From Demand-Aware Networks (DANs) to Self-Adjusting Networks (SANs)

Stefan Schmid et al., most importantly: Chen Avin
From Demand-Aware Networks (DANs) to Self-Adjusting Networks (SANs)

Stefan Schmid et al., most importantly: Chen Avin

New in Austria, looking for collaborations etc. 😊

Nice to meet you!
A Brief Overview

• Vision and mission: Make networked systems self-*
  – Self-repairing
  – Self-stabilizing
  – Self-adjusting

• Using different methodologies
  – Algorithms and analysis (LPs, online/approx. algorithms, etc.)
  – Machine-learning (data-driven and “self-driving” networks)
  – Formal methods (e.g., automata theory and synthesis)
A Brief Overview

monitor

react
Example: Fast Reroute in MPLS Networks

Original Routing

One failure: push 30: route around (v₂, v₃)
Two failures:

First push 30: route around \((v_2, v_3)\)

Recursively push 40: route around \((v_2, v_6)\)
Polynomial-Time What-if Analysis

What if...?

MPLS configurations, etc.

Symbol Table

Compilation

Prefix Rewriting System and Push-Down Automata Theory

Interpretation

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

What if...?
Polynomial-Time What-if Analysis

What if...?

Polynomial-time (arbitrary failures)!

Büchi

MPLS configurations, etc.

Compilation

\[ pX \Rightarrow qXX \]
\[ pX \Rightarrow qYX \]
\[ qY \Rightarrow rYY \]
\[ rY \Rightarrow r \]
\[ rX \Rightarrow pX \]

Prefix Rewriting System and Push-Down Automata Theory

IEEE INFOCOM 2018
Software-Defined Networks

Traditional networks: algorithms and functionality fixed, blackbox

Software-defined networks: bring your own algorithm, match-action (formally verify)
Software-Defined Networks

• Software-defined network and network virtualization: networks become software and open
  – “the Linux of networking”

• Challenges:
  – More expressive forwarding: match-action on Layer-2 to Layer-4
  – Complex verification
Consistent Network Updates
Consistent Network Updates

Flow 1
Consistent Network Updates

Can you find an update schedule?
Consistent Network Updates

e.g., cannot update red: congestion! Need to update blue first!

Can you find an update schedule?

Flow 1
Flow 2
Consistent Network Updates

Round 1: prepare

Schedule:
1. red@w, blue@u, blue@v
Consistent Network Updates

Round 2

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
Consistent Network Updates

Round 3

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s

Capacity 2: ok!
No flow!
Consistent Network Updates

Round 4

Capacity 2: ok!

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s
4. blue@w
Consistent Network Updates

Round 4

Note: this (non-trivial) example was just a DAG, without loops!

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s
4. blue@w
Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.

Flow 1
Flow 2
Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.

Just one red block: r1
Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.

Two blue blocks: \textcolor{blue}{b1} and \textcolor{blue}{b2}
Block Decomposition and Dependency Graph

Block for a given flow: subgraph between two consecutive nodes where old and new route meet.

Dependencies: update b2 after r1 after b1.
Many Open Problems

• We know for DAG:
  • For k=2 flows, polynomial-time algorithm to compute schedule with minimal number of rounds!
    • For general k, NP-hard
  • For general k flows, polynomial-time algorithm to compute feasible update

• Everything else: unknown!
  • In particular: what if flow graph is not a DAG?
Trend: Virtualization

- Routers, switches, **middleboxes** run on commodity x86 hardware
  - A.k.a. **virtual switches**
- Mainly in datacenters
  - Uncharted security landscape!
Virtual Switches are Complex, e.g.: (Unified) Packet Parsing
Compromising the Cloud
Compromising the Cloud
Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)
Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)

Started as a theoretical project, but then:

t=1
Today: Demand-Aware Networks (DANs) and Self-Adjusting Networks (SANs)

Started as a theoretical project, but then:

t=2
Today (still): Static Networks

- Traditional datacenter networks are **static**
  - Lower bounds and undesirable **trade-offs**, e.g., degree vs diameter
  - Usually optimized for the “worst-case” (all-to-all communication)
  - Example, fat-tree topologies: provide **full bisection bandwidth**
Next: Reconfigurable Networks?

• The physical topology becomes **reconfigurable**
  – Enables demand-aware network designs
  – Example: **ProjecToR** (SIGCOMM 2016)
Our Research Vision: Demand-Aware Networks (DANs)

Demand matrix: joint distribution

DAN (of constant degree)
Our Research Vision: Demand-Aware Networks (DANs)

Can be seen as a graph as well: the 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>2/65</td>
<td>1</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>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>0</td>
<td>4/65</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>

Demand matrix: joint distribution

DAN (of constant degree)
Our Research Vision: Demand-Aware Networks (DANs)

**Demand matrix**: joint distribution

**DAN** (of constant degree)

```
<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</td>
<td>1</td>
<td>1/65</td>
<td>2/3</td>
</tr>
<tr>
<td>2</td>
<td>2/65</td>
<td>0</td>
<td>1/65</td>
<td>1/65</td>
<td>1/65</td>
<td>2</td>
<td>0</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>1/13</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1/65</td>
<td>1/65</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>0</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>
```

Much from 4 to 5.

Makes sense to add link.

design
Our Research Vision: Demand-Aware Networks (DANs)

Demand matrix: joint distribution

DAN (of constant degree)
Our Research Vision:
Demand-Aware Networks (DANs)

Demand matrix: joint distribution

Sources

<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>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>0</td>
<td>3/65</td>
</tr>
</tbody>
</table>

DAN (of constant degree)

4 and 6 don’t communicate...

... but „extra“ link still makes sense: not a subgraph!
Our Research Vision:
Or Even Self-Adjusting Networks (SANs)

Demands may change over time

Adjust

\[
t=1
\]

\[
t=2
\]
Our Research Vision:
Or Even Self-Adjusting Networks (SANs)

Demands may change over time

Adjust

How to minimize reconfigurations?
How to keep it locally routable?

$t=1$

$t=2$
Our Research Vision: An Analogy to Coding

structure: static / future demand: unknown

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

static / known

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

dynamic / unknown

dynamic Demand-Aware Nets
dynamic Huffman codes
Our Research Vision: An Analogy to Coding

DANs

structure: static / future demand: unknown

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

static / known

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

SANs
dynamic / unknown
dynamic Demand-Aware Nets
dynamic Huffman codes
Relationship to Coding: Example

- Instead of optimizing worst-case performance, leverage information about specific communication pattern to design better networks.
  - Example: Huffman Coding.
DAN: Relationship to...

Sparse, low-distortion graph spanners

- Similar: keep distances in a „compressed network“ (few edges)

- But:
  - We only care about path length *between communicating nodes*, not all node pairs
  - We want *constant degree*
  - Not restricted to subgraph but can have „*additional links“* (like geometric spanners)
DAN: Relationship to...

Minimum Linear **Arrangement** (MLA)

– MLA: map guest graph to line (host graph) so that sum of distances is minimal
– DAN similar: if degree bound = 2, DAN is line or ring (or sets of lines/rings)
– *But* unlike “**graph embedding problems**“
  • The host graph is also subject to optimization
  • Does this render the problem simpler or harder?
SAN: Relationship to...

- **Self-adjusting datastructures** like splay trees

- **But:** Requests are „pair-wise“, not only „from the root“

![Splay Tree](image1)

![SplayNet](image2)
Many interesting research questions

• How to design **static** demand-aware networks?

• How much better can demand-aware networks be compared to **demand-oblivious** networks?

• How to design **dynamic** or even **decentralized** self-adjusting demand-aware networks?
Remainder of This Talk

• Insights into Demand-Aware Networks (DANs)  
  DISC 2017

• Insights into Self-Adjusting Networks (SANs)  
  TON 2016

• Some words about migration... 
  DISC 2016

• Conclusion
DANs: The Problem

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

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

Expected Path Length (EPL): Basic measure of efficiency

\[
EPL(\mathcal{D}, \mathcal{N}) = \mathbb{E}_{\mathcal{D}}[d_N(\cdot, \cdot)] = \sum_{(u,v) \in \mathcal{D}} p(u, v) \cdot d_N(u, v)
\]
Bounded Network Design (BND)

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

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

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

- Clique and star have constant EPL but unbounded degree.
- What about a complete binary tree?
  - Degree 3
  - $d_N(u,v) \leq 2 \log n$
  - Hence EPL = $O(\log n)$
- Can we do better than $\log n$?
An Entropy Lower Bound

- EPL related to **entropy**. Intuition:
  - Low entropy: e.g., uniform distribution, not much structure, long paths
  - High entropy: can exploit structure to create topologies with short paths

- **Theorem**: Let $X, Y$ be the marginal distributions of the sources and destinations in $\mathcal{D}$ respectively. Then

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

- **Conditional entropy**: Average uncertainty of $X$ given $Y$
  - $H(X|Y) = \sum_{i=1}^{n} p(x_i, y_j) \log_2(1/p(x_i|y_j))$
Lower Bound: Idea

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

- Build **optimal** \( \Delta \)-ary tree for each source \( i \): entropy lower bound known on \( \text{EPL} \) known for binary trees (Mehlhorn 1975 for BST but proof does not need search property)

- Consider **union** of all trees

- Violates **degree restriction** but valid lower bound
Lower Bound: Idea

Do this in both dimensions:

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

![Table and Diagram]
Upper Bound: Sparse Distributions

- **Real distributions are** **sparse**!
  - E.g., datacentre's traffic shows that demand distributions are sparse

- **Theorem:** $G_\mathcal{D}$ is a sparse graph with **constant average degree** $\Delta_{avg}$, then it is possible to find a DAN $N$ with maximum degree $12\Delta_{avg}$, such that
  \[EPL(\mathcal{D}, N) \leq O(H(Y|X) + H(X|Y))\]
Sparse Distributions: Construction

- Idea similar to lower bound: “union of bounded-degree trees”
  - However, reduce degree: leverage fact that sparse graphs have at least $n/2$ constant-degree nodes (low-degree nodes)
  - Use them as helper nodes between two “large” (i.e., high-degree) nodes
  - Sparse: there are enough helper nodes,
Many Open Questions

• Demand-aware bounded doubling dimension graphs?

• Demand-aware continuous-discrete graphs?
  – Shannon-Fano-Elias coding

• Demand-aware skip graphs?

• …
SANs: Example “SplayNet”

• Recall:

• Also related: Move-to-Front
Desirable Properties

- Bounded degree
- Supports **local routing**
- “Good over time”: account for **reconfiguration** costs
- **Decentralized**
- ...

SplayNet
SAN Idea 1: SplayNet

- Idea: Binary Search Tree (BST) network
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to Least Common Ancestor (LCA)
SAN Idea 1: SplayNet

- Idea: Binary Search Tree (BST) network
- Supports local routing
  - Left child, right child, upward?
- Search preserving reconfigurations like splay trees: zig, zigzag, zigzag
- But splay only to Least Common Ancestor (LCA)
SAN Idea 1: SplayNet

• Idea: Binary Search Tree (BST) network

• Supports local routing
  – Left child, right child, upward?

• Search preserving reconfigurations like splay trees: zig, zigzag, zigzag

• But splay only to Least Common Ancestor (LCA)
SplayNet: Properties

Property 1: Optimal static network can be computed in polynomial-time (dynamic programming)
   - Unlike unordered tree?

1. Define: flow out of interval $I$
   \[
   W_I(v) = \sum_{u \in I} w(u, v) + w(v, u)
   \]

2. Cost of a given tree $T_I$ on $I$:
   \[
   \text{Cost}(T_I, W_I) = \left[ \sum_{u, v \in I} (d(u, v) + 1)w(u, v) \right] + D_I \ast W_I
   \]
   ($D_I$ distances of nodes in $I$ from root of $T_I$)

3. Dynamic program over intervals.

Choose optimal root and add dist to root

Decouple cost to outside: distance to root of $T_I$ only

$I = [1..8]$

$I' = [9..25]$
Property 2: Provides **amortized cost** and **amortized throughput** guarantees

Rotations can happen concurrently: independent clusters

Splay tree: requests one after another

<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>
<th>8</th>
<th>...</th>
<th>i - 6</th>
<th>i - 5</th>
<th>i - 4</th>
<th>i - 3</th>
<th>i - 2</th>
<th>i - 1</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ₁</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>...</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>σ₂</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>...</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>σₘ₋₁</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>σₘ</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>...</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

SplayNet: concurrent

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>i</th>
<th>i + 1</th>
<th>i + 2</th>
<th>i + 3</th>
<th>i + 4</th>
<th>i + 5</th>
<th>i + 6</th>
<th>...</th>
<th>j</th>
<th>...</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>-</td>
<td>✓</td>
<td>...</td>
</tr>
<tr>
<td>d₁</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>-</td>
<td>✓</td>
<td>...</td>
</tr>
<tr>
<td>s₂</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>-</td>
<td>-</td>
<td>...</td>
</tr>
<tr>
<td>d₂</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>...</td>
<td>-</td>
<td>-</td>
<td>...</td>
</tr>
<tr>
<td>s₃</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>...</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>...</td>
<td>✓</td>
<td>...</td>
</tr>
<tr>
<td>d₃</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>...</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>...</td>
<td>✓</td>
<td>...</td>
<td>-</td>
</tr>
</tbody>
</table>

Analysis more challenging: potential function sum no longer **telescopic**. One request can “push-down” another.
Property 3: Converges to optimal network under specific demands

**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.
SplayNet: Improved Lower Bounds

**Interval Cuts 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 $v$:
  $$w_S(u) = \sum_{(u,v) \in E(S,S')} w(u,v)/W(S)$$
- Define entropy of cut and src($S$), dst($S$) distributions accordingly:
  $$\varphi_H(S) = W(S) (H(\text{src}(S)) + H(\text{dst}(S)))$$
- Conductance entropy is lower bound:
  $$\Omega(\phi_H(\mathcal{R}(\sigma)))$$

**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 $v$:
  $$w_S(u) = \sum_{(u,v) \in E(S,S')} w(u,v)/W(S)$$
- Define entropy of cut and src($S$), dst($S$) distributions accordingly:
  $$\varphi_H(S) = W(S) (H(\text{src}(S)) + H(\text{dst}(S)))$$
- Conductance entropy is lower bound:
  $$\Omega(\phi_H(\mathcal{R}(\sigma)))$$

Cost = $\Omega(\max_j \min_{j,l} H(\text{cut}_{\text{in}}(I_j^l)))$
Cost = $\Omega(\max_j \min_{j,l} H(\text{cut}_{\text{out}}(I_j^l)))$
SAN Idea 2: “Splay Tree DAN”

• Recall:
  – DAN: Union of per-node binary (search) trees

• Idea for SAN:
  – Replace per-node BST with per-node splay tree
Another Dimension of Flexibility: Migration

• Reduce communication cost by online *re*partitioning

How to embed communication pattern across \( l=4 \) servers (or racks, pods, etc.) of size \( k=4 \)?
Another Dimension of Flexibility: Migration

• Reduce communication cost by online repartitioning
Another Dimension of Flexibility: Migration

Most communication within cluster (intra-cluster)...

... little inter-cluster communication.

A classic (hard) combinatorial problem!

Communication cost by online repartitioning
Another Dimension of Flexibility: Migration

• Reduce communication cost by online re-partitioning

Communication patterns can change!
Another Dimension of Flexibility: Migration

- Reduce communication cost by online *re*partitioning

Communication patterns can change!

Changes:
- (1,3), (3,4), (2,5)
  - More
- (5,6)
  - Less

Makes sense that nodes 1 and 5 change clusters!
A Simple Model

• A single switch network:
A Simple Model

- A single switch network:
Adversary Models

Weak adversary
• Chooses request distribution $D$
• Requests sampled i.i.d. from $D$
• Cannot react to online algo

Strong adversary
• Can generate arbitrary request sequence $\sigma$
• Knows and can react to online algo

The Crux: Do not know $D$ resp. $\sigma$ ahead of time

Upon each communication request $(u, v)$:
• Migrate $u$ and $v$ together? «Rent-or-buy»:
• Migrate where? $u$ to $v$, $v$ to $u$, both to a third cluster?
• If cluster is full already: what to evict?
Example: Special Case $k=2$

Clusters of size 2: Need to find pairs!
Example: Special Case $k=2$

Clusters of size 2: A new type of online rematching problem!

Clusters of size 2: Need to find pairs!
It is hard to compete under unfair conditions!

- Assume **two clusters**: for offline algorithm they are of size $k$.

- ... whereas online algorithm can use clusters of size $2k - 1$ (augmentation)!

**OFF:**

**ON:**

extra space!
It is hard to compete under !

- Assume two clusters: for offline algorithm they are of size $k$...

- ... whereas online algorithm can use clusters of size $2k-1$ even (augmentation)!

OFF:

ON:

E.g., ON can even collocate all except for one!
It is hard to compete under 🌟!

- Assume **two clusters**: for offline algorithm they are of size $k$...
- ... whereas online algorithm can use clusters of size $2k-1$ even (augmentation)!

For the sake of lower bound, let us restrict the adversary more: can only ask for node pairs taken from a cyclic order: $k$ pairs (resp. links) in total!
It is hard to compete under !

- Assume **two clusters**: for offline algorithm they are of size k...
- ... whereas online algorithm can use clusters of size 2k-1 even (augmentation)!

**OFF:**

**ON:**

What is the cost of OFF?

Adversary can always request an inter-cluster link: always exists!

Ouch! Cost 1 for each request.

Note: adversarial strategy only depends on ON. So ON cannot learn anything about OFF!
It is hard to compete under $\Delta$!

- Assume **two clusters**: for offline algorithm they are of size $k$...
- ... whereas online algorithm can use clusters of size $2k-1$ even (augmentation)!

**OFF:**

Move to configuration $i \in \{1,\ldots,k\}$ which is asked the least.
Averaging argument: At least $k$ times less communication cost!

**ON:**

Adversary can always request an inter-cluster link: always exists!

Extra space!

Lower bound of $\Omega(k)$ for competitive ratio, despite big augmentation!

Ouch! Cost 1 for each request.
A Simple $O(n^2)$ Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of $n$ nodes
A Simple $O(n^2)$ Upper Bound

Algorithm

• Based on «growing communication components»
• Cycles through phases
  • Initially in each phase: empty graph of $n$ nodes
  • For each inter-cluster request for ON: insert edge
A Simple O(n^2) Upper Bound

Algorithm

• Based on «growing communication components»
• Cycles through phases
  • Initially in each phase: empty graph of n nodes
  • For each inter-cluster request for ON: insert edge
  • Induces a «communication component»: edge weight = # requests
A Simple $O(n^2)$ Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of $n$ nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests
  - If an edge $(u,v)$ weight reaches $\alpha$, DET repartitions nodes, so that all edges which have reached $\alpha$ so far are in same cluster!
A Simple O(n²) Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of n nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests
  - If an edge (u,v) weight reaches α, DET repartitions nodes, so that all edges that have reached α so far are in same cluster.
  - If this is not possible: phase ends

Components cannot be partitioned perfectly (first component alone too large)!
A Simple O(n²) Upper Bound

Algorithm

- Based on «growing communication components»
- Cycles through phases
  - Initially in each phase: empty graph of n nodes
  - For each inter-cluster request for ON: insert edge
  - Induces a «communication component»: edge weight = # requests
  - If an edge (u,v) weight reaches α, DET repartitions nodes, so that all edges have reached α so far are in same cluster
  - If this is not possible: phase ends

Components cannot be partitioned perfectly (first component alone too large)!
Analysis (costs per phase):

- Observe: edge weights always $\leq \alpha$: once reach $\alpha$, their endpoints will always be collocated (by algorithm definition)
A Simple $O(n^2)$ Upper Bound

Analysis (costs per phase):

- Observe: edge weights always $\leq \alpha$: once reach $\alpha$, their endpoints will always be collocated (by algorithm definition)
- $\alpha$-edges form a forest (so at most $n$ many!): once two nodes $(u,v)$ are connected by a path of $\alpha$–edges, they are in a single cluster and will no longer communicate across clusters
A Simple $O(n^2)$ Upper Bound

Analysis (costs per phase):

- Observe: edge weights always $\leq \alpha$: once reach $\alpha$, their endpoints will always be collocated (by algorithm definition)
- $\alpha$-edges form a forest (so at most $n$ many!): once two nodes $(u,v)$ are connected by a path of $\alpha$-edges, they are in a single cluster and will no longer communicate across clusters
- Thus: ON cost per phase:
  - At most 1 reorganization per $\alpha$-edge (at most $n$ $\alpha$-edges), so $n$ times reconfig cost $n \cdot \alpha$, so $n^2 \alpha$
  - Communication cost: at most $\alpha$ per edge (at most $n^2$ many), so also at most $n^2 \alpha$
A Simple $O(n^2)$ Upper Bound

Analysis (costs per phase):

- Observe: edge weights always $\leq \alpha$: once reach $\alpha$, their endpoints will always be collocated (by algorithm definition)
- $\alpha$-edges form a forest (so at most $n$ many!): once two nodes $(u,v)$ are connected by a path of $\alpha$–edges, they are in a single cluster and will no longer communicate across clusters
- Thus: ON cost per phase:
  - At most 1 reorganization per $\alpha$-edge (at most $n \alpha$-edges), so $n$ times reconfig cost $n \cdot \alpha$, so $n^2 \alpha$
  - Communication cost: at most $\alpha$ per edge (at most $n^2$ many), so also at most $n^2 \alpha$
- Costs of OFF per phase:
  - If OFF migrates any node, it pays at least $\alpha$
  - If not, it pays communication cost at least $\alpha$: the grown components do not fit clusters (intra-cluster edges only): definition of «end-of-phase»!

Upper bound of $O(n^2 \alpha / \alpha) = O(n^2)$ for competitive ratio!
Known Results So Far

• Case $k=2$ („online rematching“): constant competitive ratio

• General case: with a little bit of augmentation: $O(k \log k)$ possible
  • Recall $\Omega(k)$ lower bound
  • Nice: independent of number of clusters!
  • Practically relevant: # VM slots per server usually small
Conclusion

• Communication networks become more flexible:
  – Software-defined: bring your own algorithm
  – Topology subject to optimization
  – Placement subject to optimization

• Challenges:
  – Consistent reconfiguration
  – (Dynamic) network design
  – Online migration
Thank You !
References

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.

Congestion-Free Rerouting of Flows on DAGs

Taking Control of SDN-based Cloud Systems via the Data Plane

Demand-Aware Network Designs of Bounded Degree

Online Balanced Repartitioning

SplayNet: Towards Locally Self-Adjusting Networks
What about 🤔?
What about ?

- Recall: weak adversary cannot choose request sequence but only the distribution
  - Adversary needs to sample i.i.d. from this distribution
  - Moreover: Adversary knows (deterministic or randomized) «learning» algorithm, i.e., chooses worst distribution

Any ideas?
The Crux: *Joint* Optimization of Efficient Learning

*and* Searching

- Naive idea 1: Take it easy and first learn distribution
  - Do not move but just sample requests in the beginning: until exact distribution has been learned whp
  - Then move to the best location for good
The Crux: Joint Optimization of Efficient Learning and Searching

- Naive idea 1: Take it easy and first learn distribution
  - Do not move but just sample requests in the beginning: until exact distribution has been learned
  - Then move to the best location for good

Waiting can be very costly: maybe start configuration is very bad and others similarly good: takes long to learn, not competitive! Need to move early on, away from bad locations!
The Crux: *Joint* Optimization of Efficient Learning *and* Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned *whp*
  - Then move to the best location *for good*

- **Naive idea 2: Pro-actively always move to the lowest cost configuration seen so far**
The Crux: *Joint* Optimization of Efficient Learning *and* Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned *whp*
  - Then move to the best location for good

- **Naive idea 2: Pro-actively always move to the lowest cost configuration seen so far**

  Bad: if requests are uniform at random, you should not move at all! Migration costs cannot be amortized. Crucial difference to classic distribution learning problems: guessing costs!
The Crux: *Joint* Optimization of Efficient Learning and Searching

- **Naive idea 1: Take it easy and first learn distribution**
  - Do not move but just sample requests in the beginning: until exact distribution has been learned.
  - Then move to the best location for good performance.

- **Naive idea 2: Pro-actively always move to the lowest cost configuration**
  - Bad, e.g., if requests are distributed uniformly at random: better not to move at all (moving costs cannot be amortized).

- Only move when it pays off! But e.g., how to differentiate between uniform and “almost uniform” distribution?
Example Learning Algorithm for Ring: Rotate Locally!

- Mantra of our algorithm: Rotate!
  - Rotate early, but not too early!
  - And: rotate **locally**
Example Learning Algorithm for Ring:
Rotate Locally!

- Mantra of Algorithm: Rotate!
  - Rotate early, but not too early!
  - And: rotate locally

Define **conditions** for configurations: if met, **never go back** to it (we can afford it w.h.p.: seen enough samples)
Example Learning Algorithm for Ring:
Rotate Locally!

- Mantra of our algorithm: Rotate!
  - Rotate early, but not too early!
  - And: rotate locally

If current configuration is eliminated, go to nearby configuration (in directed manner: no frequent back and forth)!
Mantra of our algorithm: Rotate!
- Rotate early, but not too early!
- And: rotate locally

Growing radius strategy: allow to move further only once amortized!
Example Learning Algorithm for Ring:
Rotate Locally!

- Mantra of our algorithm: Rotate!
- Rotate early, but not too early!
- And: rotate locally
  - If current configuration is eliminated, go to nearby configuration (in directed manner: no frequent back and forth)!

log(n)-competitive w.h.p.
Future work

• More general graphs: regular/maximum degree $n^{1/r}$, for any $r$.

• Do we require alternate flavours of graph entropy?

• Maintaining the bounded degree network dynamically.
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.

rDAN: Toward Robust Demand-Aware Network Designs
Chen Avin, Alexandr Hercules, Andreas Loukas, and Stefan Schmid.

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