Self-Adjusting Networks
Stefan Schmid

“We cannot direct the wind, but we can adjust the sails.”
(Folklore)
Trend
Data-Centric Applications

Datacenters ("hyper-scale")

Interconnecting networks: a critical infrastructure of our digital society.

Source: Facebook
Trend
Data-Centric Applications

Datacenters (“hyper-scale”)

Interconnecting networks: a critical infrastructure of our digital society.

Credits: Marco Chiesa
The Problem
Huge Infrastructure, Inefficient Use

Network equipment reaching capacity limits
- Transistor density rates stalling
- "End of Moore's Law in networking"

Hence: more equipment, larger networks

Resource intensive and: **inefficient**

Annoying for companies, **opportunity** for researchers!

[1] Source: Microsoft, 2019
Root Cause

Fixed and Demand-Oblivious Topology

How to interconnect?
Root Cause

Fixed and Demand-Oblivious Topology

Many flavors, but in common: fixed and oblivious to actual demand.
Root Cause

Fixed and Demand-Oblivious Topology

Highway which ignores actual traffic: frustrating!

Many flavors, but in common: fixed and oblivious to actual demand.
A Vision
Flexible and Demand-Aware Topologies
A Vision
Flexible and Demand-Aware Topologies
A Vision
Flexible and Demand-Aware Topologies

e.g., mirrors
new flexible interconnect

demand matrix:
A Vision

Flexible and Demand-Aware Topologies

Matches demand

demand matrix:

1 2 3 4 5 6 7 8

1 8

2 7

3 6

4 5

e.g., mirrors

new flexible interconnect
A Vision
Flexible and Demand-Aware Topologies

e.g., mirrors
new flexible interconnect

new demand:
A Vision
Flexible and Demand-Aware Topologies

Matches demand:

new demand:

e.g., mirrors

new flexible interconnect
A Vision
Flexible and Demand-Aware Topologies

Self-Adjusting Networks

new demand:

- e.g., mirrors
- new flexible interconnect
The Motivation
Much Structure in the Demand

Empirical studies:
traffic matrices sparse and skewed

The hypothesis: can be exploited.

traffic bursty over time
Recent Representation of Trace Structure:

Complexity Map

Griner et al., SIGMETRICS 2020
Recent Representation of Trace Structure:

**Complexity Map**

Different structures!
Traffic is also clustered:

**Small Stable Clusters**

Opportunity: *exploit* with little reconfigurations!
Sounds Crazy?
Emerging Enabling Technology.

H2020:
“Photonics one of only five key enabling technologies for future prosperity.”

US National Research Council:
“Photons are the new Electrons.”
Spectrum of prototypes

- Different sizes, different reconfiguration times
- From our ACM SIGCOMM workshop OptSys
Enabler

Novel Reconfigurable Optical Switches

→ **Spectrum** of prototypes
  → Different sizes, different reconfiguration times
  → From our ACM SIGCOMM workshop OptSys

- Prototype 1
  - Moving antenna (ms)

- Prototype 2
  - Moving mirrors (mus)

- Prototype 3
  - Changing lambdas (ns)
Example

Optical Circuit Switch

→ Optical Circuit Switch rapid adaption of physical layer
  → Based on rotating mirrors

Optical Circuit Switch
By Nathan Farrington, SIGCOMM 2010
First Deployments

E.g., Google
The Big Picture

Flexibility

Structure

Self-Adjusting Networks

Efficiency

New!

More!

Now is the time!
The Big Picture

Flexibility: New!

Structure: More!

Self-Adjusting Networks: Now is the time!

Efficiency

Potential Gain

- bursty & skewed
- non-temporal complexity
- temporal complexity

- pF
- CNS
- ML
- DB
- Web
- Had
- Multi Grid
- NN

Potential Gain: 26
Potential Gain

- non-temporal complexity
- temporal complexity
- bursty & skewed
- bursty
- uniform
- skewed

- pF
- CNS
- ML
- DB
- Web
- Had
- Multi Grid
- NN

Potential Gain: 27
Unique Position

Demand-Aware, Self-Adjusting Systems

Everywhere, but mainly in software

Algorithmic trading

Recommender systems

Neural networks

Our focus: in hardware
The Natural Question:

Given This Structure, What Can Be Achieved? Metrics and Algorithms?

A first insight: entropy of the demand.
Case Study “Route Lengths”

**Constant-Degree Demand-Aware Network**

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

\[
ERL(\mathcal{D},N) = \sum_{(u,v) \in \mathcal{D}} p(u,v) \cdot d_N(u,v)
\]
### Case Study “Route Lengths”

#### Constant-Degree Demand-Aware Network

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

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

Expected Route Length
Case Study “Route Lengths”

Constant-Degree Demand-Aware Network

\[
\text{ERL}(\mathcal{O}, N) = \sum_{(u,v) \in \mathcal{O}} p(u,v) \cdot d_N(u,v)
\]
Case Study “Route Lengths”

Constant-Degree Demand-Aware Network

\[
\text{ERL}(\mathcal{D},N) = \sum_{(u,v) \in \mathcal{D}} p(u,v) \cdot d_N(u,v)
\]
Case Study “Route Lengths”

**Constant-Degree Demand-Aware Network**

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

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

Huffman tree: “ego-tree”
## Algorithm: Idea

### Huffman tree: “ego-tree”

<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></td>
<td>3/65 2/65 1/13 0 0 3/65 0</td>
</tr>
</tbody>
</table>

### Cost: Entropy!
Idea for algorithm:
- Union of trees
Idea for algorithm:
- Union of trees
- Reduce degree
- But keep distances

Entropy Upper Bound
Idea for algorithm:
→ Union of trees
→ Reduce degree
→ But keep distances

Ok for sparse demands
→ Not everyone gets tree
→ Helper nodes
Idea for algorithm:
- Union of trees
- Reduce degree
- But keep distances

Ok for sparse demands
- Not everyone gets tree
- Helper nodes

Dense?  Congestion?  Dynamic?  Distributed?
Insight:
Connection to Datastructures

Traditional BST  Demand-aware BST  Self-adjusting BST

More structure: improved access cost
Insight:

Connection to Datastructures & Coding

Traditional BST (Worst-case coding)
Demand-aware BST (Huffman coding)
Self-adjusting BST (Dynamic Huffman coding)

More structure: improved access cost / shorter codes
Insight: Connection to Datastructures & Coding

Traditional BST (Worst-case coding)
Demand-aware BST (Huffman coding)
Self-adjusting BST (Dynamic Huffman coding)

More structure: improved access cost / shorter codes

Similar benefits?
Insight:

Connection to Datastructures & Coding

Traditional BST (Worst-case coding)
Demand-aware BST (Huffman coding)
Self-adjusting BST (Dynamic Huffman coding)

More structure: improved access cost / shorter codes

More than an analogy!

Similar benefits?
Insight: Connection to Datastructures & Coding

Traditional BST (Worst-case coding)
Demand-aware BST (Huffman coding)
Self-adjusting BST (Dynamic Huffman coding)

More than an analogy!

Generalize methodology: ... and transfer entropy bounds and algorithms of data-structures to networks.

First result: Demand-aware networks of asymptotically optimal route lengths.
Related Problem

Virtual Network Embedding Problem (VNEP)

Example $\Delta=2$: A Minimum Linear Arrangement (MLA) Problem
→ Minimizes sum of virtual edges
Related Problem

Virtual Network Embedding Problem (VNEP)

Example $\Delta=2$: A Minimum Linear Arrangement (MLA) Problem
→ Minimizes sum of virtual edges
Related Problem

Virtual Network Embedding Problem (VNEP)

Example $\Delta=2$: A Minimum Linear Arrangement (MLA) Problem

$\rightarrow$ Minimizes sum of virtual edges
Example $\triangle=2$: A Minimum Linear Arrangement (MLA) Problem
→ Minimizes sum of virtual edges

MLA is \textbf{NP-hard}
→ ... and so is our problem!
Related Problem

Virtual Network Embedding Problem (VNEP)

Example $\Delta=2$: A Minimum Linear Arrangement (MLA) Problem
   $\rightarrow$ Minimizes sum of virtual edges

MLA is NP-hard
   $\rightarrow$ ... and so is our problem!

But what about $\Delta>2$?
   $\rightarrow$ Embedding problem still hard
   $\rightarrow$ But we have a new degree of freedom!
Example $\Delta = 2$: A Minimum Linear Arrangement (MLA) Problem
→ Minimizes sum of virtual edges

MLA is \textbf{NP-hard}
→ ... and so is our problem!

But what about $\Delta > 2$?
→ Embedding problem still hard
→ But we have a new degree of freedom!

Simplifies problem?!
Self-adjusting networks may be really useful to serve large flows (elephant flows): avoiding multi-hop routing.
Self-adjusting networks may be really useful to serve large flows (*elephant flows*): avoiding multi-hop routing.
Self-adjusting networks may be really useful to serve large flows (elephant flows): avoiding multi-hop routing. However, requires optimization and adaption, which takes time.
Self-adjusting networks may be really useful to serve large flows (elephant flows): avoiding multi-hop routing

- However, requires optimization and adaption, which takes time
Indeed, it is more complicated than that…

**Challenge: Traffic Diversity**

**Diverse patterns:**
- Shuffling/Hadoop: all-to-all
- All-reduce/ML: ring or tree traffic patterns
  - Elephant flows
- Query traffic: skewed
  - Mice flows
- Control traffic: does not evolve but has non-temporal structure

**Diverse requirements:**
- ML is **bandwidth** hungry,
  - small flows are **latency-sensitive**
Opportunity: Tech Diversity

Diverse topology components:
→ demand-\textit{oblivious} and demand-\textit{aware}
Diverse topology components:
→ demand-oblIVious and demand-aware
→ static vs dynamic
Diverse topology components:

→ demand-oblivious and demand-aware
→ static vs dynamic

Opportunity: Tech Diversity

- Demand-oblivious components:
  - e.g., RotorNet (SIGCOMM'17), Opera (NSDI'20), Sirius (SIGCOMM'20)
  - e.g., Clos (SIGCOMM'08), Slim Fly (SC'14), Xpander (SIGCOMM'17)

- Demand-aware components:
  - e.g., FireFly (SIGCOMM'14), ProjecToR (SIGCOMM'16), SplayNet (ToN'16)
Opportunity: Tech Diversity

Diverse topology components:
→ demand-oblivious and demand-aware
→ static vs dynamic
Opportunity: Tech Diversity

Diverse topology components:
→ demand-oblivious and demand-aware
→ static vs dynamic

Which approach is best?
Diverse topology components:
→ demand-oblivious and demand-aware
→ static vs dynamic

Which approach is best?

As always in CS: It depends…
Rack Interconnect

Typical rack internconnect: **ToR-Matching-ToR (TMT) model**
Typical rack internconnect: **ToR-Matching-ToR (TMT) model**
Details: Switch Types

Periodic Switch (aka Rotor Switch)

Rotor switch: periodic matchings (demand-oblivious)

Si:

M1  M2  M3  M1  M2  M3

time
Demand-aware switch: optimized matchings

Si:
Static switches: combine for optimized static topology

S1: M1 
S2: M2 
S3: M3 

e.g., tree, expander
Design Tradeoffs (1)
The "Awareness-Dimension"

Good for all-to-all traffic!
← oblivious: very fast
  periodic direct connectivity
← no control plane overhead

Good for elephant flows!
← optimizable toward traffic
← but slower
Design Tradeoffs (1)

The “Awareness-Dimension”

- **low tax**
  - Rotor
  - Good for all-to-all traffic!
    - oblivious: very fast
      - periodic direct connectivity
    - no control plane overhead

- **high tax**
  - Demand-Aware
  - Good for elephant flows!
    - optimizable toward traffic
    - but slower

Compared to static networks: latency tax!
Design Tradeoffs (2)

The “Flexibility-Dimension”

Good for high throughput!
→ direct connectivity saves bandwidth along links

Good for low latency!
→ no need to wait for reconfigurable links
→ compared to dynamic: bandwidth tax (multi-hop)
Design Tradeoffs (2)

The “Flexibility-Dimension”

Good for high throughput!
→ direct connectivity saves bandwidth along links

Good for low latency!
→ no need to wait for reconfigurable links
→ compared to dynamic: bandwidth tax (multi-hop)
Observation 1: Different topologies provide different tradeoffs.

Observation 2: Different traffic requires different topology types.

Observation 3: A mismatch of demand and topology can increase flow completion times.
Examples:
Match or Mismatch?

Shuffling
ML

Delay sensitive
Telemetry / control

Demand

Static
Rotor
Demand-Aware

Demand-oblivious

Dynamic

Topology
Examples: Match or Mismatch?

- Shuffling
- ML
- Delay sensitive
- Telemetry / control

- Dynamic
- Demand-aware
- Demand-oblivious
- Static

Serving mice flows on demand-aware?
Examples:
Match or Mismatch?

Serving mice flows on demand-aware?
Bad idea! Latency tax.
Examples: Match or Mismatch?

Shuffling  |  ML
---        | ---
Delay sensitive  |  Telemetry / control

Demand   |  Static
---       | ---
Dynamic  |  Demand-Aware

Rotor  |  Demand-aware

Serving elephant flows on static?
Examples: Match or Mismatch?

Serving elephant flows on static?
Bad idea! Bandwidth tax.
We have a first approach:
Cerberus* serves traffic on the “best topology”! (Optimality open)

* Griner et al., ACM SIGMETRICS 2022
On what should topology type depend? We argue: flow size.
Flow Size Matters

On what should topology type depend? We argue: flow size.

→ Observation 1: Different apps have different flow size distributions.
Observation 1: Different apps have different flow size distributions.

Observation 2: The transmission time of a flow depends on its size.

Observation 3: For small flows, flow completion time suffers if network needs to be reconfigured first.

Observation 4: For large flows, reconfiguration time may amortize.
Flow Size Matters

- **Observation 1**: Different apps have different flow size distributions.
- **Observation 2**: The transmission time of a flow depends on its size.
- **Observation 3**: For small flows, flow completion time suffers if network needs to be reconfigured first.
- **Observation 4**: For large flows, reconfiguration time may amortize.
Flow Size Matters

Observation 1: Different apps have different flow size distributions.

Observation 2: The transmission time of a flow depends on its size.

Observation 3: For small flows, flow completion time suffers if network needs to be reconfigured first.

Observation 4: For large flows, reconfiguration time may amortize.
Cerberus

Optical Switches
Cerberus

\[
\begin{array}{c}
K_s \\
\text{static switches} \\
\end{array}
\quad
\begin{array}{c}
K_r \\
\text{rotor switches} \\
\end{array}
\quad
\begin{array}{c}
K_d \\
\text{demand-aware switches} \\
\end{array}
\]
Scheduling: Small flows go via static switches...
Cerberus

Scheduling: ... medium flows via rotor switches...
Scheduling: ... and large flows via demand-aware switches (if one available, otherwise via rotor).
Throughput Analysis

Demand Matrix

$T$

**Metric**: throughput of a demand matrix...
Throughput Analysis

\[ T \times \theta(T) \]

**Metric:** throughput of a demand matrix...

... is the maximal scale down factor by which traffic is feasible.
Throughput Analysis

Metric: throughput of a demand matrix...

... is the maximal scale down factor by which traffic is feasible.

Throughput of network $\theta^*$: worst case $T$
Throughput Analysis

\[ T \times \theta(T) \Rightarrow \]

Demand Matrix

Worst demand matrix for static and rotor: permutation. Best case for demand-aware!
Throughput Analysis

\[ T \times \theta(T) \Rightarrow \]

Demand Matrix

<table>
<thead>
<tr>
<th></th>
<th>expander-net</th>
<th>rotor-net</th>
<th>Cerberus</th>
</tr>
</thead>
<tbody>
<tr>
<td>BW-Tax</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>LT-Tax</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>( \theta(T) )</td>
<td>Thm 2</td>
<td>Thm 3</td>
<td>Thm 5</td>
</tr>
<tr>
<td>( \theta^* )</td>
<td>0.53</td>
<td>0.45</td>
<td>Open</td>
</tr>
<tr>
<td>Datamining</td>
<td>0.53</td>
<td>0.6</td>
<td>0.8 (+33%)</td>
</tr>
<tr>
<td>Permutation</td>
<td>0.53</td>
<td>0.45</td>
<td>1 (+88%)</td>
</tr>
<tr>
<td>Case Study</td>
<td>0.53</td>
<td>0.66</td>
<td>0.9 (+36%)</td>
</tr>
</tbody>
</table>

Worst demand matrix for static and rotor: permutation. Best case for demand-aware!
Throughput Analysis

Demand Matrix

\[ T \times \theta(T) \Rightarrow \]

Worst demand matrix for static and rotor: permutation. Best case for demand-aware!

<table>
<thead>
<tr>
<th></th>
<th>expander-net</th>
<th>rotor-net</th>
<th>Cerberus</th>
</tr>
</thead>
<tbody>
<tr>
<td>BW-Tax</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>LT-Tax</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(\theta(T))</td>
<td>Thm 2</td>
<td>Thm 3</td>
<td>Thm 5</td>
</tr>
<tr>
<td>(\theta^*)</td>
<td>0.53</td>
<td>0.45</td>
<td>Open</td>
</tr>
<tr>
<td>Datamining</td>
<td>0.53</td>
<td>0.6</td>
<td>0.8 (+33%)</td>
</tr>
<tr>
<td>Permutation</td>
<td>0.53</td>
<td>0.45</td>
<td>(\approx 1 (+88%))</td>
</tr>
<tr>
<td>Case Study</td>
<td>0.53</td>
<td>0.66</td>
<td>0.9 (+36%)</td>
</tr>
</tbody>
</table>
Summary

→ Opportunity: *structure* in demand and *reconfigurable* networks

→ How to measure demand? A first metric: *entropy*

→ New algorithmic problem: demand-aware and *self-adjusting graphs*
  ↓ At least for sparse demands we know how
  ↓ *Open questions:* What about general demand? Load? Distributed algorithms? *Hybrid* networks (i.e., demand-aware on top of a fixed Clos topology)?

→ Cerberus aims to assign traffic to its best topology
  ↓ Depending on flow size
  ↓ *Open questions:* Analysis of throughput? Optimality?
“Zukunftsmusik”

→ So far: tip of the iceberg

→ Many more challenges
  → Shock wave through *layers*: impact on routing and congestion control?
  → *Scalability* of control in dynamic graphs: *local algorithms*? Greedy routing?
  → Complexity of demand-aware graphs
    (pure vs hybrid, e.g., SplayNet)
  → *Application-specific* self-adjusting networks:
    e.g., for AI, or similar to *active dynamic networks* (independent sets, consensus, …)
  → etc.

Thank you!
Online Video Course

Invitation to
Self-Adjusting Networks
A short video course

"We cannot direct the wind, but we can adjust the sails."
(Folklore)

Prof. Chen Avin (BGU, Israel)
Prof. Stefan Schmid (TU Berlin, Germany)

https://self-adjusting.net/course
Websites

http://self-adjusting.net/
Project website

https://trace-collection.net/
Trace collection website
Questions?

Golden Gate Zipper
Further Reading

**Static DAN**

**Overview: Models**

**Dynamic DAN**

**Concurrent DANs**

**Static Optimality**

---

**Demand-Aware Network Designs of Bounded Degree**

Chen Avin  
Koushik Mondal  
Stefan Schmid

Abstract Traditionally, networks such as datacenter interconnects are designed to optimize worst-case performance under arbitrary traffic patterns. Such network designs can however be far too optimistic when considering the actual workload and traffic patterns which they will see. This insight led to the development of demand-aware network designs which can be reconciled with the actual traffic patterns.

Motivated by these ideas, this paper initiates the study of demand-aware network designs, and in particular the design of bounded-degree multicast network designs.

**Introduction**

The problem studied in this paper is motivated by the advent of more flexible datacenter interconnects, such as Project Tofino [22]. These interconnects aim to emulate a traditional fabric of classic datacenter network designs, but the fact that network designers now have the ability to dynamically control the traffic pattern between network end points (e.g., between two Tofino switches within a datacenter). This flexibility is also enabling load balancers [12] to be deployed, which can be used to implement various traffic policies, thus making it possible to dynamically control the traffic pattern between network end points.

---

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

Chen Avin  
Benedikt Grunewald  
Stefan Schmid

This paper initiates the study of demand-aware network designs. In particular, we introduce the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.

**Design**

The design of demand-aware networks is based on the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.

**Optimality**

The optimality of demand-aware network designs is based on the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.

**Concurrent DANs**

**Static Optimality**

**ReNetS: Toward Static Optimally Self-Adjusting Networks**

Chen Avin  
Stefan Schmid

This paper initiates the study of demand-aware network designs. In particular, we introduce the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.

**Design**

The design of demand-aware networks is based on the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.

**Optimality**

The optimality of demand-aware network designs is based on the concept of a demand-aware network, which is a network that is designed to be able to dynamically control the traffic pattern between network end points. This is in contrast to traditional network designs, which are designed to be able to handle arbitrary traffic patterns.
Selected References

On the Complexity of Traffic Traces and Implications
Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid.
ACM SIGMETRICS, Boston, Massachusetts, USA, June 2020.

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.

Dynamically Optimal Self-Adjusting Single-Source Tree Networks
Chen Avin, Kaushik Mondal, and Stefan Schmid.
14th Latin American Theoretical Informatics Symposium (LATIN), University of Sao Paulo, Sao Paulo, Brazil, May 2020.

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 Anaïs Villedieu.
IFIP Networking, Warsaw, Poland, May 2019.

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

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.
Bonus Material

Hogwarts Stair
Industry Moving Forward!

Jupiter Evolving: Transforming Google’s Datacenter Network via Optical Circuit Switches and Software-Defined Networking

Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohi Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, Amin Vahdat
Google
sigcomm-jupiter-evolving@google.com

ABSTRACT
We present a decade of evolution and production experience with Jupiter datacenter network fabrics. In this period Jupiter has delivered 5x higher speed and capacity, 30% reduction in capex, 41% reduction in power, incremental deployment and technology refresh all while serving live production traffic. A key enabler for these improvements is evolving Jupiter from a Clos to a direct-connect topology among the machine aggregation blocks. Critical architectural changes for this include: A datacenter interconnection layer employing Micro-Electro-Mechanical Systems (MEMS) based Optical Circuit Switches (OCS) and a fully transparent Network Control Plane.

KEYWORDS
Datacenter network, Software-defined networking, Traffic engineering, Topology engineering, Optical circuit switches.

ACM Reference Format:
Reconfigurable Optical Networks Will Move Supercomputer Data 100X Faster

Newly designed HPC network cards and software that reshapes topologies on-the-fly will be key to success

By Michelle Hampson
Question:

How to Quantify such “Structure” in the Demand?
Intuition

Which demand has more structure?

→ Traffic matrices of two different distributed ML applications
  → GPU-to-GPU
Intuition

Which demand has more structure?

→ Traffic matrices of two different distributed ML applications
   → GPU-to-GPU

More uniform **VS** More structure
Intuition
Spatial vs temporal structure

→ Two different ways to generate same traffic matrix:
  → Same non-temporal structure

→ Which one has more structure?
Intuition
Spatial vs temporal structure

→ Two different ways to generate same traffic matrix:
  → Same non-temporal structure

→ Which one has more structure?

Systematically?
Trace Complexity

Information-Theoretic Approach

“Shuffle&Compress”
Trace Complexity
Information-Theoretic Approach
“Shuffle&Compress”

Increasing complexity (systematically randomized)

More structure (compresses better)
Trace Complexity

Information-Theoretic Approach

“Shuffle&Compress”
Trace Complexity

Information-Theoretic Approach
“Shuffle&Compress”

- Difference in size (entropy)?
- Difference in size (entropy)?
Trace Complexity

Information-Theoretic Approach
“Shuffle&Compress”

Can be used to define 2-dimensional complexity map!
Our Methodology

Complexity Map

Our approach: iterative randomization and compression of trace to identify dimensions of structure.
Our Methodology

Complexity Map

Our approach: iterative randomization and compression of trace to identify dimensions of structure.

Different structures!
Our Methodology

Complexity Map

Our approach: iterative randomization and compression of trace to identify dimensions of structure.

Different structures!
On the Complexity of Traffic Traces and Implications

CHEN AVIN, School of Electrical and Computer Engineering, Ben Gurion University of the Negev, Israel
MANYA GHOBADI, Computer Science and Artificial Intelligence Laboratory, MIT, USA
CHEN GRINER, School of Electrical and Computer Engineering, Ben Gurion University of the Negev, Israel
STEFAN SCHMID, Faculty of Computer Science, University of Vienna, Austria

This paper presents a systematic approach to identify and quantify the types of structures featured by packet traces in communication networks. Our approach leverages an information-theoretic methodology, based on iterative randomization and compression of the packet trace, which allows us to systematically remove and measure dimensions of structure in the trace. In particular, we introduce the notion of trace complexity which approximates the entropy rate of a packet trace. Considering several real-world traces, we show that trace complexity can provide unique insights into the characteristics of various applications. Based on our approach, we also propose a traffic generator model able to produce a synthetic trace that matches the complexity levels of its corresponding real-world trace. Using a case study in the context of datacenters, we show that insights into the structure of packet traces can lead to improved demand-aware network designs: datacenter topologies that are optimized for specific traffic patterns.

CCS Concepts: • Networks → Network performance evaluation; Network algorithms; Data center networks; • Mathematics of computing → Information theory;

Additional Key Words and Phrases: trace complexity, self-adjusting networks, entropy rate, compress, complexity map, data centers

ACM Reference Format:

1 INTRODUCTION
Packet traces collected from networking applications, such as datacenter traffic, have been shown to feature much structure: datacenter traffic matrices are sparse and skewed [16, 39], exhibit
Another Related Problem

Low Distortion Spanners

→ Classic problem: find *sparse, distance-preserving* (low-distortion) spanner of a graph

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

**Theorem:** If demand matrix is regular and uniform, and if we can find a constant distortion, linear sized (i.e., constant, sparse) spanner for this request graph: then we can design a constant degree DAN providing an optimal expected route length (i.e., \(O(H(X|Y)+H(Y|X))\).
Yet, can leverage the connection to spanners sometimes!

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

- \textbf{r-regular and uniform demand:}
- \textbf{Sparse, irregular (constant) spanner:}
- \textbf{Constant degree optimal DAN (ERL at most log r):}

Our degree reduction trick again!

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