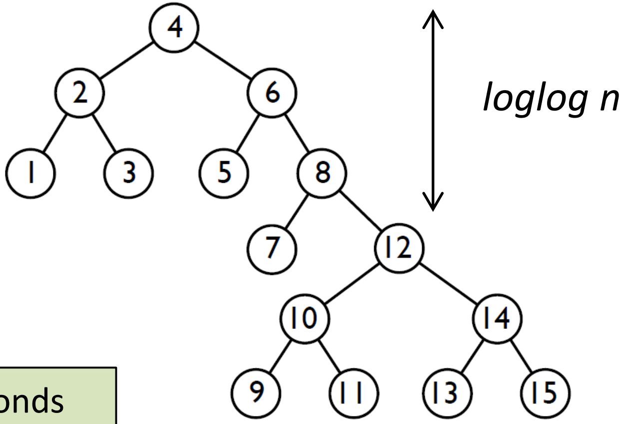
… Demand-Aware …

- **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)***

Demand-Aware

Fixed

Reconfigurable



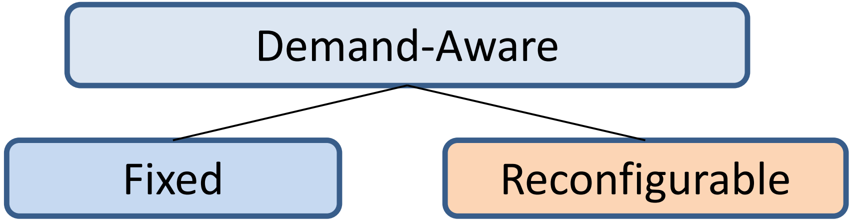
Amortized cost corresponds to ***empirical entropy of demand***!

… 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)

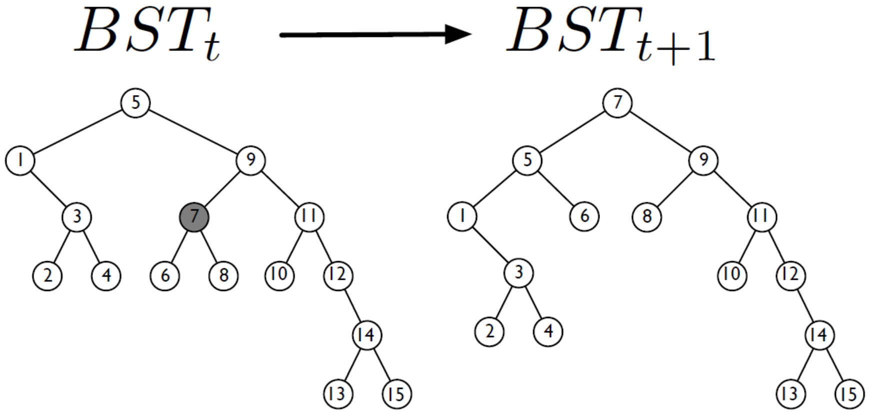
- Self-adjusting BSTs e.g., useful for implementing ***caches*** or garbage collection



Demand-Aware

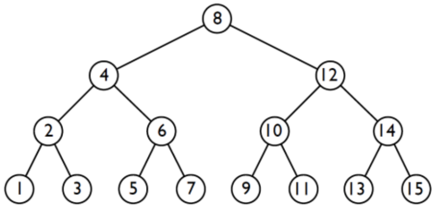
Fixed

Reconfigurable



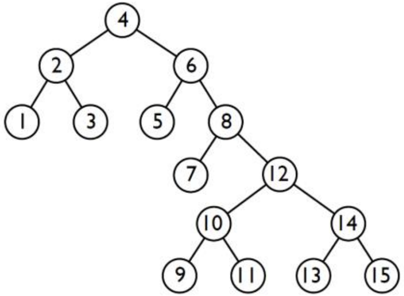
Datastructures

Oblivious



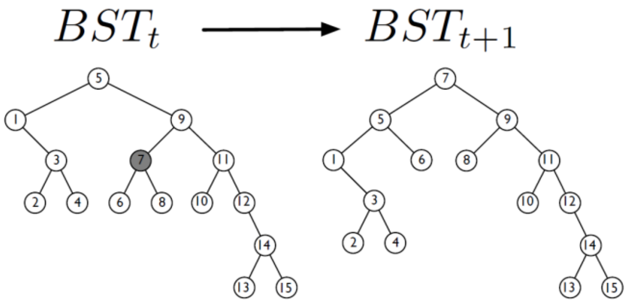
Lookup ***O(log n)***

Demand-Aware



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

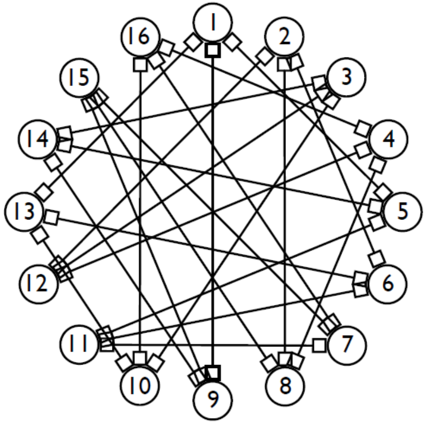
Self-Adjusting



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

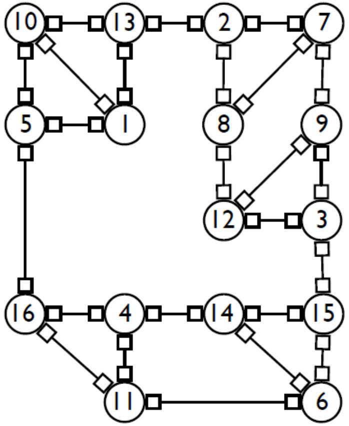
Analogously for Networks

Oblivious



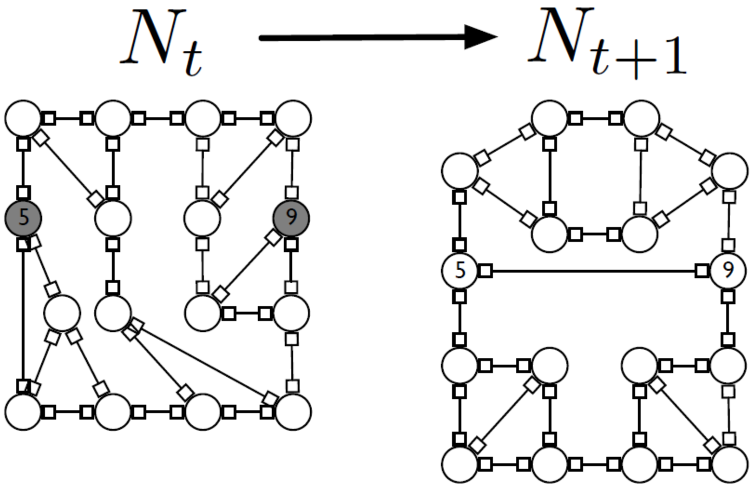
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



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

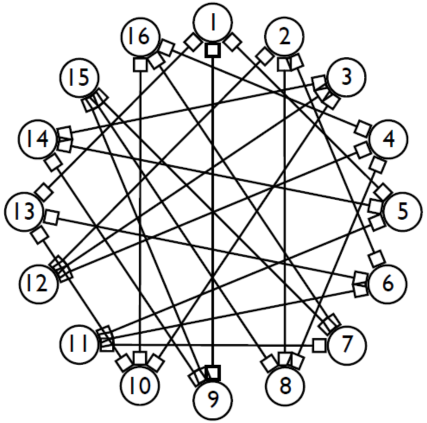
SAN



Exploit **temporal locality** as well

Analogously for Networks

Oblivious

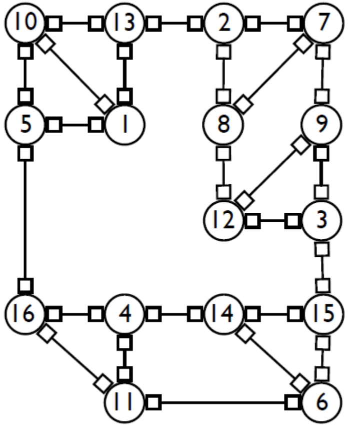


Const degree

(e.g., **expander**):

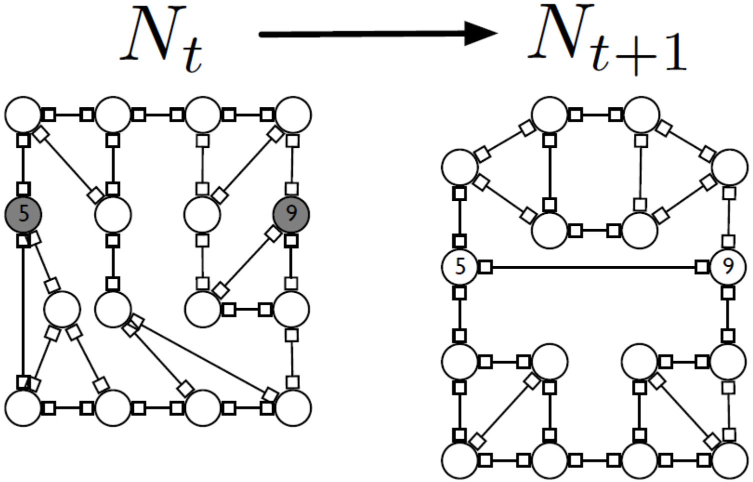
route lengths ***O(log n)***

DAN



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

SAN



Exploit **temporal locality** as well

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

How useful are DANs/SANs?

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

Roadmap

- Vision and Motivation ✓
- **An analogy: coding and datastructures** ✓
- **Principles of Demand-Aware Network (DAN) Designs**
- Principles of Self-Adjusting Network (SAN) Designs
- Principles of Decentralized Approaches

The DAN Design Problem

Input: Workload

Destinations

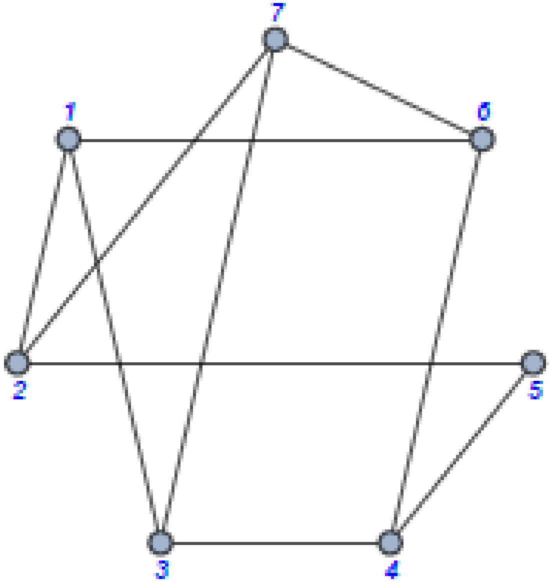
Sources

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

Demand matrix: joint distribution

design

Output: DAN



... of constant degree (**scalability**)

The DAN Design Problem

Input: Workload

Destinations

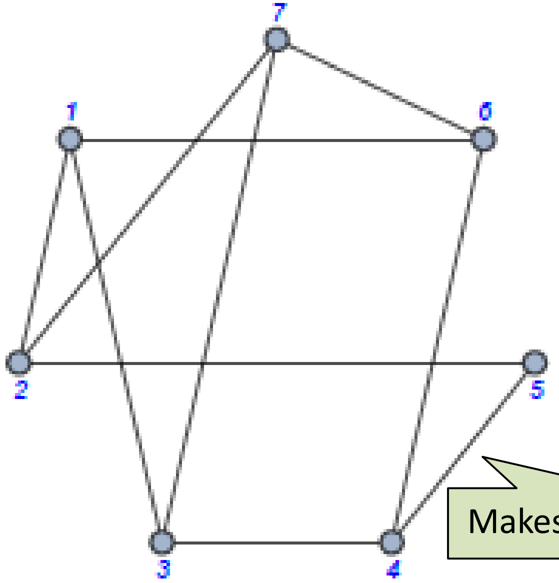
Sources

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

Much from 4 to 5.

design

Output: DAN



Makes sense to add link!

Demand matrix: joint distribution

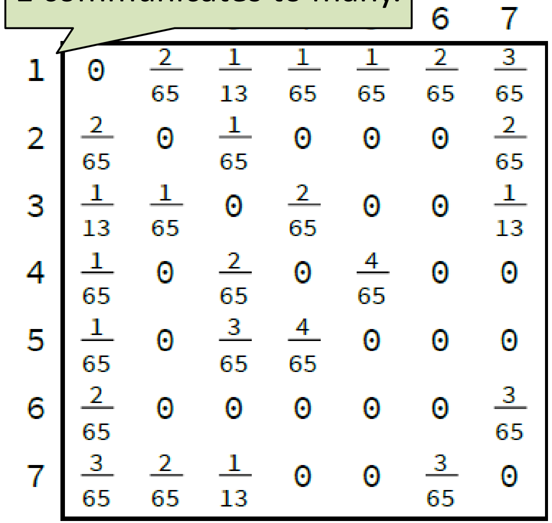
… of constant degree (**scalability**)

The DAN Design Problem

Input: Workload

Destinations

1 communicates to many.



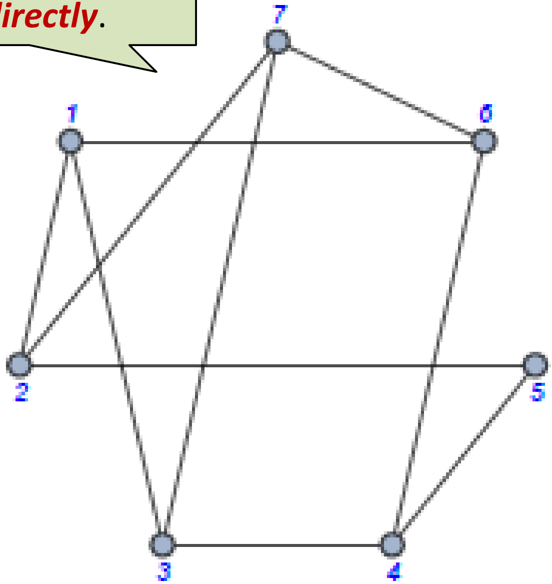
$$\begin{array}{c|ccccccc} & & & & & & 6 & 7 \\ \hline 1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\ 2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\ 3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\ 4 &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

Sources

design

Output: DAN

Bounded degree: route to 7 *indirectly*.



Demand matrix: joint distribution

… of constant degree (**scalability**)

The DAN Design Problem

Input: Workload

Destinations

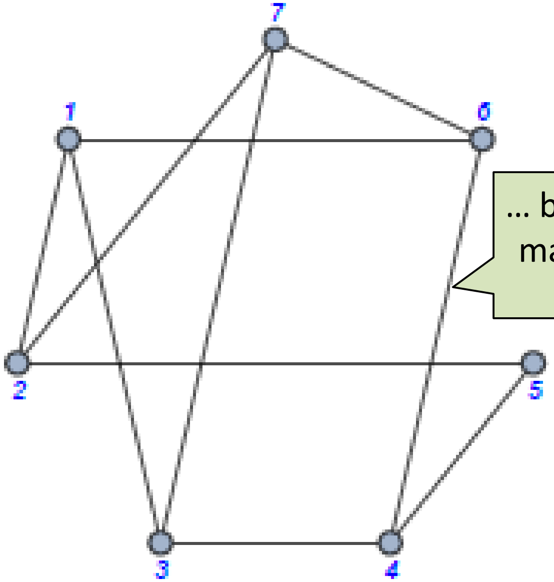
Sources

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

4 and 6 don’t communicate…

design

Output: DAN



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

Demand matrix: joint distribution

… of constant degree (**scalability**)

Case Study: Expected Route Length

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

More Formally: DAN Design Problem

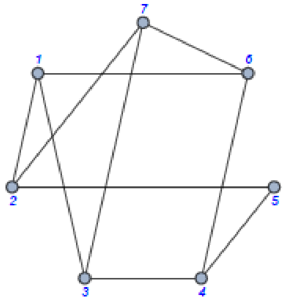
Input:

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

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

Output:

N: DAN



Bounded degree Δ=3

Expected Path Length (EPL): Basic measure of efficiency

$$\mathrm{EPL}(\mathcal{D},N) = E_{\mathcal{D}}[d_N(\cdot,\cdot)] = \sum_{(u,v)\in\mathcal{D}} p(u,v)\cdot d_N(u,v)$$

Path length ***on DAN***.

Frequency

The Goal: Bounded Network Design (BND)

- **Inputs**: Communication distribution $\mathcal{D}[p(i,j)]_{nxn}$ and a maximum degree Δ.

- **Output**: A Demand Aware Network $N \in \mathcal{N}_\Delta$ s.t.

$$\mathrm{BND}(\mathcal{D}, \Delta) = \min_{N \in \mathcal{N}_\Delta} \mathrm{EPL}(\mathcal{D}, N)$$

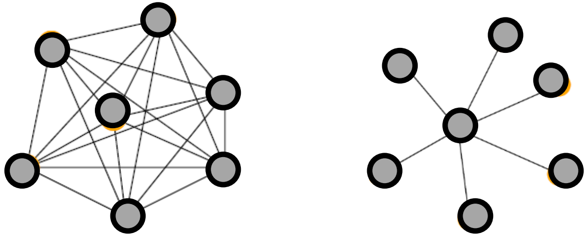
Belong to some graph family:
bounded-degree, but e.g. also local routability!

Examples

- What about BND for $\Delta = n$?

Examples

- What about BND for Δ = n?
- Easy: e.g., **clique** and **star** have constant EPL but unbounded degree.

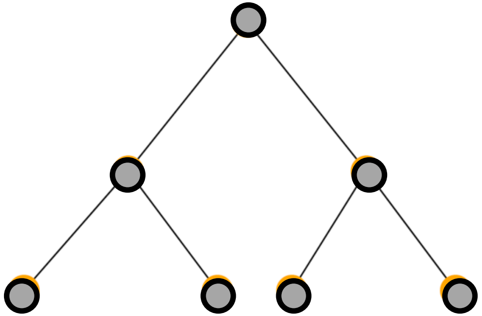


Some Insights

- What about $\Delta = 3$?

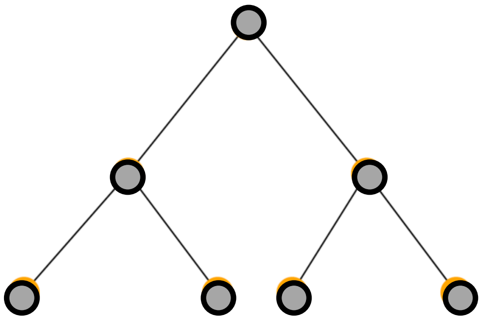
Some Insights

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



Some Insights

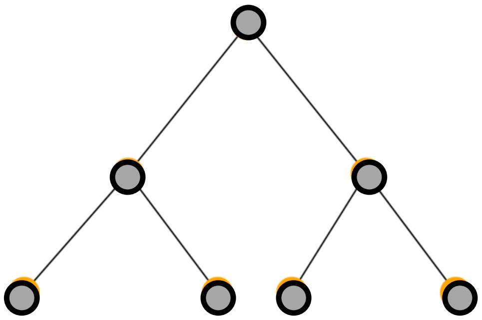
- What about $\Delta = 3$?
  - E.g., complete binary tree
  - $d_N(u,v) \leq 2 \log n$
  - Can we do better than $\log n$?



- What about $\Delta = 2$?

Some Insights

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

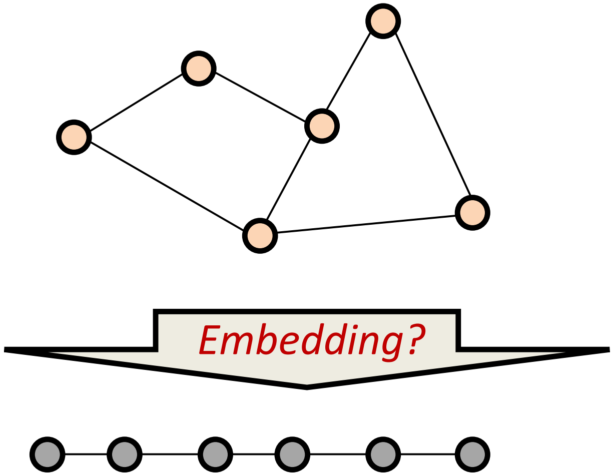


- What about Δ = 2?
  - E.g., set of lines and cycles

How hard is it to design a DAN?

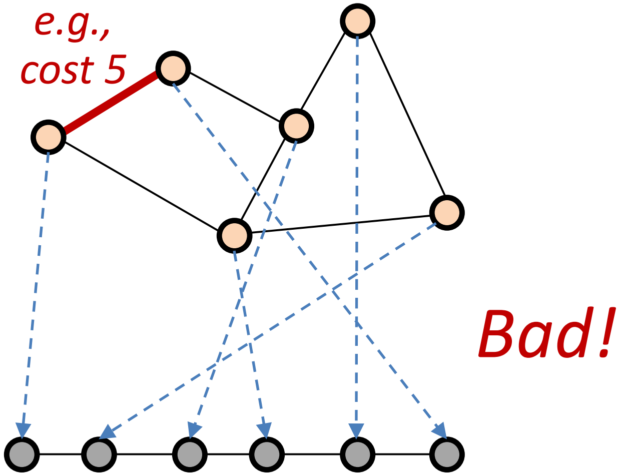
DAN design can be NP-hard

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



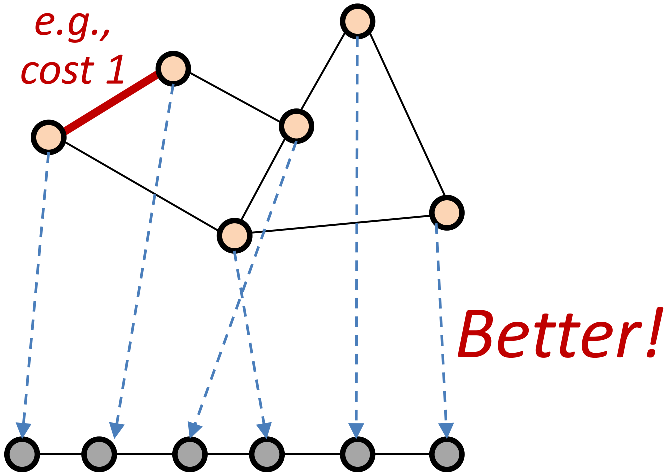
DAN design can be NP-hard

- 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

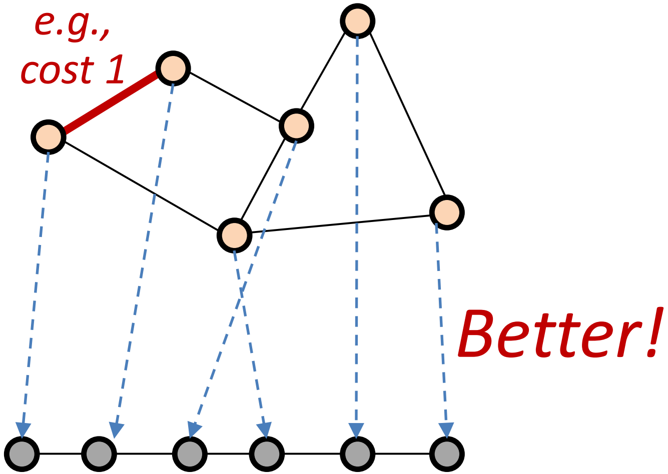
- Example Δ = 2: A Minimum Linear Arrangement (MLA) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - Minimize sum of lengths of virtual edges



DAN design can be NP-hard

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

NP-hard, and so is DAN design.



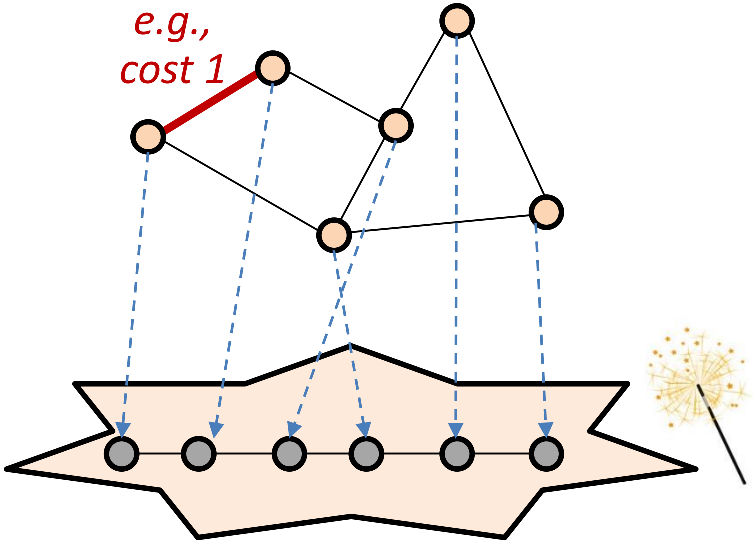
DAN design can be NP-hard

- 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!

Also Related To…:

- Sparse, distance-preserving (low-distortion) **spanners**

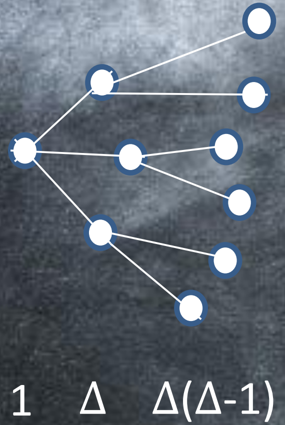
- But:
  - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
  - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
  - We require ***constant degree***

Expected Path Length in Traditional Networks?

Theorem (Traditional Networks):

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

Proof.



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

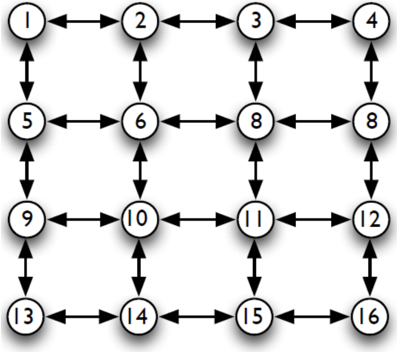
Corollary: Constant-degree graphs have at least logarithmic diameter.

Example: Clos, Bcube, Xpander.

Can DANs do better?

Yes, constant-degree DANs can!

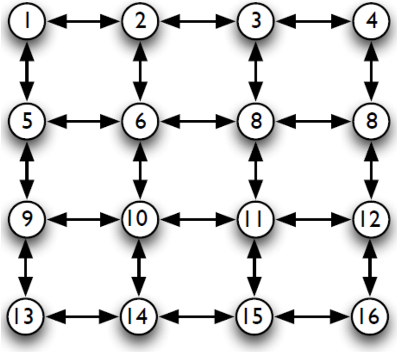
- Example 1: **demand**



- Oblivious design diameter ***O(log n)*** (e.g., Δ-ary tree)
- But constant-degree DAN can serve it at cost ***O(1)*** .

Yes, constant-degree DANs can!

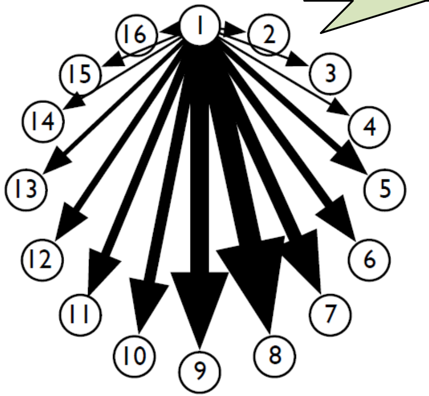
- Example 1: **demand**



- Oblivious design diameter ***O(log n)*** (e.g., Δ-ary tree)
- But constant-degree DAN can serve it at cost ***O(1)*** .

- Example 2: **demand**

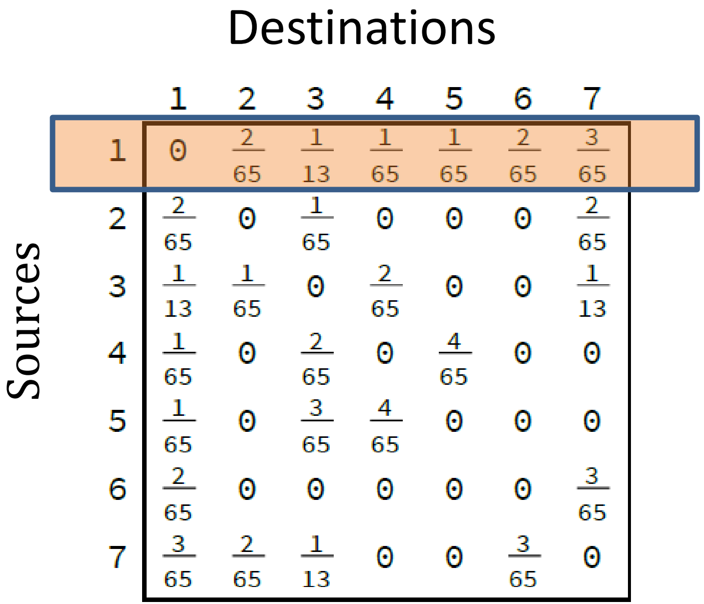
High-degree, but ***skewed***.



- Oblivious design: diameter ***O(log n)*** (e.g., Δ-ary tree)
- But constant-degree DAN can serve it at cost ***O(1)*** (e.g., **Huffman** tree for node 1)

Limitations of DANs: Lower Bounds on EPL

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

- Consider ***(source) node 1***: best Δ-ary tree we can build for this source is **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?

Lower Bound Idea:
Leverage Coding or Datastructure!

Destinations

Sources

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

- Consider ***(source) node 1***: best Δ-ary tree we can build for this source is **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 known on EPL known for binary trees, e.g. ***Mehlhorn*** 1975 for BST

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

- Consider ***(source) node 1***: best Δ-ary tree we can build for this source is **Huffman tree** for its destinations [0,1/65,1/13,1/65
  - resp. **Knuth/M**
    property requir

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

- How good can this tree be?

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

Entropy lower bound for Binary Search Trees (BST):

Let H(***p***) be the entropy of the frequency distribution ***p*** = (p1, p2, . . . , pn). Let T be an optimal binary search tree built for the above frequency distribution. Then EPL(***p***, T) ≥ H(***p***)/log(3).

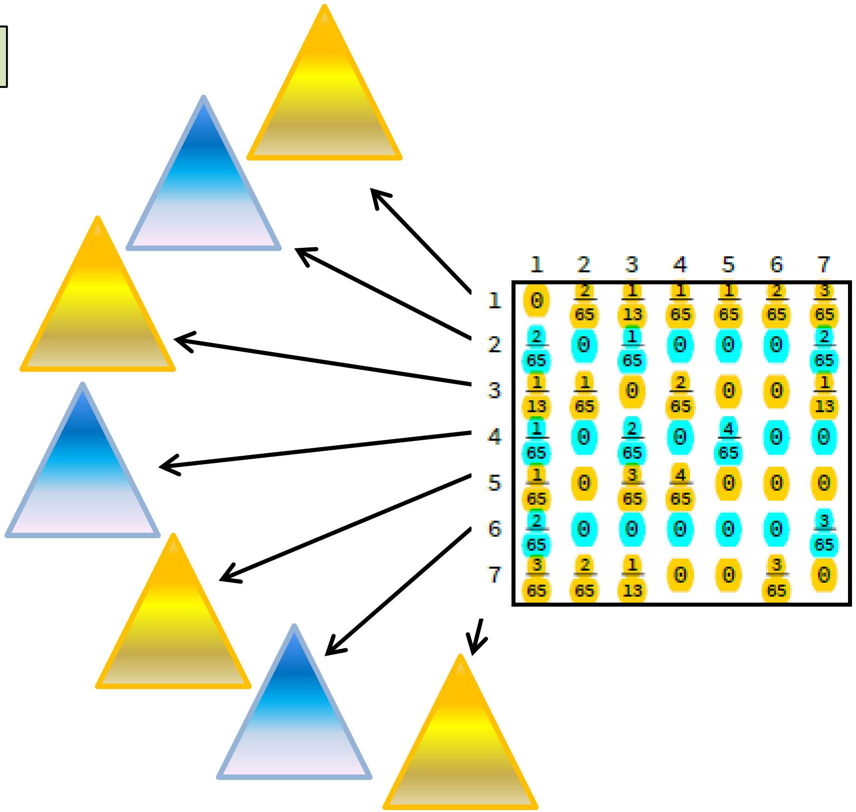
Corollary: Can generalize it easily to other degrees Δ.

An Entropy Lower Bound (Sources)

sources

destinations

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

Lower Bound: Sources + 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

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

$\mathcal{D}$

Lower Bound: Summary

Considering union of n trees,

Marginal distribution of source i

Optimal tree

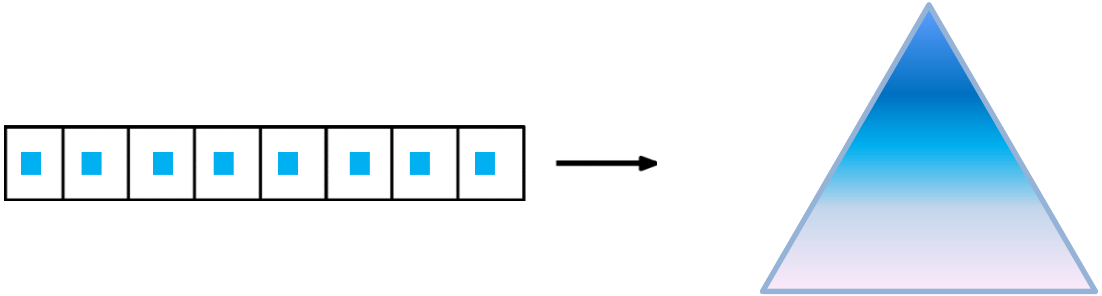
$$\mathrm{EPL}(\mathcal{D},N_{\Delta}) \geq \sum_{i=1}^{n} p(i)\, \mathrm{EPL}(\mathcal{D}[i], T^{i}_{\Delta})$$

$$\geq \sum_{i=1}^{n} p(i)\, (H_{\Delta}(Y\,|X=i))$$

$$= \Omega(H_{\Delta}(Y\,|X))$$

Can DANs Reach The Entropy Speed Limit?
Upper Bounds on EPL

Ego-Trees Revisited

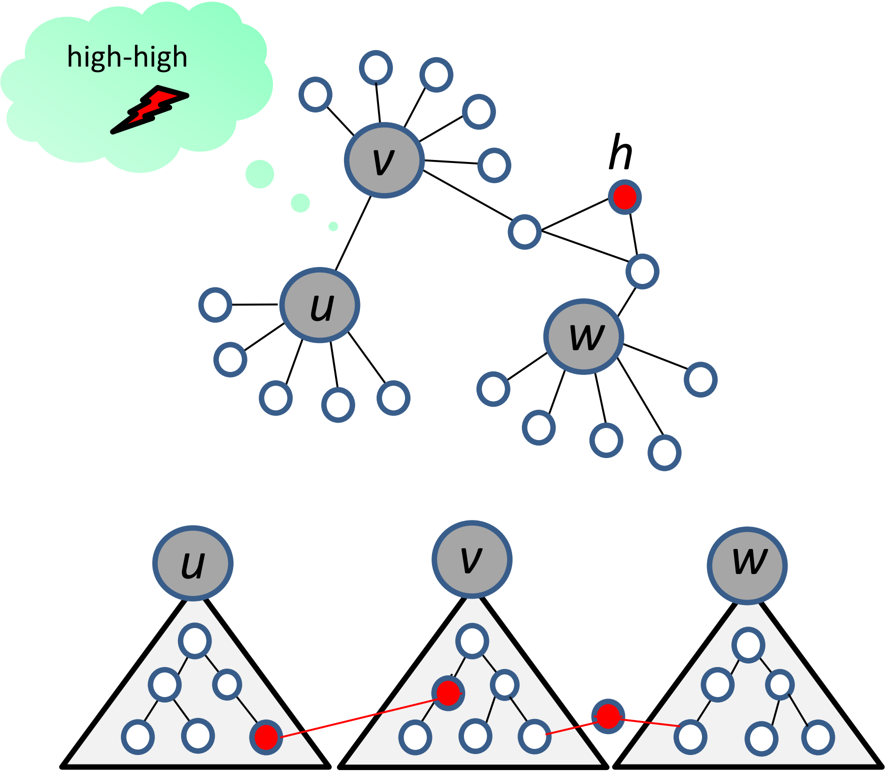


$\mathcal{D}[i]$

T^{i}_{Δ}

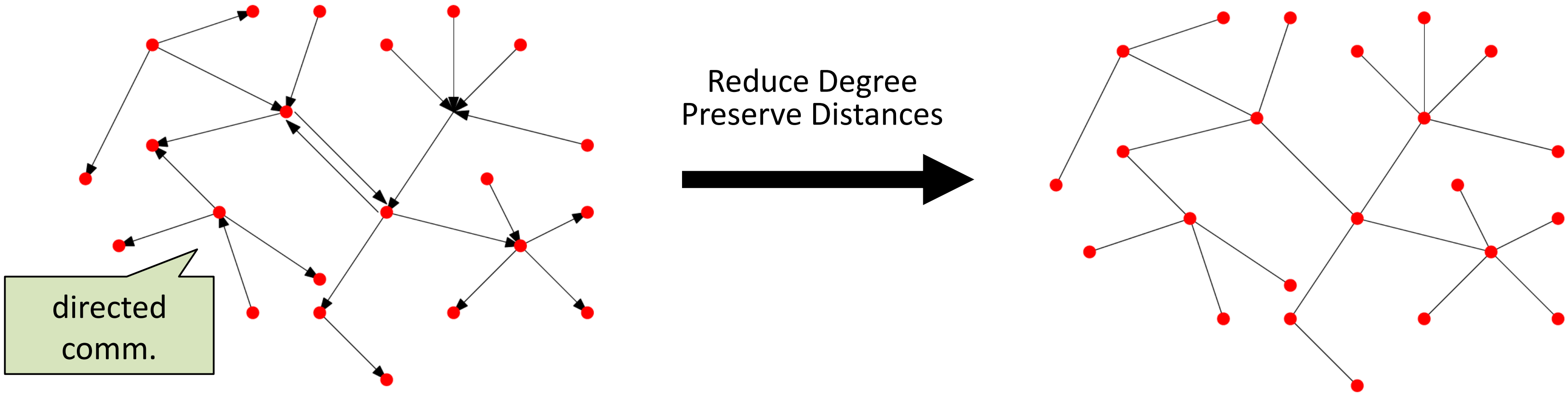
(Tight) Upper Bounds: Algorithm Idea

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



How to Reduce Degree?
Example: Tree Distributions

Theorem: Let $\mathcal{D}$ be such that $G_{\mathcal{D}}$ is a tree (ignoring the edge direction). It is possible to generate a DAN with maximum degree 8, such that, $\mathrm{EPL}(\mathcal{D},\mathrm{N}) \leq \mathrm{O}(\mathrm{H}(\mathrm{Y}|\mathrm{X})+\mathrm{H}(\mathrm{X}|\mathrm{Y}))$.



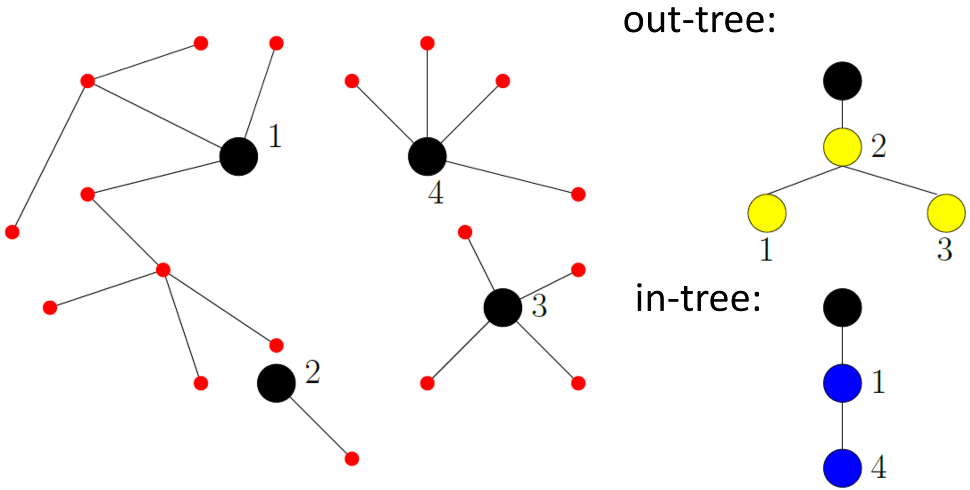
Reduce Degree
Preserve Distances

directed
comm.

Tree Distributions

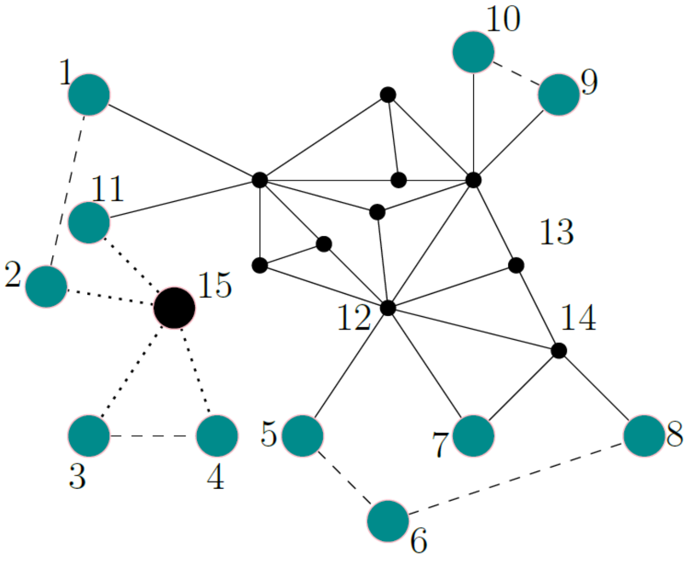
Proof idea:

- Make tree **rooted** and directed: gives **parent-child** relationship
- Arrange the outgoing edges (to children) of each node in a **binary (Huffman) tree**
- Repeat for the incoming edges: make another **another binary (Huffman) tree** with incoming edges from children
- Analysis
  - Can ***appear in at most 4 trees***: in&out own tree and in&out tree of parent (parent-child helps to avoid many “children trees”)
  - ***Degree*** at most 6
  - Huffman trees maintain ***distortion***: proportional to ***conditional entropy***



Generalize to Arbitrary Sparse Distributions

Demand graph:

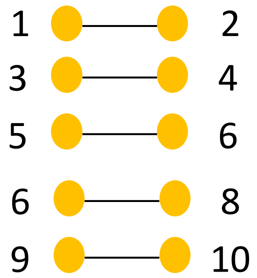


Demand

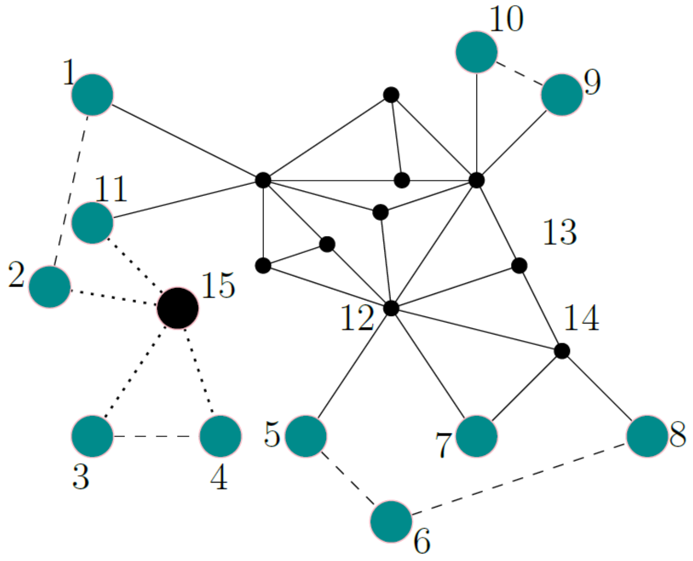
Sparse Distributions: Construction

Demand graph:

Low-low edges: remove from guest, add to DAN



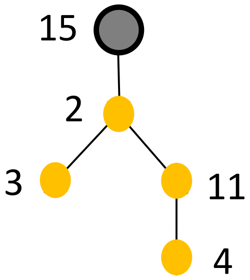
DAN



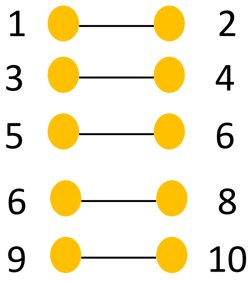
Demand

Sparse Distributions: Construction

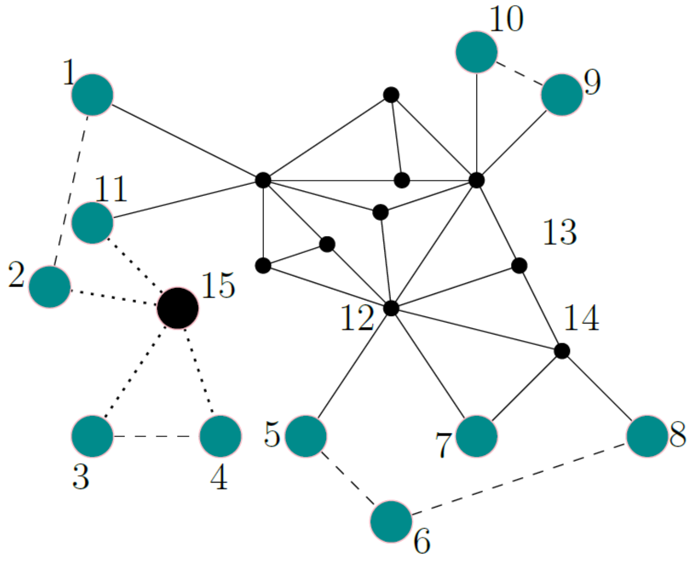
High degree nodes with low degree neighbors: need binarization



Low-low edges: remove from guest, add to DAN



Demand graph:

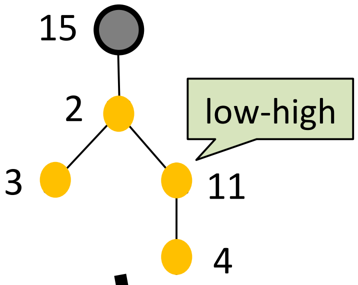


DAN

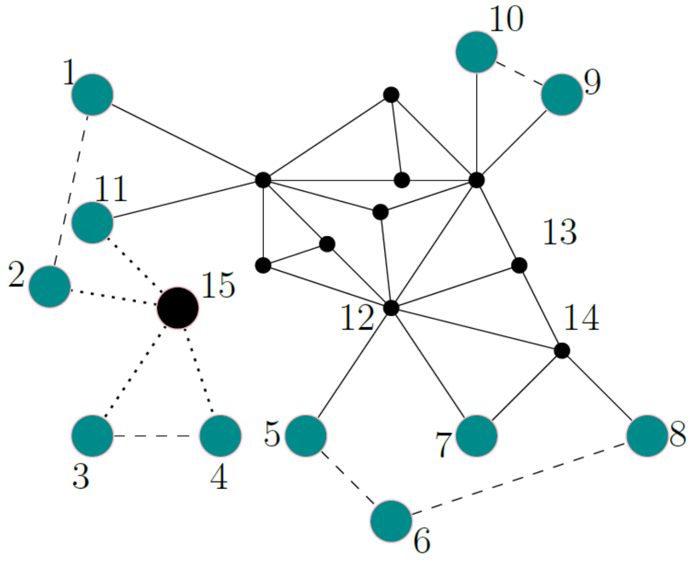
Demand

Sparse Distributions: Construction

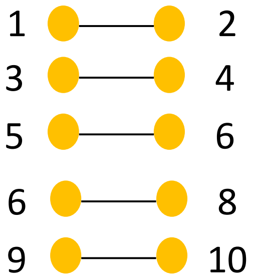
High degree nodes with low degree neighbors: need binarization



Demand graph:



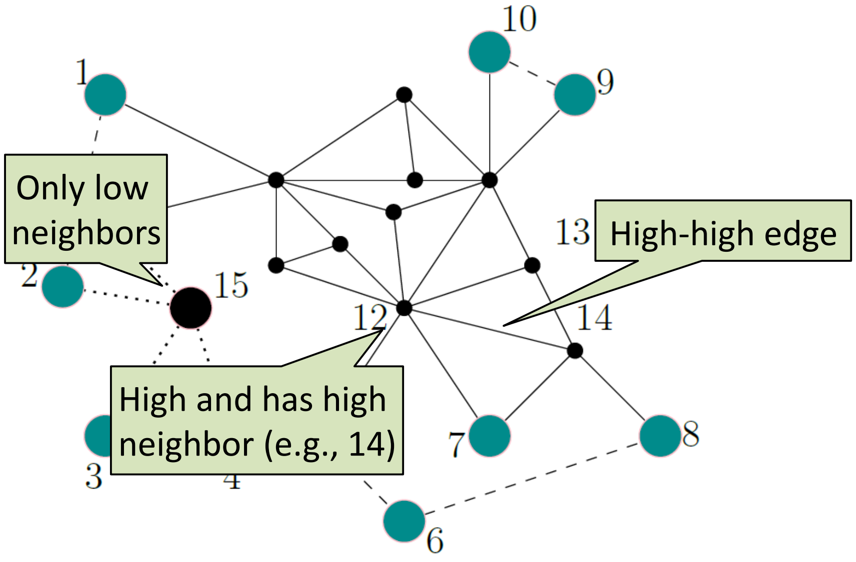
Low-low edges: remove from guest, add to DAN



Remove from guest, add to DAN!

Sparse Distributions: Construction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- Find high degree nodes having only low degree neighbors (e.g., 15 but not 12):
  - Create optimal **binary tree** with low degree neighbors
- **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** again

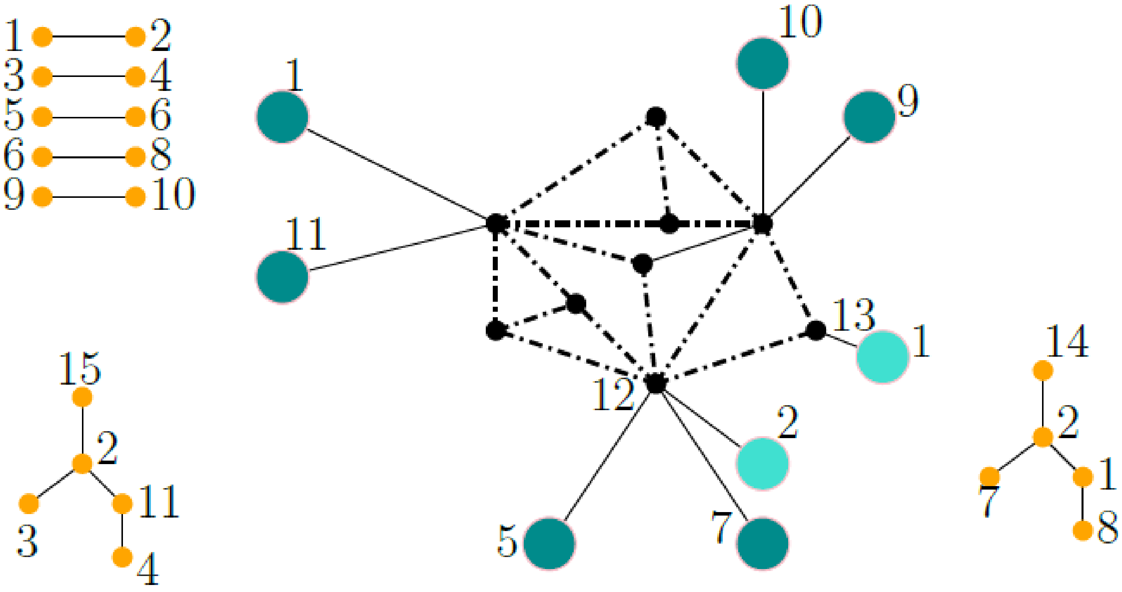


Example Illustrated

- Find **low** degree nodes
- Mark **low-low** edges
- Find **high** degree nodes with only **low** degree neighbors (e.g., 15)
- Make **binary tree** for them
- Add **low** degree node (e.g., 1 and 2) between **high-high** edge (e.g., 12-14, e.g., 14 has *two* high-degree neighbors 12 and 13)
- Now high nodes have only low neighbors as well, so make **tree** again (at 12-14)

DAN:

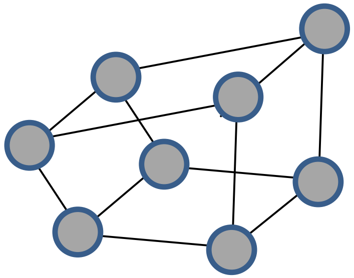
Demand:



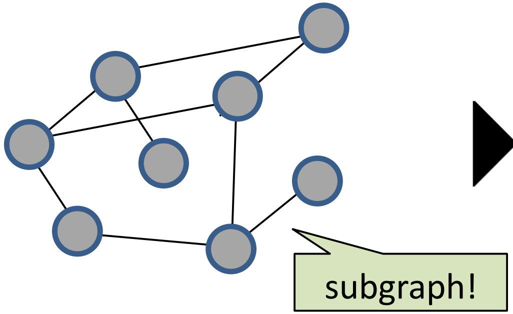
Regular and Uniform Distribution: Leveraging The Connection to Spanners

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: can design a constant degree DAN providing an ***optimal expected path length*** (i.e., O(H(X|Y)+H(Y|X)).

r-regular *and uniform* **request***:*

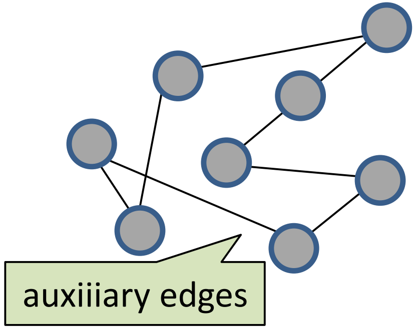


Sparse, ***irregular (constant)*** **spanner***:*



subgraph!

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



auxiiiary edges

Regular and Uniform Distribution: Leveraging The Connection to Spanners

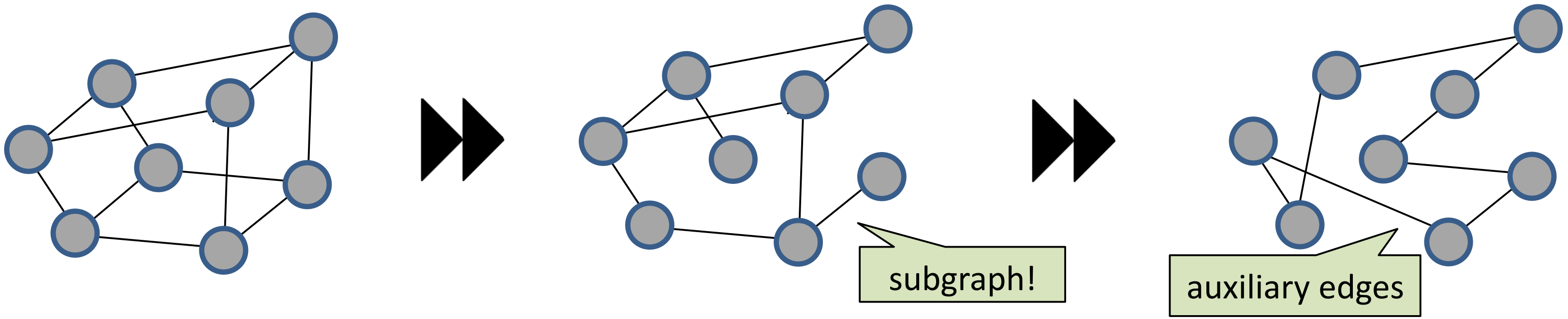
Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant**, **sparse**) spanner for this request graph: can design a constant degree DAN providing an ***optimal expected path length*** (i.e., O(H(X|Y)+H(Y|X)).

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

r-regular *and uniform* **request***:*

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

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



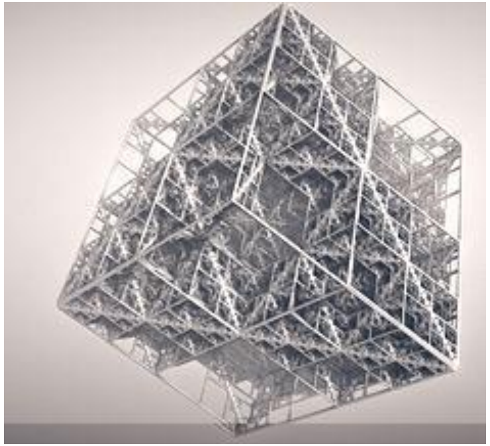
Reduction from Spanner

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

- Consequences: optimal DAN designs for
  - **Hypercubes** (with n log n edges)
  - **Chordal** graphs
  - Trivial: graphs with polynomial degree (**dense graphs**)

Has sparse 3-spanner.

Has sparse O(1)-spanner.



A dense graph

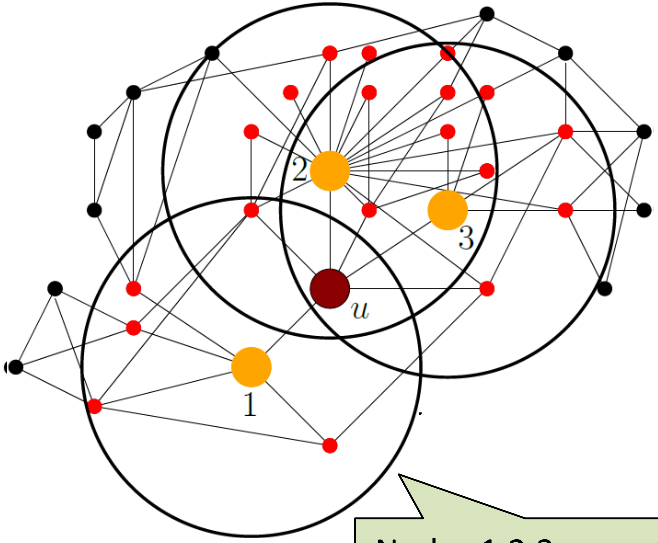
Another Example: Demands of Locally-Bounded Doubling Dimension

- **LDD**: $G_{\mathscr{D}}$ 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

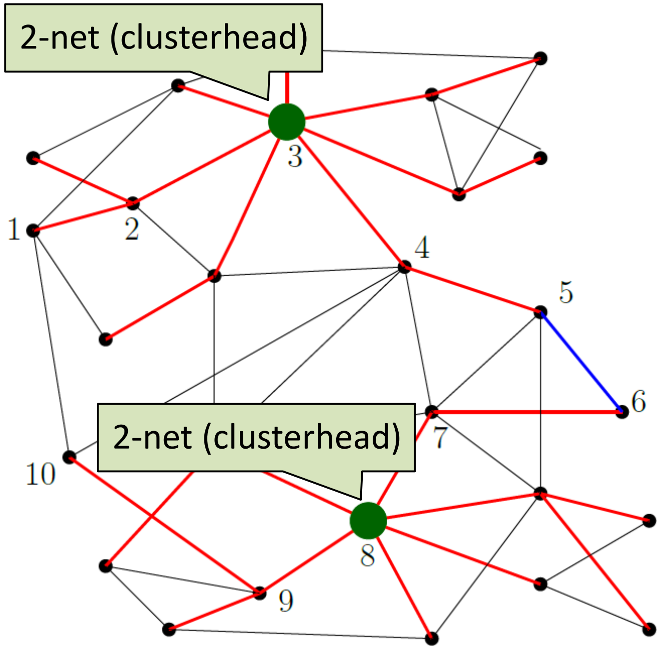
- V' sufficiently "independent": for every $u, v \in V'$, $d_G(u, v) > \boldsymbol{\varepsilon}$
- "dominating" V: for each $w \in V$, $\exists$ at least one $u \in V'$ such that, $d_G(u,w) \leq \boldsymbol{\varepsilon}$

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.

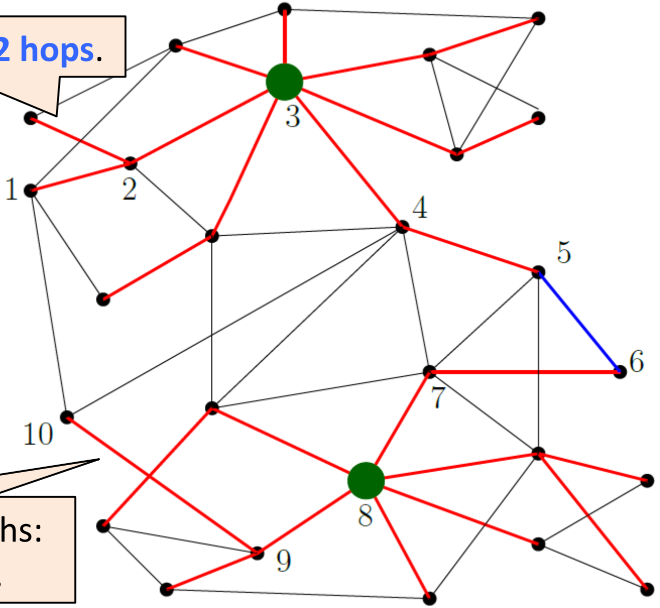


9-Spanner for LDD (= optimal DAN)

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**.

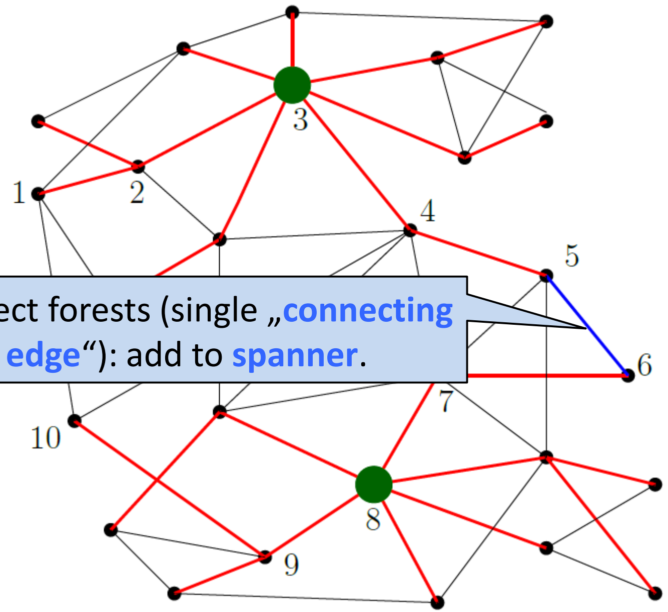
9-Spanner for LDD (= optimal DAN)

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**.

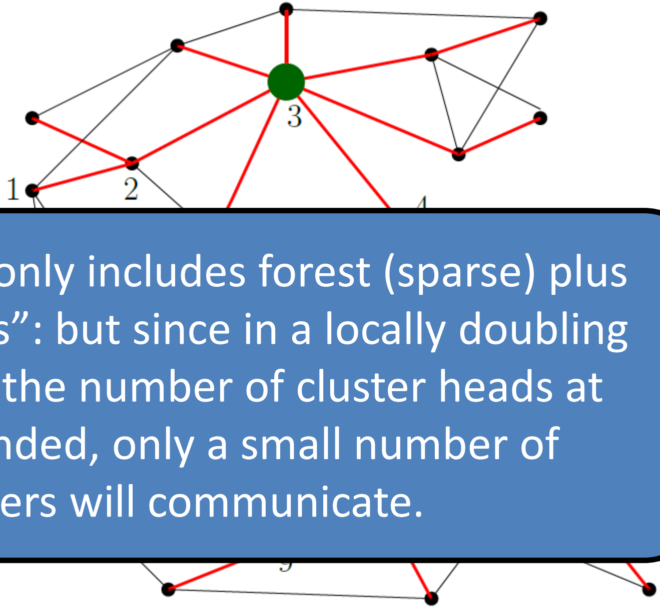
9-Spanner for LDD (= optimal DAN)

Distortion 9: 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

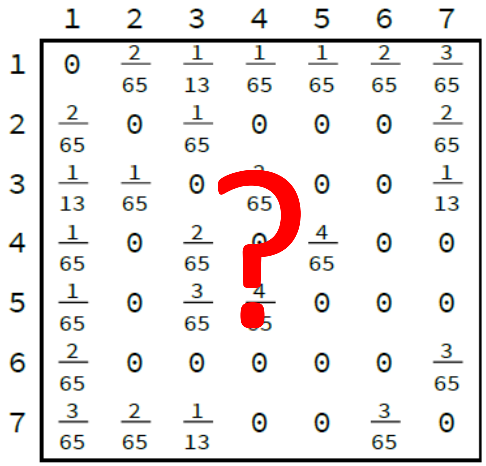
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**),
Vienna, Austria, October 2017.

So how useful are entropy-proportional DANs?

It depends…

- Demand-oblivious network: no information about matrix: **maximum uncertainty (entropy)**, so EPL = entropy = **Ω(log n)…**
  - … even if the demand matrix has a very low **actual entropy**

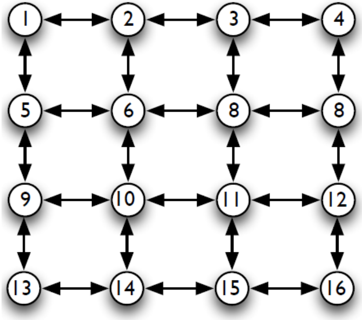
- **Actual entropy** depends on ***spatial locality*** of communication

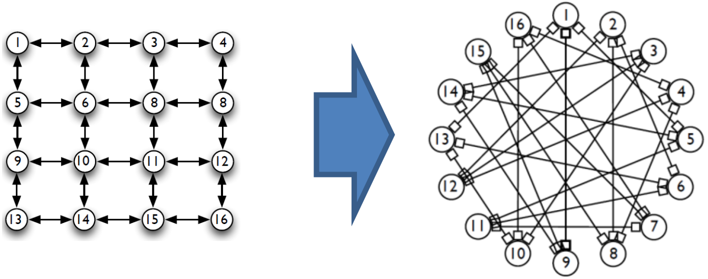


$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13}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

Example 1: 2-dim Grid

Low **spatial locality**: conditional entropy ***less than two*** (i.e., O(1))

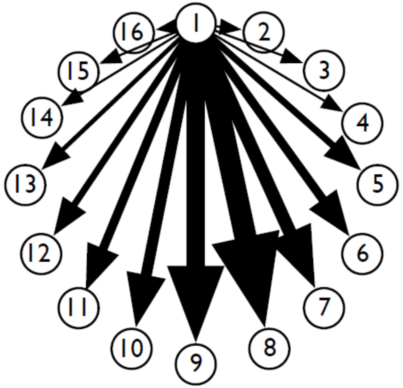


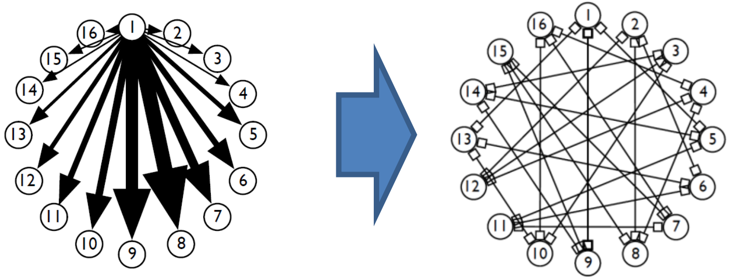


But **embedding** a 2-dim grid demand graph on a const-degree expander would result in ***Ω(log n)*** path lengths

Example 2: Weighted Star

Low **spatial locality**: conditional entropy can be ***very small*** if skew is large

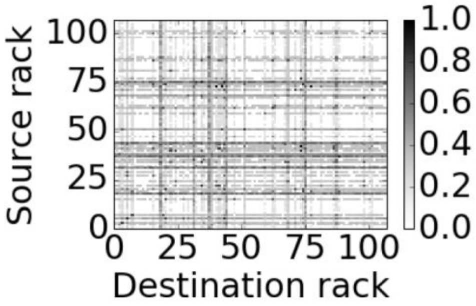




But **embedding** a weighted star demand graph on a const-degree expander would result in ***Ω(log n)*** path lengths

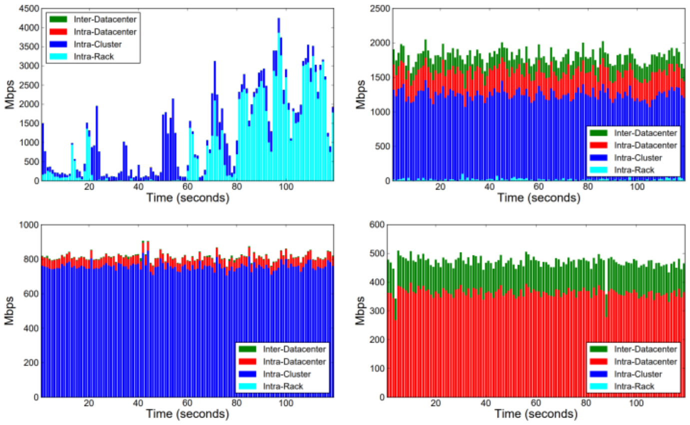
Many Empirical Studies Confirm Spatial Locality

Microsoft



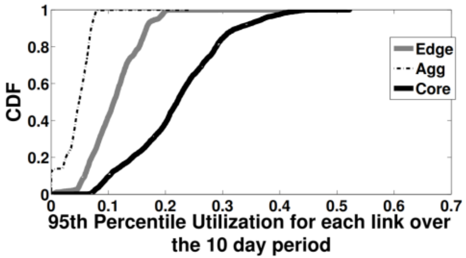
ProjecToR @ SIGCOMM 2016

Facebook



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

Benson et al.



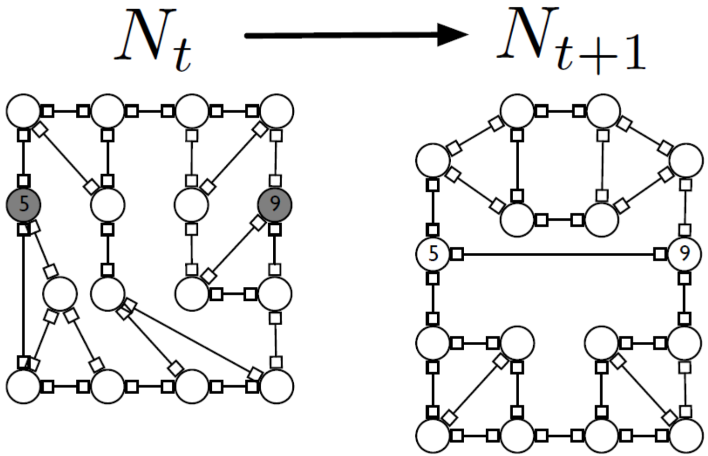
Understanding Data Center Traffic Characteristics @ WREN 2009

Roadmap

- Vision and Motivation ✓
- An analogy: coding and datastructures ✓
- Principles of Demand-Aware Network (DAN) Designs ✓
- **Principles of Self-Adjusting Network (SAN) Designs**
- Principles of Decentralized Approaches

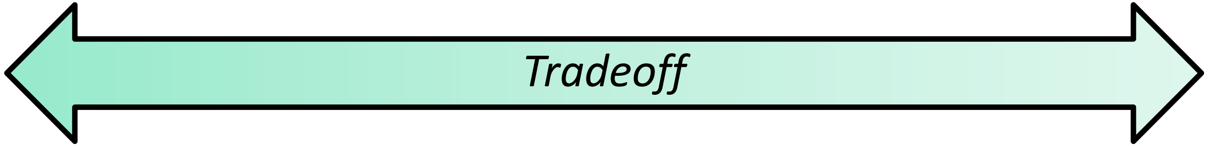
What Are the Objectives and Metrics for SANs?

Model: A Cost-Benefit Tradeoff



Basic question:

How often to reconfigure?

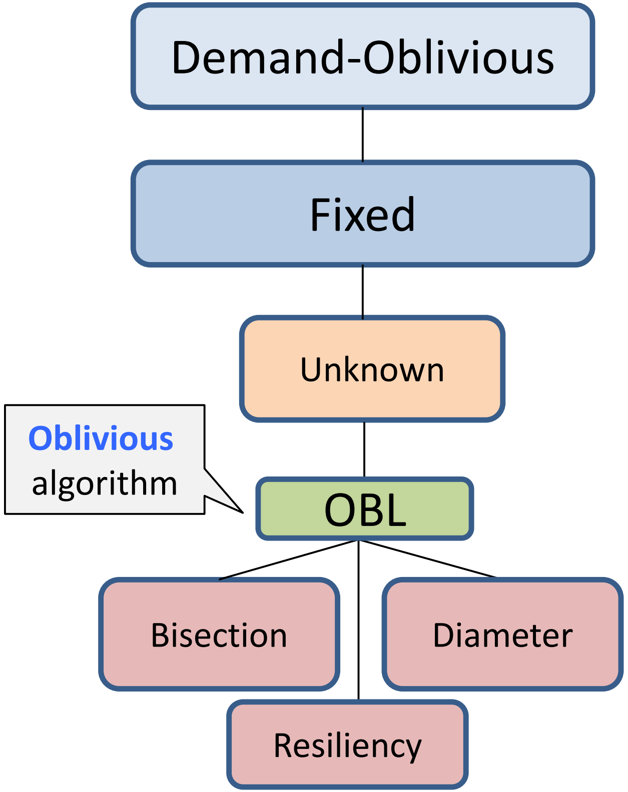


Tradeoff

Short routes
High reconfiguration cost

Low reconfiguration cost
Long routes

A Taxonomy: Traditional Networks



Demand-Oblivious

Fixed

Unknown

Oblivious algorithm

OBL

Bisection

Diameter

Resiliency

Awareness

Topology

Input

Algorithm

Property

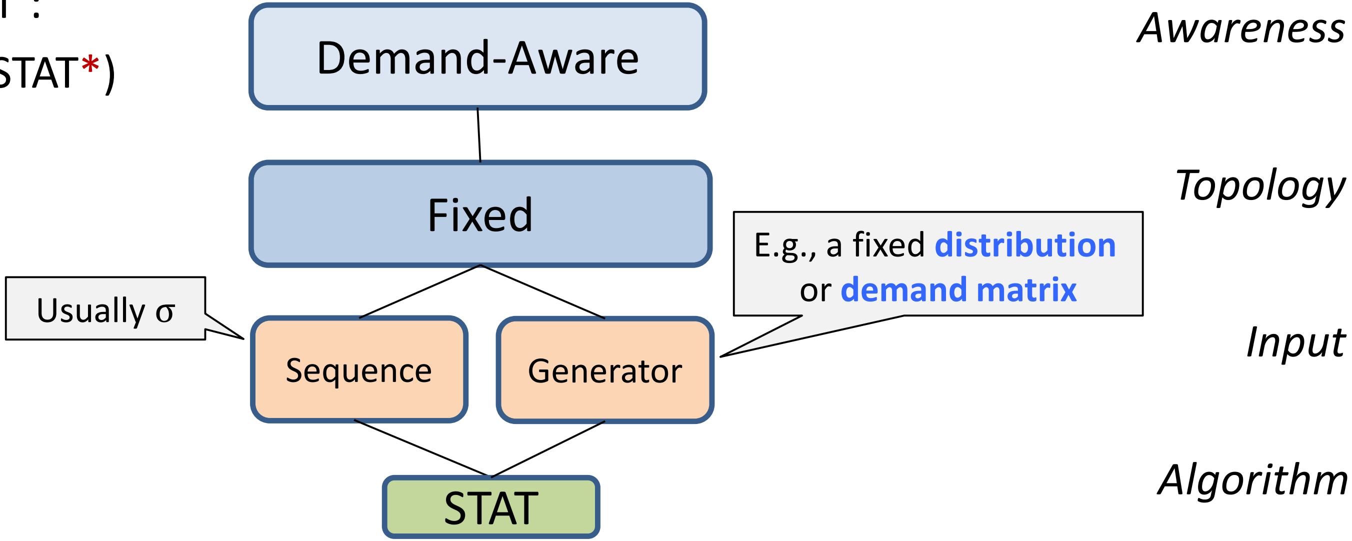
A Taxonomy: Static DANs

Objective:

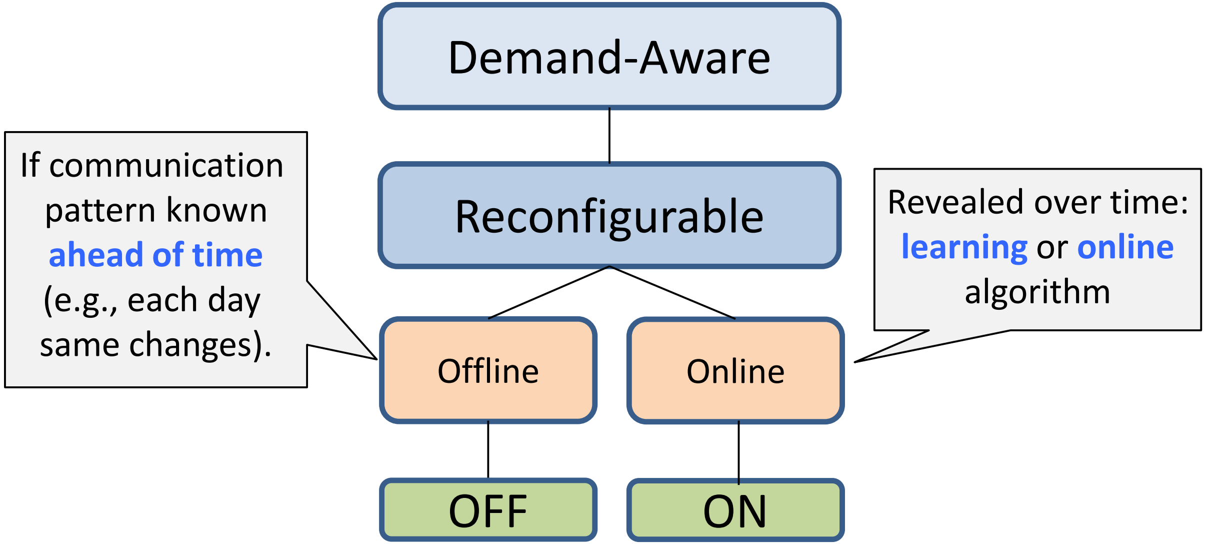
"Close to optimal":

ρ = Cost(STAT)/Cost(STAT*)

is small.



A Taxonomy: Reconfigurable Networks



If communication pattern known **ahead of time** (e.g., each day same changes).

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

Awareness

Topology

Input

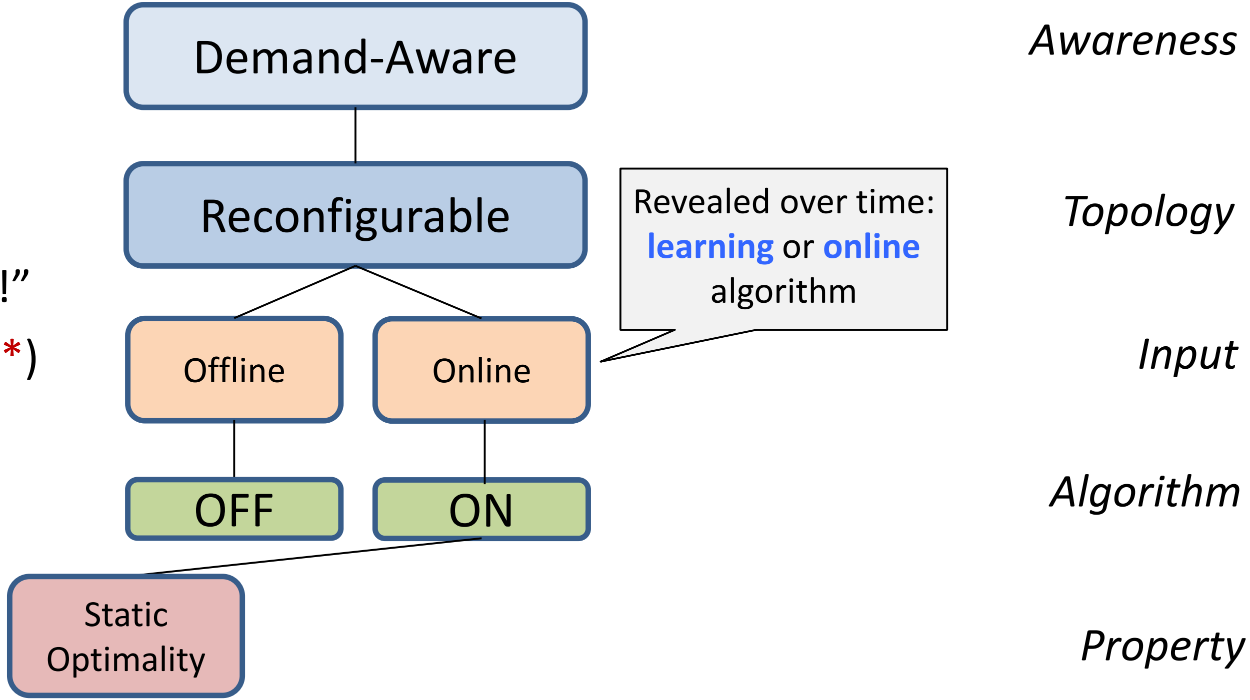
Algorithm

A Taxonomy: Reconfigurable Networks

Static Optimality:

"Don't be worse than static which knows demand ahead of time!"

ρ = Cost(ON)/Cost(STAT*) is constant.



Demand-Aware

Reconfigurable

Offline

Online

OFF

ON

Static Optimality

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

Awareness

Topology

Input

Algorithm

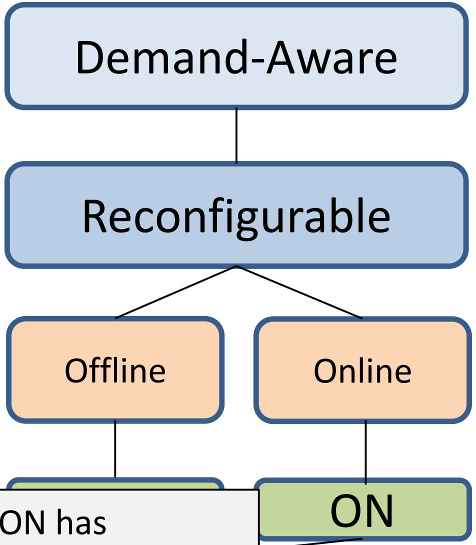
Property

A Taxonomy: Reconfigurable Networks

Static Optimality:

“Don’t be worse than static which knows demand ahead of time!”

ρ = Cost(ON)/Cost(STAT*) is constant.



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

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

Awareness

Topology

Input

Algorithm

Property

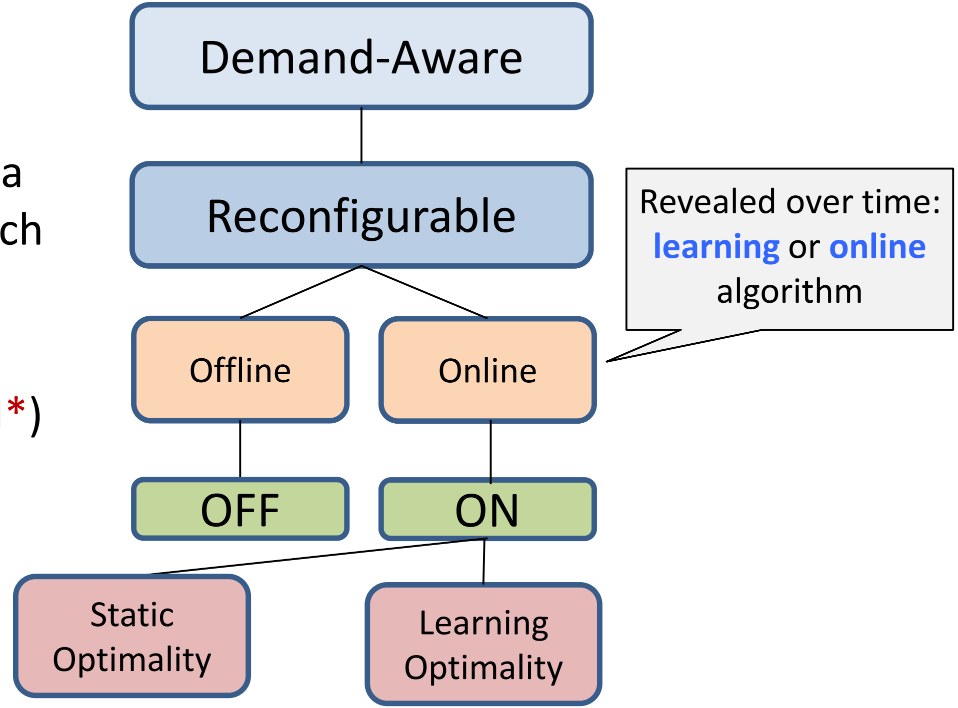
A Taxonomy: Reconfigurable Networks

Learning Optimality:

"Don't be worse than a dynamic algorithm which knows the (fixed) generator!"

ρ = Cost(ON)/Cost(GEN*) is constant.

Always >=1.



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

Awareness

Topology

Input

Algorithm

Property

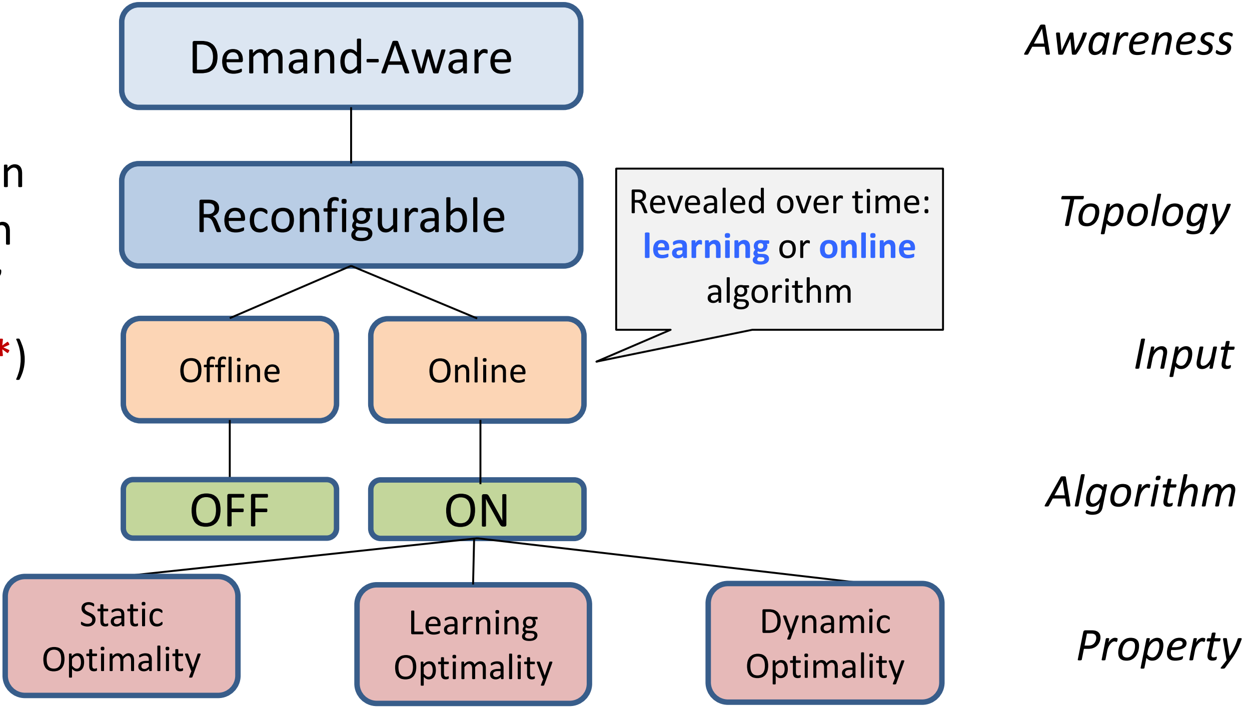
A Taxonomy: Reconfigurable Networks

Dynamic Optimality:

"Don't be worse than an offline algorithm which knows the sequence!"

ρ = Cost(ON)/Cost(OFF*) is constant.

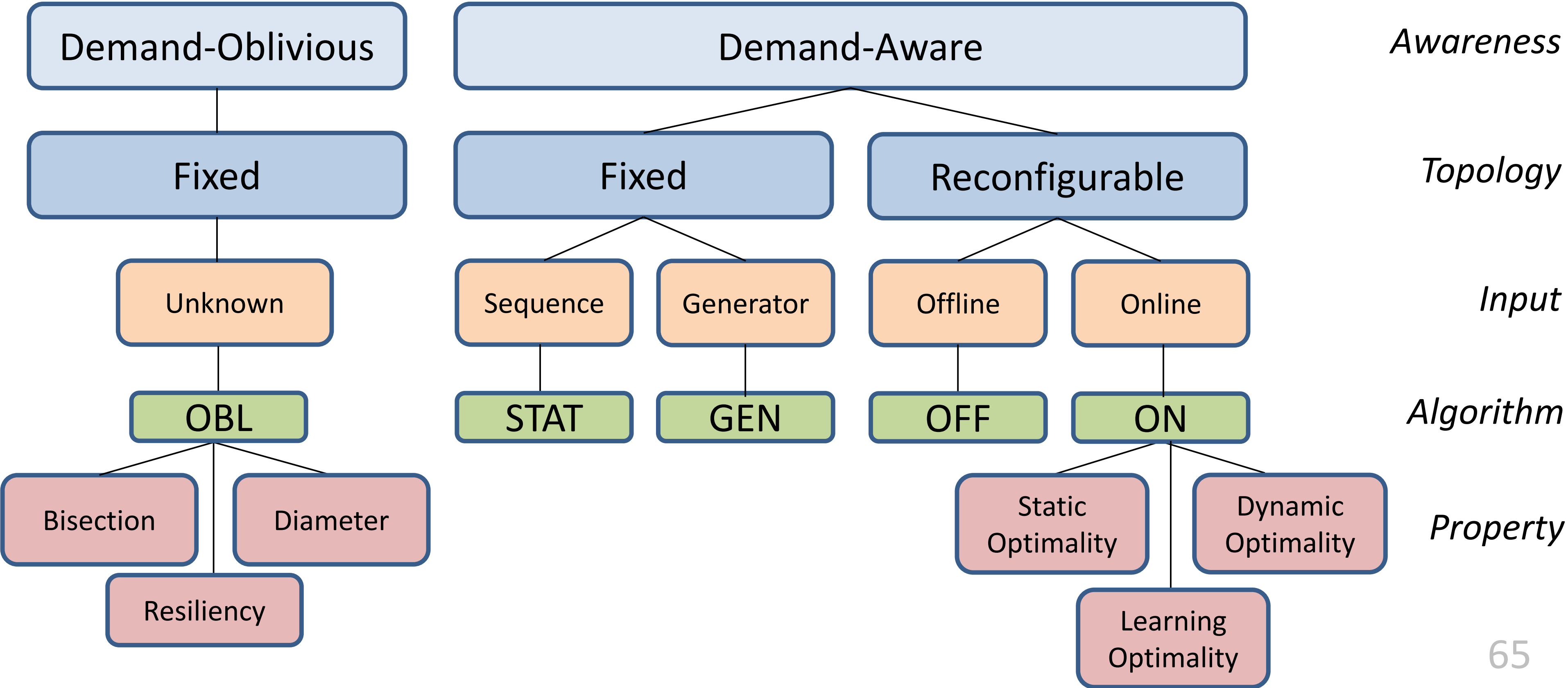
Always >=1.



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

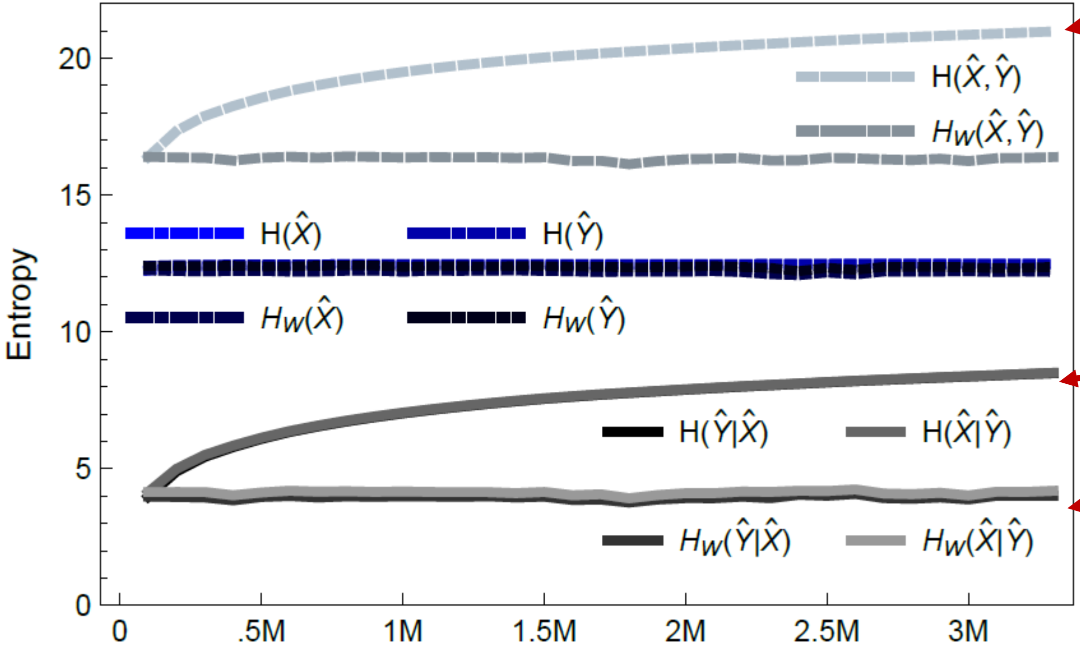
Taxonomy

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



When are SANs better than DANs?

If There is Much Temporal Locality



Oblivious design: proportional to ***joint entropy***

total entropy

entropy of **ego-trees**

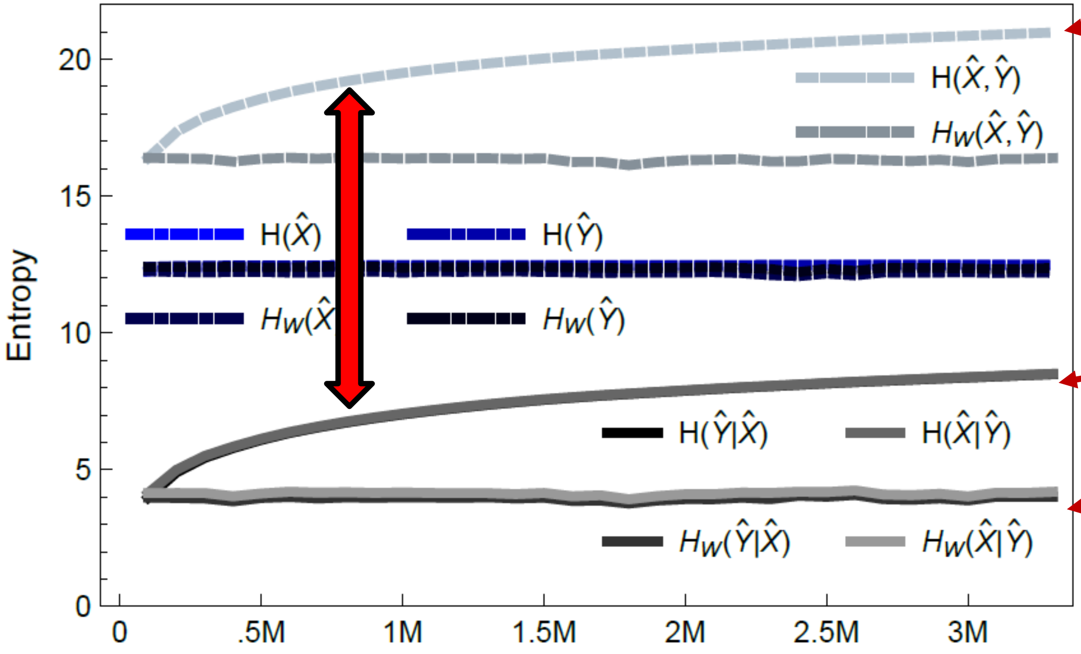
DAN: proportional to ***conditional entropy***

SAN: proportional to ***conditional entropy in time windows W***

Entropy measures in Facebook's workload
(3 million requests)

If There is Much Temporal Locality

Benefit of DAN



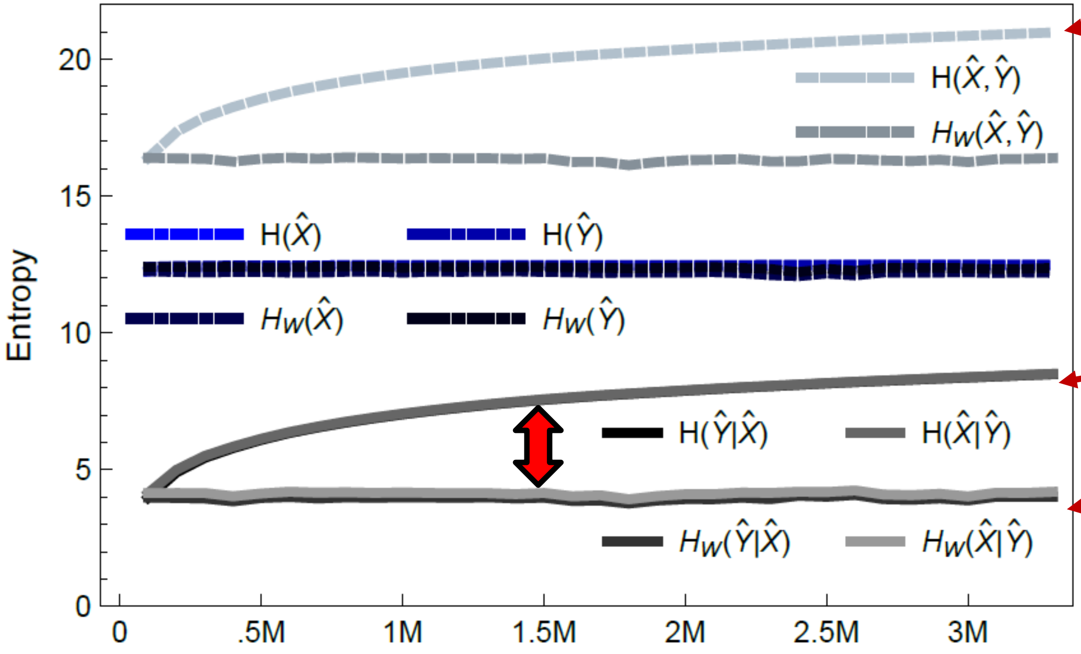
Oblivious design: proportional to ***joint entropy***

DAN: proportional to ***conditional entropy***

SAN: proportional to ***conditional entropy in time windows W***

Entropy measures in Facebook's workload
(3 million requests)

If There is Much Temporal Locality



Oblivious design: proportional to ***joint entropy***

DAN: proportional to ***conditional entropy***

SAN: proportional to ***conditional entropy in time windows W***

Benefit of SAN

(if we change every W=100k requests)

Entropy measures in Facebook's workload
(3 million requests)

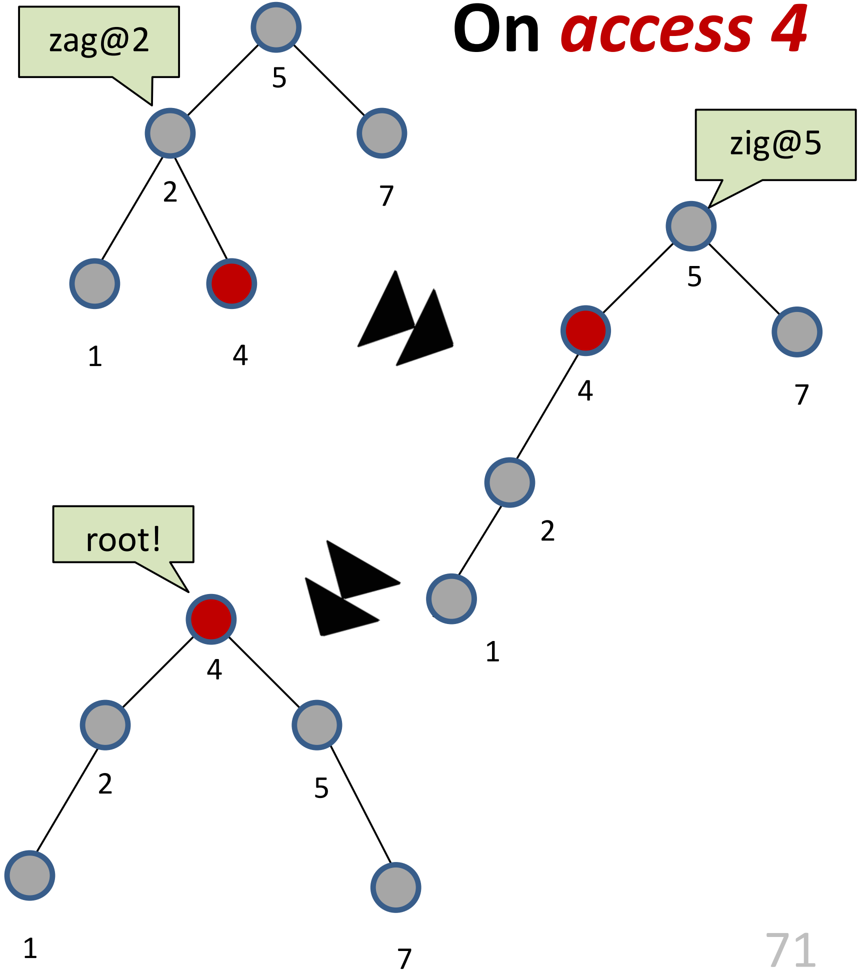
How to Design SANs?

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

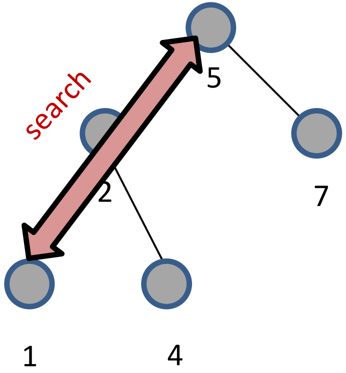
The Classic Self-Adjusting Datastructure: Splay Tree

- 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**, …

On *access 4*



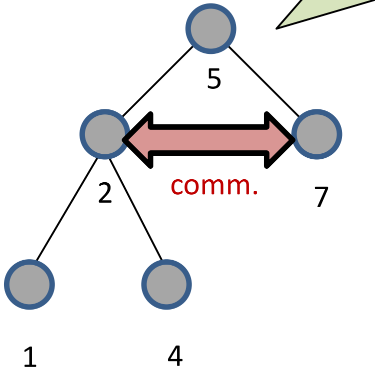
A Simple Idea: Generalize Splay Tree To *SplayNet*



Splay Tree

vs

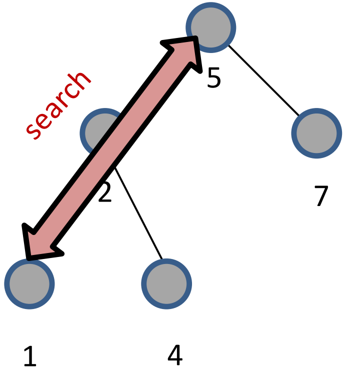
BST is nice for networks:
local (greedy) search!



SplayNet

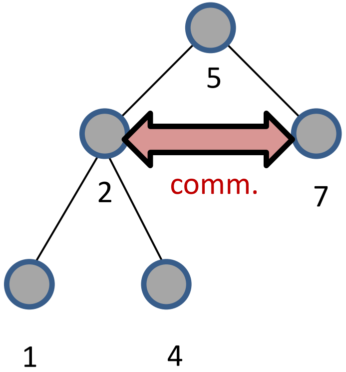
A Simple Idea: Generalize Splay Tree To *SplayNet*

But how?



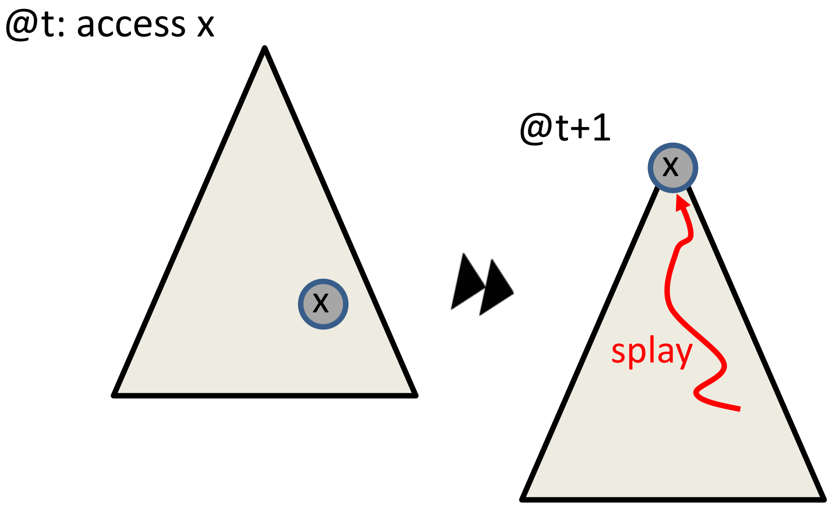
Splay Tree

VS

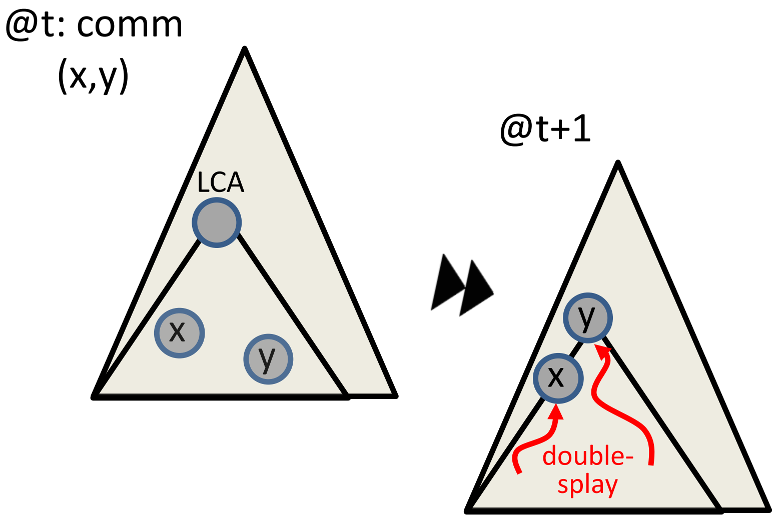


SplayNet

SplayNet: A Simple Idea

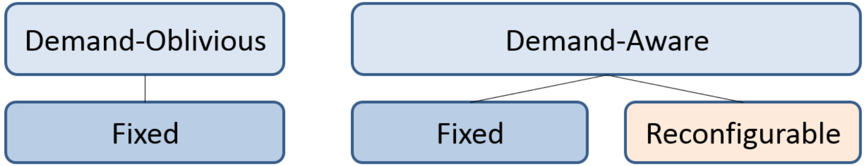


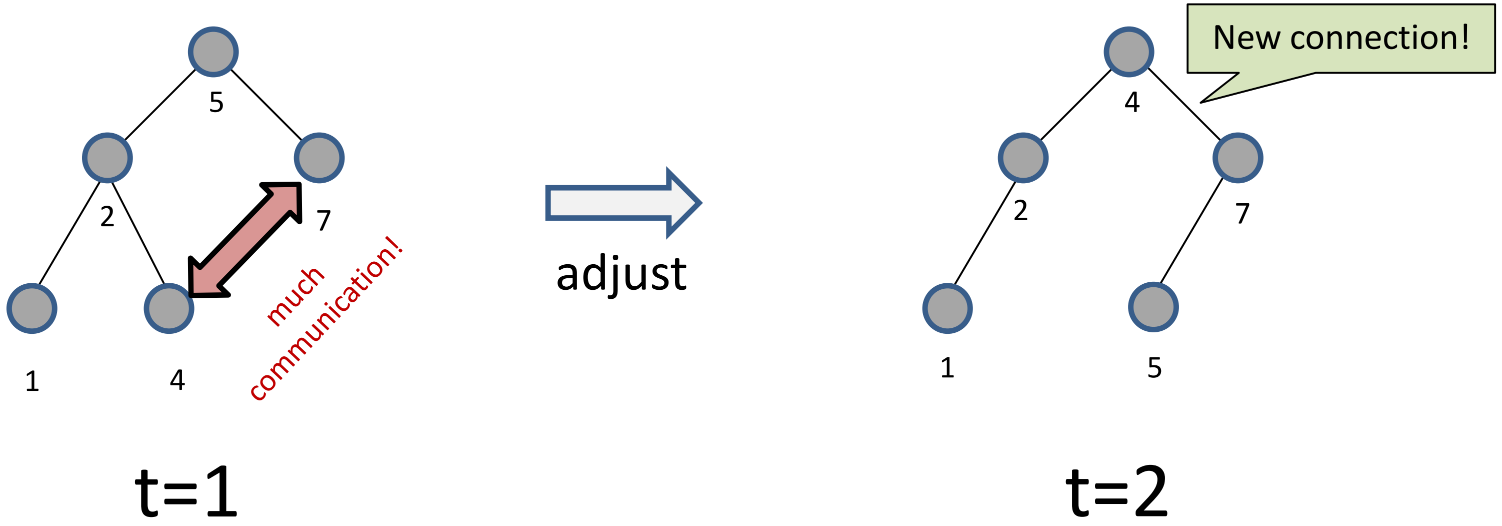
Splay Tree



SplayNet

Example



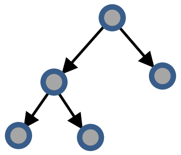


Challenges: How to **minimize reconfigurations**?
How to keep network **locally** routable?

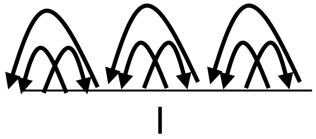
Properties of SplayNets

- **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“

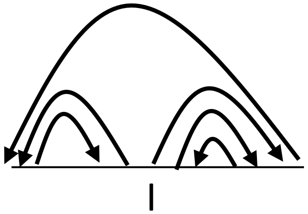
Multicast Scenario



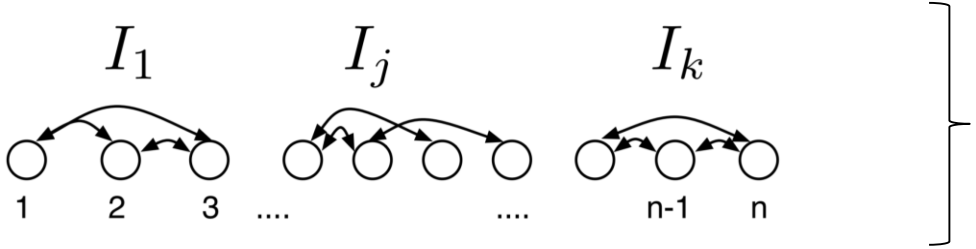
Cluster Scenario



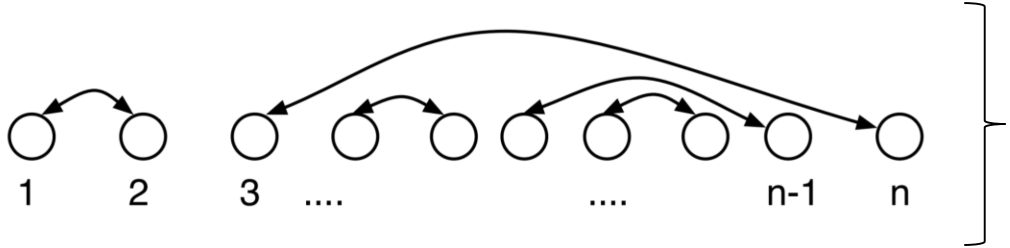
Laminated Scenario



More Specifically



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

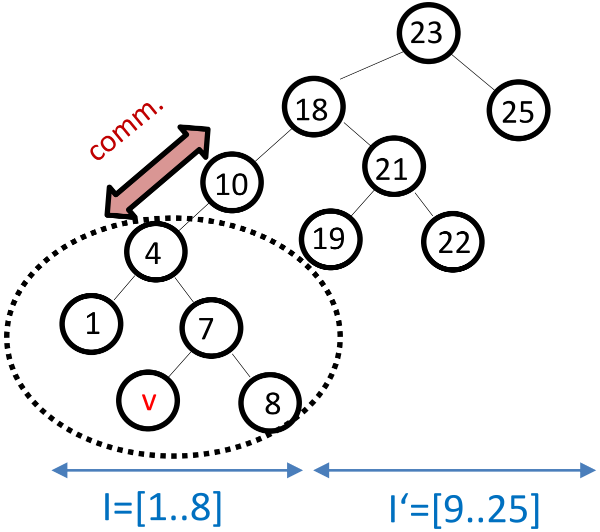


Non-crossing matching scenario: SplayNet will converge to state where all communication pairs are adjacent

Remark: Fast Static SplayNet

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

- **Unlike unordered tree?**



1. Define: flow out of interval I

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

Decouple cost to ouside: distance to root of T_I only

2. Cost of a given tree T_I on I:

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

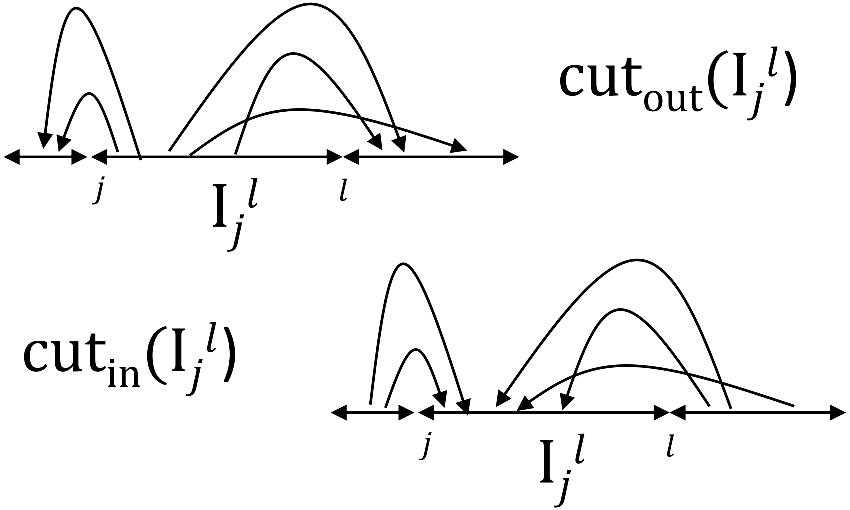
(D_I distances of nodes in I from root of T_I)

3. Dynamic program over intervals

Choose optimal root and add dist to root

Remark: Improved Lower Bounds

Interval Cuts Bound



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

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

If much goes over same cut: leads to long paths!

Edge Expansion Bound

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

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

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

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

- Conductance entropy is lower bound:

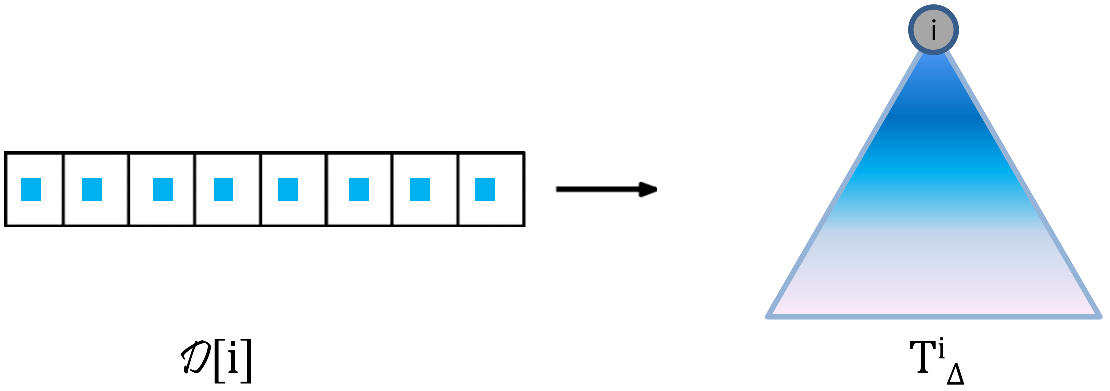
$$\Omega(\phi_H(\mathcal{R}(\sigma)))$$

For SplayNets: stronger lower bounds than conditional entropy.

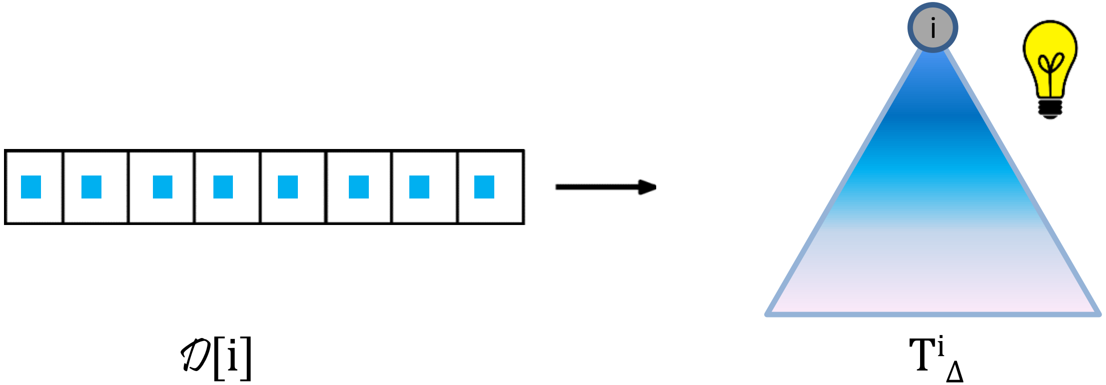
Further Reading

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!



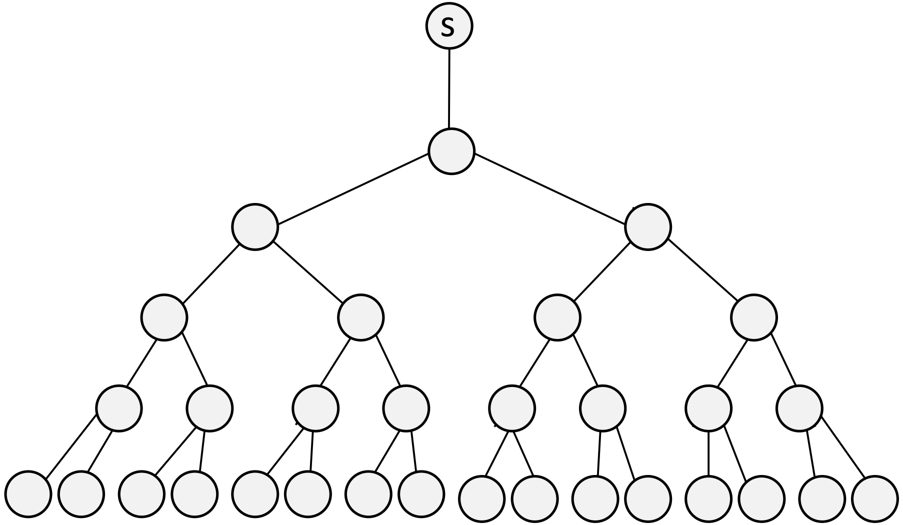
$\mathcal{D}[i]$

T^i_Δ

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

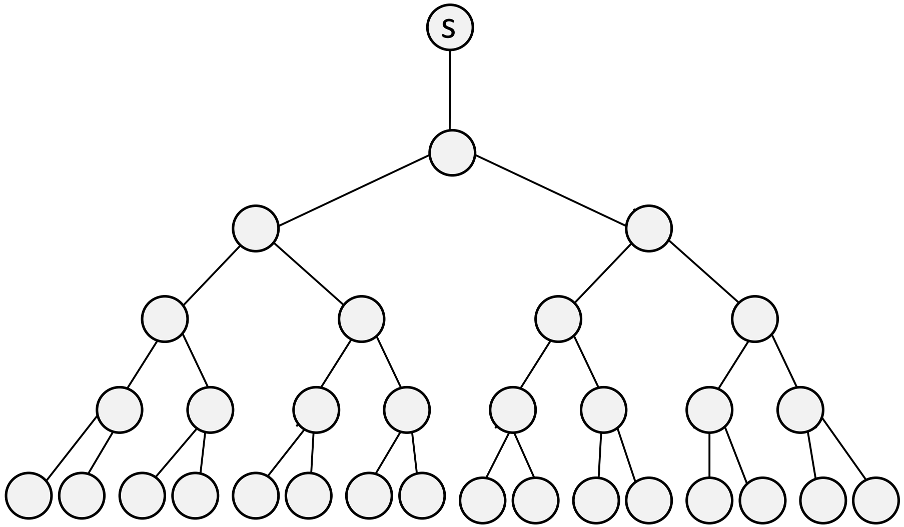
A Balanced Self-Adjusting Tree: Push-Down Tree

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



A Balanced Self-Adjusting Tree: Push-Down Tree

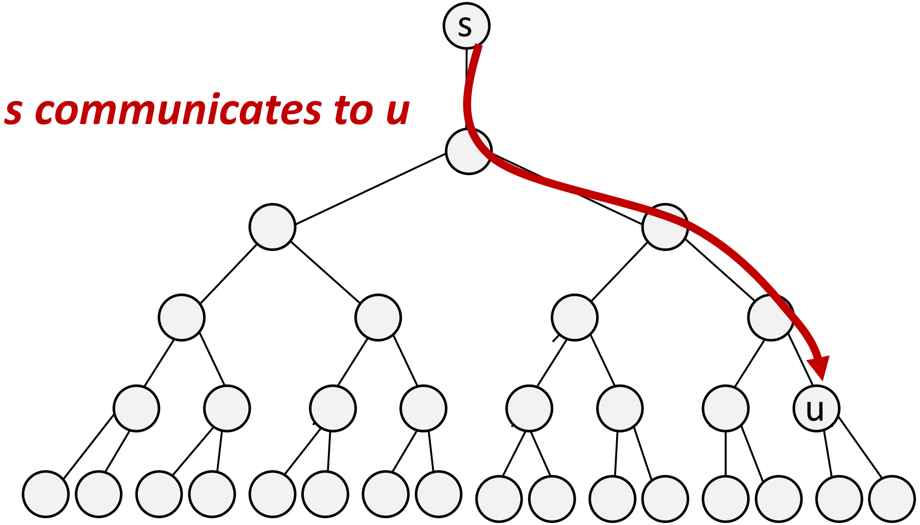
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



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

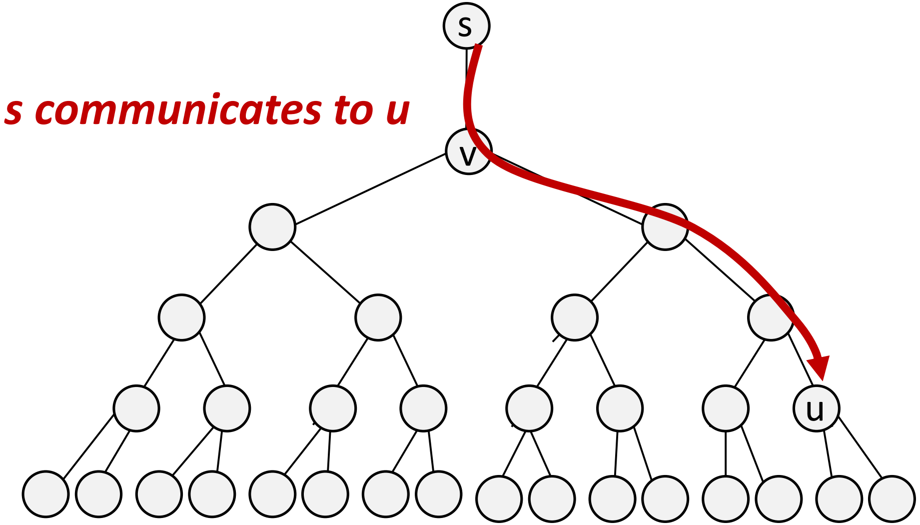
A Balanced Self-Adjusting Tree: Push-Down Tree

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



A Balanced Self-Adjusting Tree: Push-Down Tree

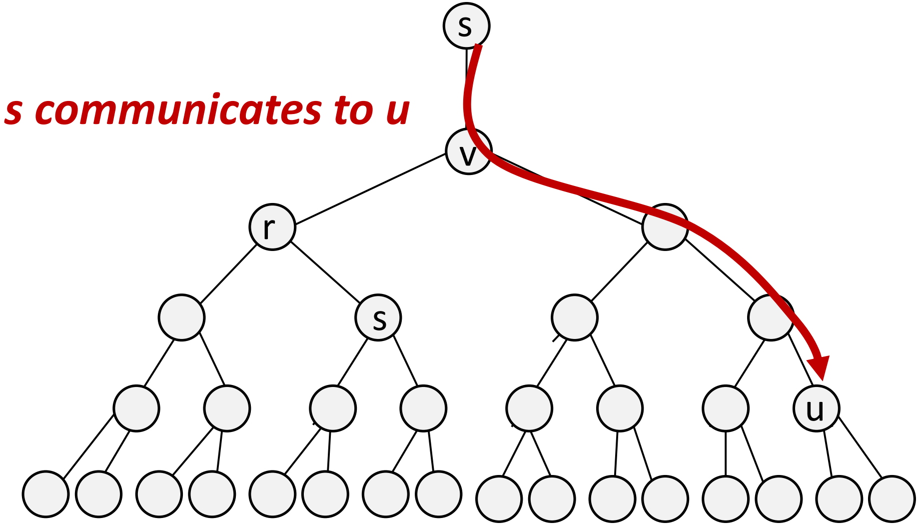
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



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**

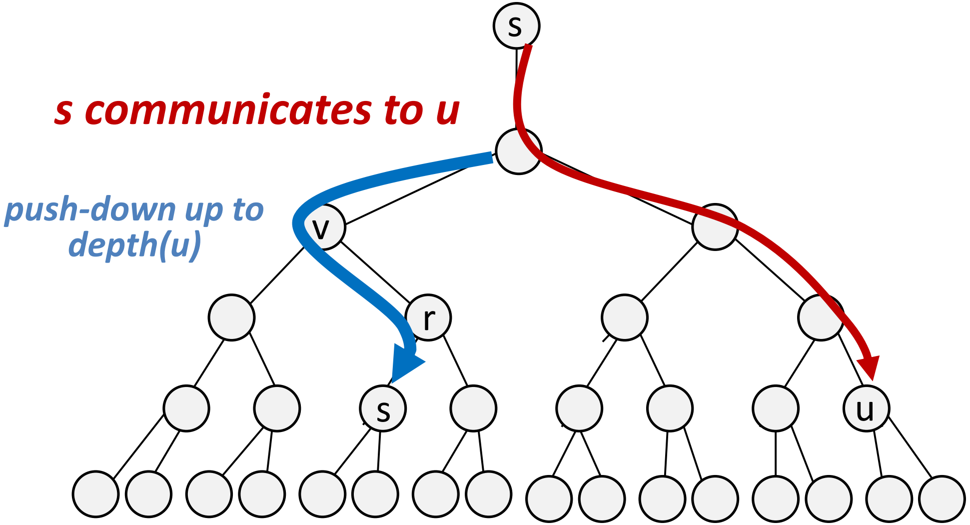


Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

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**

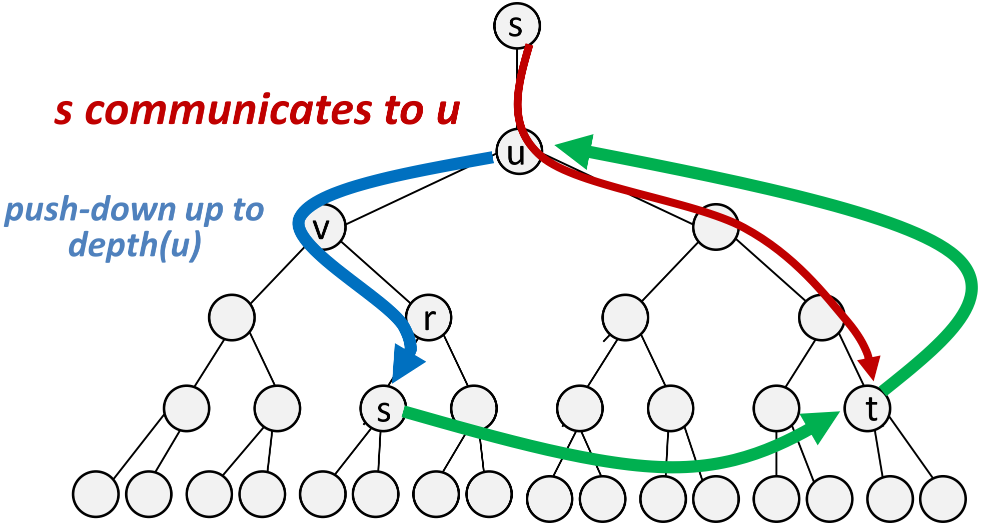


Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

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

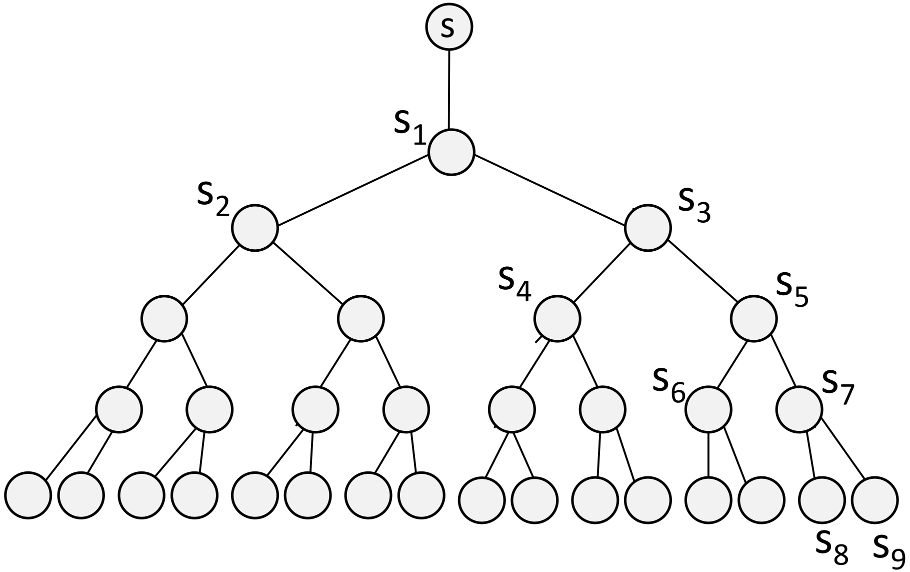
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



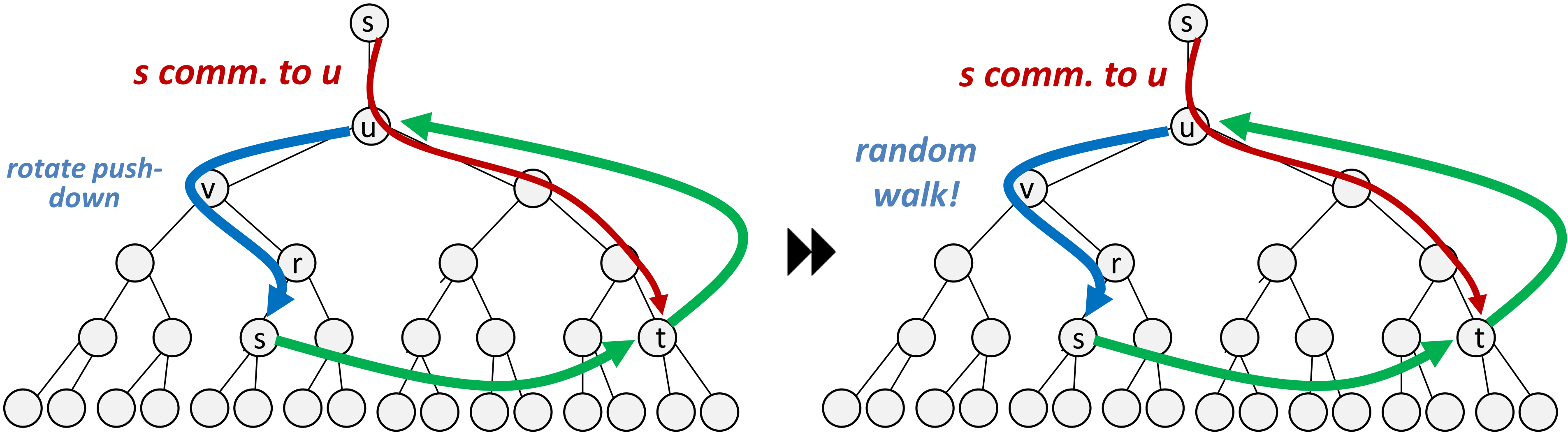
Then: promote u to available root, and t to u: at original depth!

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

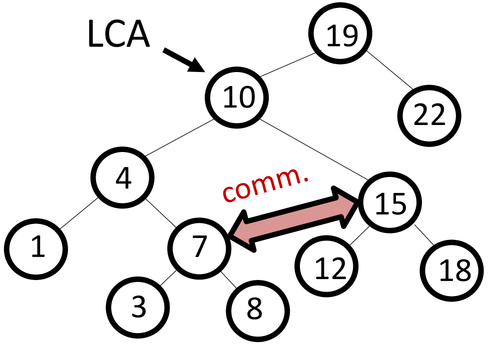
Push-Down Trees: Optimal Self-Adjusting Complete Trees
Chen Avin, Kaushik Mondal, and Stefan Schmid.
ArXiv Technical Report, July 2018.

Roadmap

- Vision and Motivation ✓
- An analogy: coding and datastructures ✓
- Principles of Demand-Aware Network (DAN) Designs ✓
- Principles of Self-Adjusting Network (SAN) Designs ✓
- **Principles of Decentralized Approaches**

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

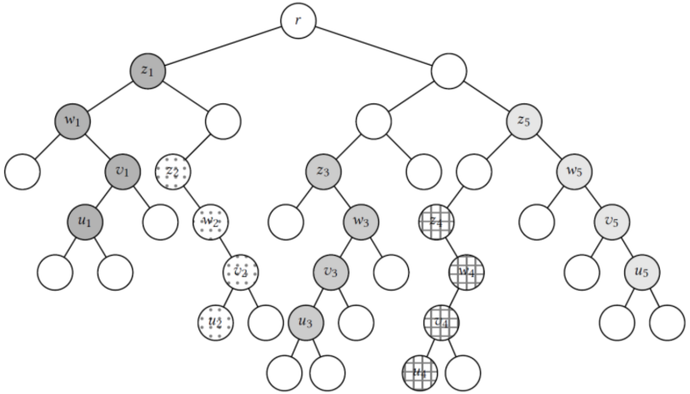
- SplayNet attractive: ordered BST supports **local routing**
  - Nodes ***maintain three ranges***: interval of left subtree, right subtree, upward
- If communicate (frequently): **double-splay** toward LCA
- Challenge: **concurrency**!
  - Access Lemma of splay trees no longer works: ***potential function***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

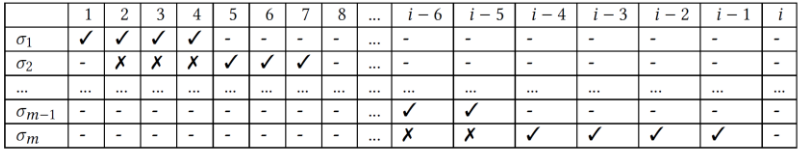


SplayNet

DiSplayNet: Challenges

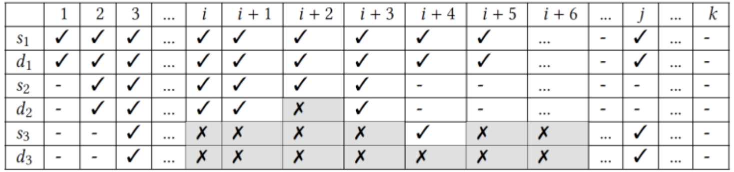
- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:





	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.

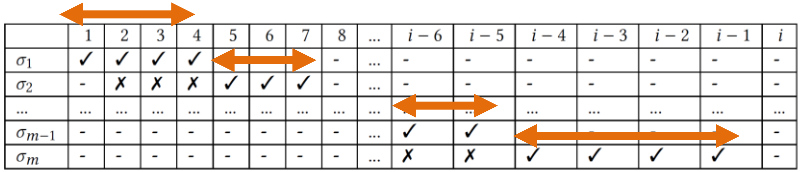
DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***

- To avoid conflict: distributed computation of **independent clusters**

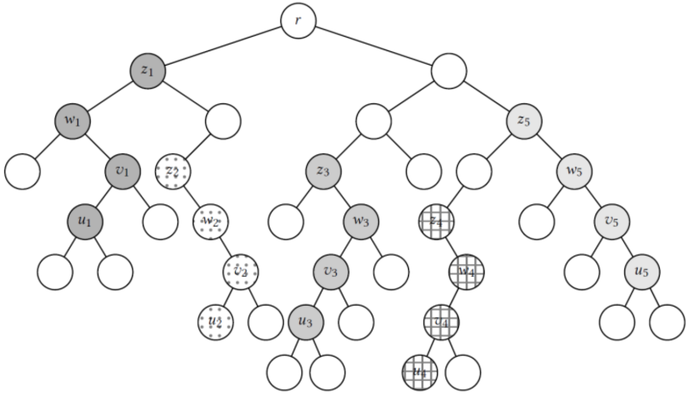
- Still challenging:

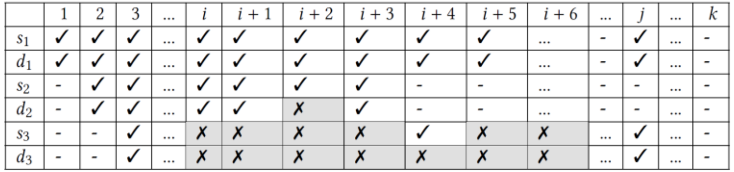
Telescopic: max potential drop



	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

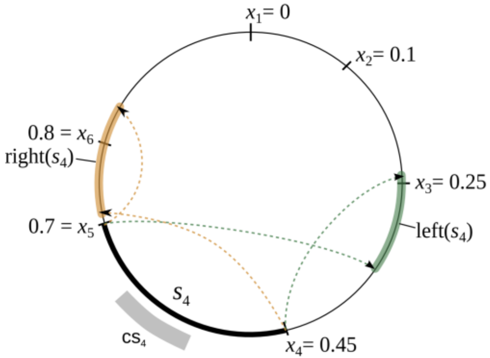
Brief Announcement: Distributed SplayNets
Bruna Peres, Olga Goussevskaia, Stefan Schmid, and Chen Avin.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

Roadmap

- Vision and Motivation ✓
- An analogy: coding and datastructures ✓
- Principles of Demand-Aware Network (DAN) Designs ✓
- Principles of Self-Adjusting Network (SAN) Designs ✓
- Principles of Decentralized Approaches ✓

Many Open Questions

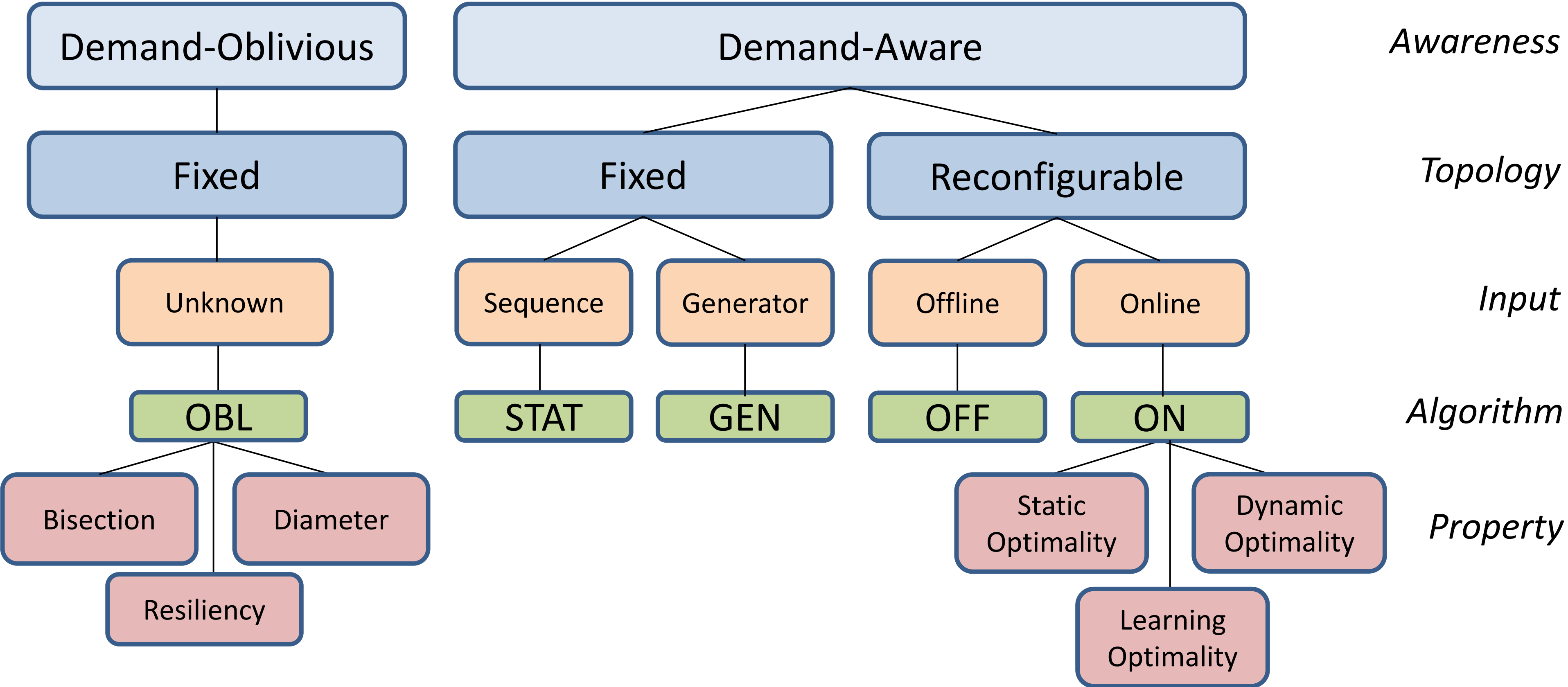
- E.g., robust demand-aware networks?
  - First idea: **rDAN**, based on continuous-discrete approach and Shannon-Fano-Elias coding
- Serving dense communication patterns
- Distributed version
- …



rDAN: Toward Robust Demand-Aware Network Designs. Chen Avin, Alexandr Hercules, Andreas Loukas, and Stefan Schmid. Information Processing Letters (**IPL**), Elsevier, 2018.

Uncharted Landscape!

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



Conclusion

- Networked systems become reconfigurable
- First techniques emerging (e.g., ego-trees)
- How much it helps depends on spatial and temporal locality
- Entropy seems to be a useful metric

Thank you!
Question?

Further Reading

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks
Chen Avin and Stefan Schmid.
ArXiv Technical Report, July 2018.
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.
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.
Chapter 72: Overlay Networks for Peer-to-Peer Networks
Andrea Richa, Christian Scheideler, and Stefan Schmid.
Handbook on Approximation Algorithms and Metaheuristics (**AAM**), 2nd Edition, 2017.