Self-* networks: when flexibility meets algorithms

Stefan Schmid (University of Vienna)
The Trend: Flexibilities
Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany...
Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany...
Flexibilities: Along 3 Dimensions

- **Routing**: Enabler: SDN
- **Embedding**: Enabler: Virtualization
- **Topology**: Enabler: Optics
Another Trend: Improved Visibility of the Networks
Visibility: SDN, Telemetry, Sketching

- Can also improve **security**

- Traditionally: e.g., **trajectory sampling**
  - Sample packets with \( \text{hash(imm. header)} \in [x,y] \)
  - See routes of **some packets**
Visibility: SDN, Telemetry, Sketching

- Can also improve **security**

- Traditionally: e.g., **trajectory sampling**
  - Sample packets with
    \[ \text{hash(imm. header)} \in [x, y] \]
  - See routes of **some packets**
Visibility: SDN, Telemetry, Sketching

- Can also improve security
- Traditionally: e.g., trajectory sampling
  - Sample packets with $\text{hash(imm. header)} \in [x,y]$
  - See routes of some packets
  - Others not! (Usually later...)
Visibility: SDN, Telemetry, Sketching

- Can also improve **security**

- Traditionally: e.g., **trajectory sampling**
  - Sample packets with $\text{hash(imm. header)} \in [x,y]$
  - See routes of **some** packets
  - **Others not!** (Usually later...)

- BSI question: What can we do if switches may be **malicious**?
  - Problem: all switches sample the **same space**: known!
  - Can exploit, e.g., **know when unobserved**.
Visibility: SDN, Telemetry, Sketching

- **Solution:** adversarial trajectory sampling with SDN

- **Idea:**
  - Use **secure** channels between controller and switches to distribute hash ranges
  - Give **different hash ranges** hash ranges to different switches, but add some **redundancy**: risk of being caught!

- In general: obtaining live data from the network **becomes easier**!
Together, Enables A Paradigm Shift: Demand-Aware Networks
Flexibility

Data about Demand

Demand-Aware Networks

Efficiency / Performance
A Case Study: Flexible Topologies
Enabling optical technologies for reconfigurable networks

Example: Manya Ghobadi et al.
*Kudos for some slides!*
Enabling optical technologies for reconfigurable networks

Example: Manya Ghobadi et al.

Kudos for some slides!

Also for WAN!
Example: ProjecToR

- Based on free-space optics

$t=1$
Example: ProjecToR

- Based on free-space optics

$t=2$
Example: ProjecToR

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

Digital Micromirror Devices (DMDs)
Example: ProjecToR

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

\[ t=2 \]
ProjecToR in More Details: Technological Enabler
ProjecToR in More Details: DMDs

- Each micromirror can be turned on/off
- Essentially a **0/1-image**: e.g., array size 768 x 1024
- Direction of the diffracted light can be finely tuned
ProjecToR in More Details:
DMDs to Redirect Light *Fast*
ProjecToR in More Details: DMDs to Redirect Light *Fast*

**Challenge:** limited angular range +/- 3°
ProjecToR in More Details: Coupling DMDs with angled mirrors

**Coupling:** point the DMDs toward a “disco-ball” mirror assembly installed overhead.

Assembly’s angled facets *magnify* the DMD’s reach to the entire DC.
ProjecToR in More Details:
Coupling DMDs with angled mirrors

60x higher fan-out (can directly connect *all* pairs) and 2500x faster switching time than optical circuit switches
Other Technologies

Based on silicon photonics
2-NEMS
Rotating disks

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

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

---

**Survey of Reconfigurable Data Center Networks.** Foerster and Schmid. SIGACT News, 2019.
When Are Demand-Aware Networks Useful?
A Simple Answer

Demand-Oblivious Networks =
Seriously: We believe, often, in practice!

In theory: traffic matrix uniform and static

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

In practice: skewed and dynamic

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Empirical Motivation

Observation 1:
- Many rack pairs exchange little traffic
- Only some hot rack pairs are active

Observation 2:
- Some source racks send large amounts of traffic to many other racks

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

How to measure it?
And which types of structures? E.g., temporal structure in addition to non-temporal structure? Tricky!
Often only intuitions in the literature...

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

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

“over 90% bytes flow in elephant flows”
... and it *is* intuitive!

Non-temporal Structure

Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):
Which one has *more structure*?
... and it *is* intuitive!

Non-temporal Structure

Traffic matrix of two different *distributed ML* applications (GPU-to-GPU):
Which one has *more structure*?
Two different ways to generate *same traffic matrix* (same non-temporal structure):
Which one has *more structure*?

... and it *is* intuitive!

Temporal Structure

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

Which one has *more structure*?

... and it *is* intuitive!

**Temporal Structure**

- More bursty
- More random

VS
Two different ways to generate *same traffic matrix* (same non-temporal structure):
Which one has *more structure*?

... and it *is* intuitive!

Temporal Structure

Quite intuitive: but how to define and measure systematically?
The Trace Complexity

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

• Henceforth called the trace complexity

• Simple approximation: „shuffle&compress“
  – Remove structure by iterative randomization
  – Difference of compression before and after randomization: structure
The Trace Complexity

Original src-dst trace

Randomize rows

Randomized columns

Uniform trace

Increasing complexity (systematically randomized)

More structure (compresses better)
The Trace Complexity

Original src-dst trace  Randomize rows  Randomized columns  Uniform trace

- Remove temporal locality
- Break src-dst pairs
- Remove non-temporal locality

Difference in compression?  Difference in compression?  Difference in compression?
The Trace Complexity

Original src-dst trace
Randomize rows
Randomized columns
Uniform trace

Remove temporal locality
Break src-dst pairs
Remove non-temporal locality

Difference in compression?
Difference in compression?
Difference in compression?

Can be used to define a „complexity map“!
The Complexity Map

Complexity Map: Entropy ("complexity") of traffic traces.

More complexity

More structure
The Complexity Map

**Complexity Map**: Entropy ("complexity") of traffic traces.
The Complexity Map

**Complexity Map**: Entropy ("complexity") of traffic traces.

Size = product of entropy

- More complexity
- More structure
- Skewed

Temporal complexity

Non-temporal complexity
Uniform: Today’s datacenters

- Traditional networks are optimized \textit{for the “worst-case”} (all-to-all communication traffic)
- Example, fat-tree topologies: provide \textit{full bisection bandwidth}
The Complexity Map

Good in the worst case but: cannot leverage different temporal and non-temporal structures of traffic traces!
Non-temporal structure could be exploited already with static demand-aware networks! Good in the worst case but: cannot leverage different temporal and non-temporal structures of traffic traces!
To exploit **temporal** structure, need **adaptive demand-aware** ("self-adjusting") networks.

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

**Non-temporal** structure could be exploited already with **static demand-aware networks**!
Observation: different applications feature quite significant (and different!) temporal and non-temporal structures.

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

Goal: Design self-adjusting networks which leverage both dimensions of structure!
Potential gain / tax of self-adjusting networks!

Both structures!

No structure!

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

The Complexity Map

So: How to design networks which exploit this structure? How good can they be? **Metrics** again!
Roadmap

• Entropy: A metric for demand-aware networks?
  – Intuition
  – A lower bound
  – Algorithms achieving entropy bounds

• From static to dynamic demand-aware networks
  – Empirical motivation
  – A connection to self-adjusting datastructures
Roadmap

• Entropy: A metric for demand-aware networks?
  – Intuition
  – A lower bound
  – Algorithms achieving entropy bounds

• From static to dynamic demand-aware networks
  – Empirical motivation
  – A connection to self-adjusting datastructures
A Simple Example
demand matrix:

- e.g., mirrors
- new flexible interconnect
Matches demand

demand matrix:

e.g., mirrors
new flexible interconnect
new demand:

```
e.g., mirrors

new flexible interconnect
```
Matches demand

new demand:

e.g., mirrors
new flexible interconnect
More Formally
### Input: Workload

<table>
<thead>
<tr>
<th>Sources</th>
<th>Destinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2/65</td>
</tr>
<tr>
<td>3</td>
<td>1/13</td>
</tr>
<tr>
<td>4</td>
<td>1/65</td>
</tr>
<tr>
<td>5</td>
<td>1/65</td>
</tr>
<tr>
<td>6</td>
<td>2/65</td>
</tr>
<tr>
<td>7</td>
<td>3/65</td>
</tr>
</tbody>
</table>

### Output: Constant-Degree DAN

Graph representation of the workload distribution.
### Input: Workload

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2/65</td>
<td>1/13</td>
<td>1/65</td>
<td>1/65</td>
<td>2/65</td>
<td>3/65</td>
</tr>
<tr>
<td>2</td>
<td>2/65</td>
<td>0</td>
<td>1/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2/65</td>
</tr>
<tr>
<td>3</td>
<td>1/13</td>
<td>1/65</td>
<td>0</td>
<td>2/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1/65</td>
<td>0</td>
<td>2/65</td>
<td>0</td>
<td>4/65</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1/65</td>
<td>0</td>
<td>3/65</td>
<td>4/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3/65</td>
</tr>
<tr>
<td>7</td>
<td>3/65</td>
<td>2/65</td>
<td>1/13</td>
<td>0</td>
<td>0</td>
<td>3/65</td>
<td>0</td>
</tr>
</tbody>
</table>

### Destinations

Much from 4 to 5.

### Output: Constant-Degree DAN

- From 1 to 7
- From 2 to 6
- From 5 to 4
- From 3 to N

Makes sense to add link!
### Input: Workload

<table>
<thead>
<tr>
<th>Sources</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>(\frac{2}{65})</td>
<td>(\frac{1}{13})</td>
<td>(\frac{1}{65})</td>
<td>(\frac{1}{65})</td>
<td>(\frac{2}{65})</td>
<td>(\frac{3}{65})</td>
</tr>
<tr>
<td>2</td>
<td>(\frac{2}{65})</td>
<td>0</td>
<td>(\frac{1}{65})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(\frac{2}{65})</td>
</tr>
<tr>
<td>3</td>
<td>(\frac{1}{13})</td>
<td>(\frac{1}{65})</td>
<td>0</td>
<td>(\frac{2}{65})</td>
<td>0</td>
<td>0</td>
<td>(\frac{1}{13})</td>
</tr>
<tr>
<td>4</td>
<td>(\frac{1}{65})</td>
<td>0</td>
<td>(\frac{2}{65})</td>
<td>0</td>
<td>(\frac{4}{65})</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>(\frac{1}{65})</td>
<td>0</td>
<td>(\frac{3}{65})</td>
<td>(\frac{4}{65})</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>(\frac{2}{65})</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(\frac{3}{65})</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>(\frac{3}{65})</td>
<td>(\frac{2}{65})</td>
<td>(\frac{1}{13})</td>
<td>0</td>
<td>0</td>
<td>(\frac{3}{65})</td>
<td>0</td>
</tr>
</tbody>
</table>

### Destinations

1 communicates to many.

### Output: Constant-Degree DAN

Bounded degree: route to 7 indirectly.

```plaintext
1 - 2 - 7
1 - 3 - 4 - 6
1 - 5
```
Input: Workload

Destinations

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2/65</td>
<td>1/13</td>
<td>1/65</td>
<td>1/65</td>
<td>2/65</td>
<td>3/65</td>
</tr>
<tr>
<td>2</td>
<td>2/65</td>
<td>0</td>
<td>1/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2/65</td>
</tr>
<tr>
<td>3</td>
<td>1/13</td>
<td>1/65</td>
<td>0</td>
<td>2/65</td>
<td>0</td>
<td>0</td>
<td>1/13</td>
</tr>
<tr>
<td>4</td>
<td>1/65</td>
<td>0</td>
<td>2/65</td>
<td>0</td>
<td>4/65</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1/65</td>
<td>0</td>
<td>3/65</td>
<td>4/65</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2/65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3/65</td>
</tr>
<tr>
<td>7</td>
<td>3/65</td>
<td>2/65</td>
<td>1/13</td>
<td>0</td>
<td>0</td>
<td>3/65</td>
<td>0</td>
</tr>
</tbody>
</table>

4 and 6 don’t communicate…

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

Output: Constant-Degree DAN

N

1 2 3 4 5 6 7
Objective: Expected Route Length

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

- \( \mathcal{D}[p(i, j)] \): joint distribution
- \( DAN \) of degree \( \Delta \)
- path length on \( N \)
- frequency
Remark

- Can represent demand matrix as a demand graph

sparse distribution $\mathcal{D}$

sparse graph $G(\mathcal{D})$
Some Examples

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

• DANs of $\Delta = 2$:
  – E.g., set of lines and cycles
Remark: Hardness Proof
DAN design can be NP-hard

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

Better!
DAN design can be NP-hard

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

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

Do topological flexibilities make problem easier or harder?!
Rewinding the clock of the Internet to a decade ago...
Rewinding the Clock:
Degree-Diameter Tradeoff

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

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

Kudos to: Pedro Casas
Proof Idea

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

$1 \Delta \Delta(\Delta - 1) \ldots$
Is there a better tradeoff in DANs?
Sometimes, DANs can be much better!

Example 1: low-degree demand

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

Just take $\text{DAN} = \text{demand graph}$!
Sometimes, DANs can be much better!

Example 2: skewed demand

If demand is highly skewed, it is also possible to achieve an expected route length of $O(1)$ in a constant-degree DAN.
Sometimes, DANs can be much better!

Example 2: skewed demand

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

E.g., arrange neighbors of node 1 in a Huffman tree!
So on what does it depend?
So on what does it depend?

We argue (but still don’t know!): on the “entropy” of the demand!
Intuition: Entropy Lower Bound
**Lower Bound Idea:**
Leverage Coding or Datastructure

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

<table>
<thead>
<tr>
<th>Sources</th>
<th>Destinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0, 2/65, 1/13, 1/65, 1/65, 2/65, 3/65</td>
</tr>
<tr>
<td>2</td>
<td>2/65, 0, 1/65, 0, 0, 0, 2/65</td>
</tr>
<tr>
<td>3</td>
<td>1/13, 1/65, 0, 2/65, 0, 0, 1/13</td>
</tr>
<tr>
<td>4</td>
<td>1/65, 0, 2/65, 0, 4/65, 0, 0</td>
</tr>
<tr>
<td>5</td>
<td>1/65, 0, 3/65, 4/65, 0, 0, 0</td>
</tr>
<tr>
<td>6</td>
<td>2/65, 0, 0, 0, 0, 0, 3/65</td>
</tr>
<tr>
<td>7</td>
<td>3/65, 2/65, 1/13, 0, 0, 3/65, 0</td>
</tr>
</tbody>
</table>

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

- DAN just for a single (source) node 3

- How good can this tree be? Cannot do better than $\Delta$-ary Huffman tree for its destinations

- Entropy lower bound on ERL known for binary trees, e.g. Mehlhorn 1975

<table>
<thead>
<tr>
<th>Sources</th>
<th>Destinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0, 2/65, 1/13, 1/65, 1/65, 2/65, 3/65</td>
</tr>
<tr>
<td>2</td>
<td>2/65, 0, 1/65, 0, 0, 0, 2/65</td>
</tr>
<tr>
<td>3</td>
<td>1/13, 1/65, 0, 2/65, 0, 0, 1/13</td>
</tr>
<tr>
<td>4</td>
<td>1/65, 0, 2/65, 0, 4/65, 0, 0</td>
</tr>
<tr>
<td>5</td>
<td>1/65, 0, 3/65, 4/65, 0, 0, 0</td>
</tr>
<tr>
<td>6</td>
<td>2/65, 0, 0, 0, 0, 0, 3/65</td>
</tr>
<tr>
<td>7</td>
<td>3/65, 2/65, 1/13, 0, 0, 3/65, 0</td>
</tr>
</tbody>
</table>

An optimal “ego-tree” for this source!
So: Entropy of the *Entire* Demand

- **Proof idea** ($\text{EPL} = \Omega(\Delta \Delta(Y|X))$):
  - Compute *ego-tree* for each source node
  - Take *union* of all ego-trees
  - Violates *degree restriction* but valid lower bound
Do this in **both dimensions**:

\[ \text{EPL} \geq \Omega(\max\{H_\Delta(Y|X), H_\Delta(X|Y)\}) \]
Entropy of the * Entire Demand: Sources and Destinations *

Do this in **both dimensions**: 

$\text{EPL} \geq \Omega(\max\{ H_\Delta(Y|X), H_\Delta(X|Y) \})$
Achieving Entropy Limit: Algorithms
Ego-Trees Revisited

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

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

Can we merge the trees without distortion and keep degree low?
Ego-Trees Revisited

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

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

Can we merge the trees **without distortion** and **keep degree low**?
From Trees to Networks
Taking union of ego-trees results in high degree: 
$u$ and $v$ will appear in many ego-trees

Node $h$ helps edge $(u, v)$ by participating in ego-tree$(u)$ as a relay node toward $v$ and in ego-tree$(v)$ as a relay toward $u$
But: How to design DANs which also leverage *temporal structure*?

Inspiration from *self-adjusting datastructures* again!
Roadmap

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

• From static to dynamic demand-aware networks
  – A connection to self-adjusting datastructures
An Analogy

Static vs dynamic demand-aware networks!?

DANs vs SANs?
An Analogy to Coding

if demand *arbitrary* and *unknown*

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

$log$ diameter

$log$ # bits / symbol

„Coming to the LKN retreat?“
An Analogy to Coding

if demand \textit{arbitrary} and \textit{unknown}

DAN!

if demand \textit{known} and \textit{fixed}

entropy?/symbol

entropy / symbol
An Analogy to Coding

if demand *arbitrary* and *unknown*

if demand *known* and *fixed*

if demand *unknown* but *reconfigurable*

"Coming to the LKN retreat?"

Dynamic DANs: Aka. Self-Adjusting Networks (SANs)!
An Analogy to Coding

if demand *arbitrary* and *unknown*

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

log diameter
log # bits / symbol

**DAN!**

if demand *known* and *fixed*

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

Can exploit **spatial locality**!

**SAN!**

if demand *unknown* but *reconfigurable*

- **dynamic Demand-Aware Nets**
  - dynamic Huffman codes

Additionally exploit **temporal locality**!

„Coming to the LKN retreat?“
An Analogy to Coding

if demand *arbitrary* and *unknown*

<table>
<thead>
<tr>
<th>worst case network: Full BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst case coding: 00, 01, 10, 11</td>
</tr>
</tbody>
</table>

log diameter

log # bits / symbol

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

DAN!

if demand *known*

Can exploit *spatial* locality

"Cheating": need to know demand!

SAN!

if demand *unknown* but reconfigurable

Need online algorithms!

Additionally exploit *temporal locality*!
Analogous to *Datastructures*: Oblivious...

- Traditional, **fixed** BSTs do not rely on any assumptions on the demand
- Optimize for the **worst-case**
- Example demand: 
  \[1, \ldots, 1, 3, \ldots, 3, 5, \ldots, 5, 7, \ldots, 7, \ldots, \log(n), \ldots, \log(n)\]
- Items stored at \(O(\log n)\) from the root, **uniformly** and **independently** of their frequency

Corresponds to *max possible demand!*
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*: $1,\ldots,1,3,\ldots,3,5,\ldots,5,7,\ldots,7,\ldots,\log(n),\ldots,\log(n)$
  - Amortized cost $O(\log \log n)$

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

Oblivious

Demand-Aware

Exploit spatial locality:
empirical entropy $O(\log \log n)$

Self-Adjusting

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

Lookup $O(\log n)$
Analogously for Networks

Oblivious

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

DAN

Exploit spatial locality

SAN

Exploit temporal locality as well

Algorithms for Self-Adjusting Networks

Ego-trees strike back!

From trees to networks!
Total Recall: Ego-Trees!

∀[i] → Ego–Tree
Total Recall: Ego-Trees!

Idea: use our old approach but now let each node *adjust its ego-tree*!
A Dynamic Ego-Tree: Splay Tree
Uncharted Landscape!

Flexibilities and Algorithms: Opportunities and Challenges
Optimizing Individual Routers

Poor IP Routers

Forwarding Information Base (FIB)

<table>
<thead>
<tr>
<th>Prefix</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00* to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01* to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1* to</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

most specific prefix fast?

TCAM memory expensive and power-hungry...

... and requirements grow quickly (e.g., virtualization). IPv6 does not help.
Idea: Represent as Trie

**Good Potential:**

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

represent as trie... \[ \begin{array}{c|c}
00^* & 0 \\
01^* & 0 \\
1^* & 0 \\
\end{array} \]

... and compress it!
Idea: Represent as Trie

But may introduce *update churn*!
Idea: Represent as Trie

But may introduce update churn!

update cost 2: remove + add subtree

update cost 3: remove + add subtree

... and compress it!
An online problem:
1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size
Optimization of Local Fast Failover


Responsibilities of a Sysadmin

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

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Non-reachability**: Is it ensured that traffic originating from A never reaches B?
- **Waypoint ensurance**: Is it ensured that traffic from A to B is always routed via a node C (e.g., a firewall)?
Responsibilities of a Sysadmin

Sysadmin responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
Responsibilities of a Sysadmin

Sysadmin responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
Responsibilities of a Sysadmin

**Sysadmin** responsible for:

- **Reachability**: Can traffic from ingress port A reach egress port B?
- **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?
- **Policy**: Is it ensured that traffic from A to B never goes via C?

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

Policy ok?
Responsibilities of a Sysadmin

**Sysadmin** responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C?
Responsibilities of a Sysadmin

Sysadmin responsible for:

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

• **Loop-freedom**: Are the routes implied by the forwarding rules loop-free?

• **Policy**: Is it ensured that traffic from A to B never goes via C?

• **Waypoint enforcement**: Is it ensured that traffic from A to B is always routed via a node C?

... and everything even under multiple failures?!
Can we automate such tests or even self-repair?
Can we automate such tests or even self-repair?

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

MPLS configurations, Segment Routing etc.

Compilation

Interpretation

What if...?!

pX ⇒ qXX
pX ⇒ qYX
qY ⇒ rYY
rY ⇒ r
rX ⇒ pX

Pushdown Automaton and Prefix Rewriting Systems Theory
Leveraging Automata-Theoretic Approach

MPLS configurations, Segment Routing etc.

What if...?!

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

```
| pX ⇒ qXX |
| pX ⇒ qYX |
| qY ⇒ rYY |
| rY ⇒ r |
| rX ⇒ pX |
```

Compilation

Interpretation

MPLS configurations, Segment Routing etc.

Pushdown Automaton and Prefix Rewriting Systems Theory
A Complex and Big Formal Language! Why Polynomial Time?!

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

- Even if we reduce to push-down automaton: simple operations such as emptiness testing or intersection on Push-Down Automata (PDA) is computationally non-trivial and sometimes even undecidable!
A Complex and Big Formal Language! Why Polynomial Time?!

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

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

This is not how we will use the PDA!
A Complex and Big Formal Language! Why Polynomial Time?!

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

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

The words in our language are sequences of pushdown stack symbols, not the labels of transitions.
Time for Automata Theory (from Switzerland)!

• Classic result by Büchi 1964: the set of all reachable configurations of a pushdown automaton $a$ is regular set

• Hence, we can operate only on Nondeterministic Finite Automata (NFAs) when reasoning about the pushdown automata

• The resulting regular operations are all polynomial time
  • Important result of model checking

Julius Richard Büchi
1924-1984
Swiss logician
Part 1: Parses query and constructs Push-Down System (PDS)
- In Python 3

Part 2: Reachability analysis of constructed PDS
- Using Moped tool

Regular query language:
\[ k <a> b <c> \]

query processing flow
Traversal test with $k=2$: Can traffic starting with $[]$ go through $s_5$, under up to $k=2$ failures?

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

YES! (Gives witness!)
Formal methods are nice (give guarantees!)... But what about ML...?!
Speed Up Further and Synthesize: Deep Learning (s. talk by Fabien Geyer)

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

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


Virtual switches reside in the server’s virtualization layer (e.g., Xen’s Dom0). Goal: provide connectivity and isolation.
Security of vSwitch

Number of parsed high-level protocols constantly increases...
Security of vSwitch

Virtual Switch

VM
VM
VM

User
Kernel

L2, L2.5, L3, L4

Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6 HOPOPTS
IPv6 ROUTING
IPv6 Fragment
IPv6 DESTOPT
IPv6 ESP
IPv6 AH
RARP
IGMP
Security of vSwitch
Security of vSwitch
Security of vSwitch

Diagram showing the security aspects of a virtual switch (vSwitch) in a virtual machine (VM) environment. The diagram illustrates the relationship between the user, kernel, and virtual switch, highlighting potential security vulnerabilities.

Key components:
- **Virtual Switch**: Central to the diagram, connecting VMs and the kernel.
- **User**: Represents the non-privileged users operating within the VMs.
- **Kernel**: Represents the privileged kernel that provides services to the virtual machines.
- **VMs**: Virtual machines connected to the virtual switch.
- **Ctrl**: Likely represents the control plane, possibly involving management or supervisory functions.

The diagram uses arrows and labels to depict flows and connections, indicating potential security pathways and points of vulnerability.
Security of vSwitch

Virtual Switch

User

Kernel

VM

Ctrl

User

Virtual Switch

Kernel

VM

VM

VM

User

Kernel

VM

VM

VM

User

Kernel

VM

VM

VM

32
Thank you! Questions?
Further Reading

Demand-aware networks

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

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks (Editorial)
Chen Avin and Stefan Schmid.

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

Demand-Aware Network Design with Minimal Congestion and Route Lengths
Chen Avin, Kaushik Mondal, and Stefan Schmid.

Distributed Self-Adjusting Tree Networks
Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid.

Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks
Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anais Villedieu.

IFIP Networking, Warsaw, Poland, May 2019.

DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks
Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu.

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

SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.

Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
**Further Reading**

**What-if analysis**

**P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures**
14th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Heraklion/Crete, Greece, December 2018.

**Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks**
Stefan Schmid and Jiri Srba.
37th IEEE Conference on Computer Communications (INFOCOM), Honolulu, Hawaii, USA, April 2018.

**Secure sampling and dataplane**

**Preacher: Network Policy Checker for Adversarial Environments**
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
38th International Symposium on Reliable Distributed Systems (SRDS), Lyon, France, October 2019.

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

**Taking Control of SDN-based Cloud Systems via the Data Plane** (Best Paper Award)
ACM Symposium on SDN Research (SOSR), Los Angeles, California, USA, March 2018.
Backup Slides
How Predictable is Traffic?

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

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

ML workload (GPU to GPU):
deep convolutional neural network

*Predictable from their dataflow graph*