Tutorial: Algorithmic Issues in Network Resource Reservation Problems

Stefan Schmid
Aalborg University, Denmark & TU Berlin, Germany

DIMACS Workshop on Algorithms for Data Center Networks
A rehash: It’s a great time to be a scientist!

How to exploit these flexibilites? How not to shoot in our feet? Can be challenging!

"We are at an interesting inflection point!"
Keynote by George Varghese at SIGCOMM 2014
New Flexibilities: It’s About Time!

- Datacenter networks, enterprise networks, Internet: a critical infrastructure of the information society
- We have seen a huge shift in scale and applications...
- ... but many Internet protocols hardly changed!

Applications: file transfer, email

Goal: connectivity between researchers

Applications: live streaming, IoT, etc.

Goal: quality-of-service, predictable performance, low latency, ...
Opportunity 1 of Network Virtualization: Overcoming Ossification

- Recent concern: Ossification in the network core
  - Are computer networks future-proof?
  - Meet the new requirements of new applications?

- Example Internet-of-Things:
  - IPv4: ~4.3 billion addresses, Gartner study: 20+ billion “smart things” by 2020
  - New security threats: recent DDoS attack based on IoT (almost 1TB/s, coming from webcams, babyphones, etc.)
Opportunity 1 of Network Virtualization: Overcoming Ossification

- Recent concern: Ossification in the network core
  - Are computer networks future-proof?
  - Meet the new requirements of new applications?

- Example Internet-of-Things:
  - IPv4: ~4.3 billion addresses, Gartner study: 20+ billion “smart things” by 2020
  - New security threats: recent DDoS attack based on IoT (almost 1TB/s, coming from webcams, babyphones, etc.)
Opportunity 2: Enable Resource Sharing for Improved Utilization

Virtual Application 1
- Quality-of-Service & Resource Requirements

Virtual Application 2
- Computational & Storage Requirements

Realization and Embedding

Virtualization and Isolation
Opportunity 2: Enable Resource Sharing for Improved Utilization

Virtual Application 1
Virtualization and Isolation
Quality-of-Service & Resource Requirements

Virtual Application 2
Computational & Storage Requirements

Opportunity: flexible, fast, and cheap deployment!

Challenge: How to provide performance isolation and predictable performance?
Opportunity 2: Enable Resource Sharing for Improved Utilization

Virtual Application 1
Quality-of-Service & Resource Requirements

Virtual Application 2
Computational & Storage Requirements

Realization and Embedding

Opportunity: flexible, fast, and cheap deployment!

Virtualization and Isolation

Challenge: How to provide performance isolation and predictable performance?

In general: For a predictable application performance, performance isolation needs to be provided along all involved resources.
Focus Today: *The Network*
The Network Matters

- Cloud-based applications generate significant network traffic
  - E.g., scale-out databases, streaming, batch processing applications

Example 1: Hadoop Terrasort job

Example 2: Aggregate Server Traffic in Google datacenter

Example 3: More memory-based systems (network becoming bottleneck again)

Jupiter rising @ SIGCOMM 2015
The Network Matters

- Cloud-based applications generate significant network traffic
  - E.g., scale-out databases, streaming, batch processing applications

Example 1: Hadoop Terrasort job

Example 2: Aggregate Server Traffic in Google datacenter

Example 3: More memory-based systems (network becoming bottleneck again)

As much time is spent on communication: For a predictable application performance, bandwidth resources need to be reserved explicitly.
The Network Matters

- Cloud-based applications generate significant network traffic
  - E.g., scale-out databases, streaming, batch processing applications

Example 1: Hadoop Terrasort job
As much time is spent on communication: For a predictable application performance, bandwidth resources need to be reserved explicitly.

Example 2: Aggregate Server Traffic
Ideally, communication should be local. And bandwidth reservations along short paths!
An algorithmic problem.

Example 3: More memory-based systems
(network becoming bottleneck again)

Jupiter rising @ SIGCOMM 2015
Structure in Traffic Matrix = Optimization Opportunities

- At the same time, traffic matrices are often far from random and uniform, but have a lot of structure and are sparse.

Example 1: Often little to no traffic between many racks.

Without taking this structure into account, some links may be overprovisioned and others underprovisioned.
Focus Today: *The Network*

We will be talking a lot about bandwidth reservations. *But:* Predictable network performance is about more, and interference can come in many flavors!
Consider: 2 **SDN-based virtual networks (vSDNs)** sharing physical resources!

Assume: perfect performance isolation on the network!

An Experiment: 2 vSDNs with bw guarantee!
To enable **multi-tenancy**, take **existing network hypervisor** (e.g. Flowvisor, OpenVirteX): provides network abstraction and control plane translation!

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

Intercepts control plane messages.

Translation could include, e.g., switch DPID, port numbers, ...

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

It turns out: the network hypervisor can be source of unpredictable performance!

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

Experiment: web latency depends on hypervisor CPU load!
The Many Faces of Performance Interference

Performance also depends on hypervisor type... (multithreaded or not, which version of Nagle’s algorithm, etc.)
The Many Faces of Performance Interference

Performance also depends on hypervisor type (multithreaded or not, which version of Nagle’s algorithm, etc.)

... number of tenants...

Conclusion: for predictable performance, need to account for all resources!

But let us now focus on the network itself.
</remark>
First Algorithmic Challenge: *Keep the traffic local!*

Consider a simple data center hosting two tenants: *green* and *blue*.

Bandwidth reservations for predictable performance!
First Algorithmic Challenge: *Keep the traffic local!*

Consider a simple data center hosting two tenants: *green* and *blue*.

Bad embedding, distributed across pods: reservations along long paths, costly shuffling!
First Algorithmic Challenge:  
*Keep the traffic local!*

Consider a simple data center hosting two tenants: green and blue.

Solutions?!

Bad embedding, distributed across pods: reservations along long paths, costly shuffling!
Consider a simple data center hosting two tenants: **green** and **blue**

**Solution 1: Adjust the Network**

Adjust the network!

Short communication paths!
Consider a simple data center hosting two tenants: green and blue.

Solution 2: Adjust Embedding

Much better embedding: Locally clustered within a rack or pod: efficient!
Consider a simple data center hosting two tenants: green and blue

Solution 2: Adjust Embedding

How to compute a minimal embedding? Known as the Virtual Network Embedding Problem.

mappers
tenant 1
reducers
tenant 1

mappers
tenant 2
reducers
tenant 2
Overview

PART I: Static Embeddings

PART II: Reconfiguring Embeddings

PART III: A request comes seldom alone!
PART I:
Static Embeddings
The Virtual Network Embedding Problem

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)
2 dimensions of flexibility:
- Mapping of virtual nodes (to physical nodes)
- Mapping of virtual links (to paths)

VNet aka “guest graph”

Substrate aka “host graph”

embedding?
The Virtual Network Embedding Problem

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

aka “guest graph”

aka “host graph”

VNet

Substrate

vm₁ → vm₃

vm₂ → vm

Assume unit demand and capacity!

embedding?
Let's start simple: assume node mappings are given.

2 dimensions of flexibility:
- Mapping of virtual nodes (to physical nodes)
- Mapping of virtual links (to paths)
Let’s start simple: assume node mappings are given.

The Virtual Network Embedding Problem

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
The Virtual Network Embedding Problem

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

- Let’s start simple: assume node mappings are given

VNet Substrate

How to compute 2 shortest paths under capacity constraints?

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
Let's start simple: assume node mappings are given.

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

Let's try greedy! First \(vm_1\)-\(vm_2\).

VNet Substrate

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
Let's start simple: assume node mappings are given.

2 dimensions of flexibility:
- Mapping of virtual nodes (to physical nodes)
- Mapping of virtual links (to paths)

Let's try greedy! First $vm_1$-$vm_2$.

Let’s start simple: assume node mappings are given.

2 dimensions of flexibility:
- Mapping of virtual nodes (to physical nodes)
- Mapping of virtual links (to paths)

A better solution: cost 5!

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
The Virtual Network Embedding Problem

- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

- Let’s start simple: assume node mappings are given

VNet                 Substrate

Joint optimization of 2 flows is already a challenging combinatorial problem! If demand=capacity=1: shortest 2-disjoint paths problem.

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
Let’s start simple: assume node mappings are given.

Joint optimization of 2 flows is already a challenging combinatorial problem! If demand=capacity=1: shortest 2-disjoint paths problem.

Embedding the 2 virtual links boils down to computation of 2 shortest paths!
Essentially a 2-disjoint shortest paths problem: a deep combinatorial problem

- NP-hard on directed graphs
- For undirected graphs:
  - Feasibility more or less understood: Robertson & Seymour
  - Shortest paths: recent breakthrough, first polytime randomized algorithm (still slow: a theoretical result)
  - We are still looking for polytime deterministic algorithms!

Mapping virtual links: Already hard!
Bad news: The Virtual Network Embedding Problem is hard even if endpoints are already mapped and given.

But maybe at least mapping nodes is simple?
Let’s start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line.
Let’s start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line.

Cost 5
Let’s start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line.

Mapping Virtual Nodes

Guest

Cost 2

Host

embedding?
Let’s start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line.

Minimizing the sum of virtual link lengths is a Minimum Linear Arrangement Problem (MinLA)!

NP-hard.
Therefore: VNEP is Hard “in Both Dimensions”!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

- Remark: the VNEP can also be seen as a generalization of the Subgraph Isomorphism Problem (SIP)

Known? Why is SIP NP-hard?
Therefore: VNEP is Hard “in Both Dimensions”!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

- Remark: the VNEP can also be seen as a generalization of the **Subgraph Isomorphism Problem (SIP)**
  - The SIP problem: Given two graphs G,H, determine whether \( G \) contains a subgraph that is isomorphic to \( H \)?
  - NP-hard: “does \( G \) contain an \( n \)-node cycle?” is a Hamilton cycle problem (each node visited exactly once), a solution to “does \( G \) contain a \( k \)-clique?” solves maximum clique problem, etc.

![Diagram showing network and subgraph isomorphism](image)
Therefore: VNEP is Hard “in Both Dimensions”!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

- Remark: the VNEP can also be seen as a generalization of the Subgraph Isomorphism Problem (SIP)
  - The SIP problem: Given two graphs G,H, determine whether G contains a subgraph that is isomorphic to H?
  - NP-hard: "does G contain an n-node cycle?" is a Hamilton cycle problem (each node visited exactly once), a solution to "does G contain a k-clique?" solves maximum clique problem, etc.

So if SIP is hard, why is VNEP hard?
NP-Hardness: From SIP to VNEP

- Observe: VNEP is a generalization of SIP

- For example:

Can VNet $G=(V,E)$ be embedded in $H$ at cost $|E|$?
(I.e., each virtual edge has length 1.)

$\iff$ Is $G$ a subgraph of $H$?
Remark: Graph Minors

Note: It is possible to embed a guest graph $G$ on a host graph $H$, even though $G$ is not a minor of $H$:

Assume planar host graph $H$: $K_5$ and $K_{3,3}$ minor-free...

... but it is possible to embed non-planar guest graph $G=K_5$!
Can we at least formulate a “fast” MIP?
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

One that provides good relaxations!
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:

Initially: no variables set

Subset of variables set

All variables set: infeasible, feasible, optimal?
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:

Decide: Is it worth exploring subtree?!
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual trick: Relax! Solve LP (fast!), and if relaxed solution (more general!) not better then best solution so far: skip it!
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

  Usual trick: Relax! Solve LP (fast!), and if relaxed solution (more general!) not better then best solution so far: skip it!

Bottomline: If MIP provides «good relaxations», large parts of the search space can be pruned.
Can we at least formulate a “fast” MIP?

A typical MIP formulation:

- Introduce binary variables $map(v,s)$ to map virtual nodes $v$ to substrate node $s$.
- Introduce flow variables for paths (say splittable for now).
- Ensure flow conservation: all flow entering a node must leave the node, unless it is the source or the destination.

$$
\sum_{u \rightarrow v} f_{uv} = \sum_{v \rightarrow w} f_{vw}
$$
Can we at least formulate a “fast” MIP?

We get constraints like:

\[ \forall v: \sum_u f_{uv} - f_{vu} \geq \text{map}(s,v) \times b - \text{map}(t,v) \times \infty \]

Assume bandwidth \( b \) requested from node \( s \) to node \( t \).

What does this formula do and why is it correct?
Can we at least formulate a “fast” MIP?

We get constraints like:

Assume bandwidth \( b \) requested from node \( s \) to node \( t \).

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall v: \sum_u f_{uv} - f_{vu} \geq \text{map}(s,v) \times b - \text{map}(t,v) \times \infty )</td>
<td></td>
</tr>
</tbody>
</table>

If \( \text{map}(s,v) = 1 \), i.e., \( s \) mapped to \( v \): so **flow starts at** \( v \), and hence outgoing flow must be larger than incoming flow (plus \( b \)).
Can we at least formulate a “fast” MIP?

We get constraints like:

\[ \forall v: \sum_u f_{uv} \geq \text{map}(s,v) \cdot b - \text{map}(t,v) \cdot \infty \]

Assume bandwidth \( b \) requested from node \( s \) to node \( t \).

If \( \text{map}(s,v)=0 \) and \( \text{map}(t,v)=0 \), i.e., \( v \) is along the path from \( s \) to \( t \): then we have flow conservation: outgoing flow must equal incoming flow (here \( \geq \), objective function will remove unnecessary flow).
Can we at least formulate a “fast” MIP?

We get constraints like:

\[ \forall v: \sum_u f_{uv} - f_{vu} \geq \text{map}(s,v) \ast b - \text{map}(t,v) \ast \infty \]

Assume bandwidth b requested from node s to node t.

If \( \text{map}(t,v)=1 \), i.e., t mapped to v: so flow terminates at node v: so no constraint: minus infinity (but objective function will remove unnecessary flow).
Can we at least formulate a “fast” MIP?

We get constraints like:

\[ \forall v: \sum_u f_{uv} - f_{vu} \geq \text{map}(s,v) \times b - \text{map}(t,v) \times \infty \]

Assume bandwidth b requested from node s to node t.

If map(t,v)=1, i.e., t mapped to v: so flow terminates at node v: so no constraint: minus infinity (but objective function will remove unnecessary flow).

Will such a MIP provide effective pruning?
What will happen in this example?

$V_1$ and $V_2$ are connected by an embedding process, which connects $S_1$ and $S_2$. The question is whether the embedding will affect the relationship between $V_1$ and $V_2$. 

$V_1$ and $V_2$ are represented by green nodes, while $S_1$ and $S_2$ are represented by gray nodes. The embedding process is indicated by a dashed line between the two sets of nodes.
What will happen in this example?

\[ \text{map}(v_1, s_1) = 0.5 \]
\[ \text{map}(v_2, s_1) = 0.5 \]
\[ \text{map}(v_1, s_2) = 0.5 \]
\[ \text{map}(v_2, s_2) = 0.5 \]
What will happen in this example?

Minimal flow = 0: fulfills flow conservation! Relaxation useless: does not provide any lower bound or indication of good mapping!
What about using randomized rounding?

Recall: classic approximation approach which: (i) computes a solution to the linear relaxation of the IP, (ii) decomposes this solution into convex combinations of elementary solutions, and (iii) probabilistically chooses any of the elementary solutions based on their weight.
What about using randomized rounding?

- Problem 1: relaxed solutions may not be very meaningful
  - see example for splittable flows before

- Problem 2: also for unsplittable flows, if using a standard Multi-Commodity Flow (MCF) formulation of VNEP, the integrality gap can be huge
  - Tree-like VNets are still ok
  - VNets with cycles: randomized rounding not applicable, since problem not decomposable

The linear solutions can be decomposed into convex combinations of valid mappings.
Non-Decomposability

- Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

- Example:

```
VNet

k   j
  / \  /
 /   \ /
 i  k

Host

u_1
  / \  /
 /   \ /
 u_6  u_2

u_5
  / \  /
 /   \ /
 u_4  u_3
```

VNet Host question mark
Non-Decomposability

Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

Example:

Valid LP solution: virtual node mappings sum to 1 and each virtual node connects to its neighboring node with half a unit of flow...
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to $u_1$ and the only neighboring node that hosts j is $u_2$, so i must be fully mapped on $u_1$ and j on $u_2$. Similarly, k must be mapped on $u_3$. But flow of virtual edge (k,i) leaving $u_3$ only leads to $u_4$, so i must be mapped on both $u_1$ and $u_4$. This is **impossible**, even if capacities are infinite.

Example:
Impossible to decompose and extract any single valid mapping. Intuition: Node i is mapped to $u_1$ and the only neighboring node that hosts j is $u_2$, so i must be fully mapped on $u_1$ and j on $u_2$. Similarly, k must be mapped on $u_3$. But flow of virtual edge (k,i) leaving $u_3$ only leads to $u_4$, so i must be mapped on both $u_1$ and $u_4$. This is impossible, even if capacities are infinite.

Example:

How to devise a Linear Programming formulations, such that convex combinations of valid mappings can be recovered?!
That's all Folks!
Thank you for your attention!

Wait a minute!
These problems need to be solved!
And they often can, even with guarantees.

That's all Folks!
Guest graphs may not be general graphs, but e.g., virtual clusters: very simple and symmetric, used in context of batch processing

- $k$ VMs/compute-units/tasks/...
- Connected to virtual switch at bandwidth $b$

Theory vs Practice: In Practice There is Hope!
How to Embed a Virtual Cluster?

Consider host graph:

Physical switch

Server with 4 VM slots (3 occupied, 1 free)

occupied

free
How to Embed a Virtual Cluster?

Consider guest graph:

$b=1$

$n=5$

require 2 cores each
How to Embed a Virtual Cluster?
1. Place logical switch (try all options)
1. Place logical switch (try all options)
2. **Extend network** with artificial source \( s \) and sink \( t \)
1. Place logical switch (try all options)
2. Extend network with artificial source $s$ and sink $t$
3. Add capacities (recall that $b=1$, so each virtual node needs one unit of capacity)

```
Capacity = # virtual nodes
```

```
capacity = how many virtual nodes (requiring 2 cores) can be placed?
```
Then: Compute min-cost max flow of size $n$ from $s$ to $t$ (e.g., successive shortest paths): due to capacity constraints at most size $n$. 

$n=5$
Then: Compute min-cost max flow of size $n$ from $s$ to $t$
(e.g., successive shortest paths): due to capacity constraints at most size $n$.
... and assign virtual nodes (and edges) accordingly!
How to Embed a Virtual Cluster?

In fact: this physical network is even a tree!
For trees with servers at leaves, even simpler algorithms are possible. Ideas?
Dynamic Programming
Dynamic Programming

Bottom-up programming: given optimal solution for subtrees, can quickly compute optimal solution for entire tree!
Dynamic Programming

Given optimal embedding for $x \in \{0, \ldots, n\}$ virtual nodes in left subtree...

Given optimal embedding for $x \in \{0, \ldots, n\}$ virtual nodes in right subtree...
Dynamic Programming

... can compute optimal embedding of $x \in \{0, ..., n\}$ virtual nodes in entire subtree!

Given **optimal embedding** for $x \in \{0, ..., n\}$ virtual nodes in left subtree...

Given **optimal embedding** for $x \in \{0, ..., n\}$ virtual nodes in right subtree...
Dynamic Programming

Bottom-up «induction». Leaves easy: either x nodes fit server (cost 0) or not (cost ∞): $\text{opt}[\leq 4] = 0$, $\text{opt}[>4] = \infty$
Dynamic Programming

\[ \text{opt}(T, x) = \min_{0 \leq y \leq x} \{ \text{opt}(\text{left}, y) + \text{opt}(\text{right}, x-y) + \text{bw}(T, x) \} \]

To compute cost of embedding \( x \) nodes in \( T \), place \( y \) nodes on the left, \( x-y \) on the right subtree, and compute cost due to links across root.

account for bw to n-x remaining nodes
Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)

Logical Switch Variant vs Hose Variant

Aggregated bw in/out node at most b.
Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)

Can serve the same communication patterns! (A polytope of possible traffic matrices.)

Example:
- (1,2): b/2
- (1,3): b/2
- (2,3): b/2
Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)

But **embedding costs** can be different if we do not insist on placing the logical switch explicitly! **Not on trees** though, and **not in uncapacitated networks**: without routing restrictions, optimal routing paths form a tree (**SymG=SymT** a.k.a. **VPN Conjecture**).
The Benefit of Hose Interpretation
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

How to embed as a star?
The Benefit of Hose Interpretation

VNet: VC(n=6, b=1)

Host Graph: A Ring

How to embed as a star?

Impossible: need at least 5 units of flow from/to node where star center is mapped. However, capacity of incident links is only 4.
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

How about hose?
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Node mapping! Now: How to embed these virtual links?
VNet: VC(n=6, b=1)

Host Graph: A Ring

Recall: hose has total demand at most 1.
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Recall: hose has total demand at most 1.

Virtual links from node 1 to \{2,3,4,5,6\} can be implemented along this route: fulfills capacity constraints under any traffic matrix fulfilling hose specification!
Remaining virtual links to embed for virtual node 2.
The Benefit of Hose Interpretation

Remaining virtual links to embed for virtual node 2.

Host Graph: A Ring

Can be implemented along this route: from node 2, reach nodes \{3,4,5,6\}.
The Benefit of Hose Interpretation

VNet: VC(n=6, b=1)

Host Graph: A Ring

Remaining virtual links to embed for virtual node 3.
The Benefit of Hose Interpretation

VNet: VC(n=6, b=1)

Host Graph: A Ring

Remaining virtual links to embed for virtual node 3.

From 3, path reaches {4, 5, 6}.
The Benefit of Hose Interpretation

VNet: $VC(n=6,b=1)$

Host Graph: A Ring

Remaining virtual links to embed for virtual node 4.
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Remaining virtual links to embed for virtual node 4.

Route from 4 to {5,6}. 
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Remaining virtual link.
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

All virtual links mapped to routes!
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)  

Host Graph: A Ring

But wait: 5 paths on link {5,6}! Can demand really be satisfied given link capacity of 2?!
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Link \{5,6\} is used by routes: (1,5),(2,5),(3,6),(4,6),(5,6).
But by definition of the hose model, any traffic matrix \( M \) will respect:
\[
M_{1,5} + M_{2,5} \leq 1 \text{ and } M_{3,6} + M_{4,6} + M_{5,6} \leq 1.
\]
Hence \( \Sigma M_{i,j} \leq 2 \) holds!
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)

Host Graph: A Ring

Legal embedding! Recall: this was impossible with virtual switch abstraction.

Link \{5,6\} is used by routes: \((1,5),(2,5),(3,6),(4,6),(5,6)\).

But by definition of the hose model, any traffic matrix \(M\) will respect:

\[ M_{1,5} + M_{2,5} \leq 1 \text{ and } M_{3,6} + M_{4,6} + M_{5,6} \leq 1. \]

Hence \(\sum M_{i,j} \leq 2\) holds!
Similar problems arise in many contexts

For example, service chain embeddings: in a service chain, traffic is steered (e.g., using SDN) through a sequence of (virtualized) middleboxes to compose a more complex network service
Similar problems arise in many contexts

For example, service chain embeddings: in a service chain, traffic is steered (e.g., using SDN) through a sequence of (virtualized) middleboxes to compose a more complex network service.
The Many Faces of the VNEP: E.g., Service Chain Embedding

- Similar problems arise in many contexts.

  Interesting implication: routes from s to t become *walks* (rather than *simple paths*)! How to find shortest walks?

- To compose a more complex service chain, traffic is steered through a service chain (e.g., using SDN) through a sequence of (virtualized) middleboxes to compose a more complex network service.
Routing Through Waypoints

- Traditionally: routes form **simple paths** (e.g., shortest paths)
- Novel aspect: routing through **middleboxes** may require more general paths, with loops: a walk

How to compute a shortest route through a waypoint?
Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!
Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

Greedy fails: choose shortest path from s to w...
Walking Through Waypoints

- Computing shortest routes through waypoints is **non-trivial**!
  
  **Greedy fails**: ... now need long path from \( w \) to \( t \)
Computing shortest routes through waypoints is non-trivial!

Greedy fails: ... now need long path from w to t

Total length: 2+6=8

Assume unit capacity and demand for simplicity!
Walking Through Waypoints

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

Total length: $4 + 2 = 6$

A better solution: jointly optimize the two segments!
Walking Through Waypoints

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

Total length: $4 + 2 = 6$

How hard can it be?

A better solution: jointly optimize the two segments!
NP-hard on Directed Networks: Reduction from Disjoint Paths Problem

Reduction: From joint shortest paths \((s_1,t_1),(s_2,t_2)\) to shortest walk \((s,w,t)\) problem

Recall: computing 2-disjoint paths NP-hard on directed graphs.

We show: If waypoint routing was be in \(P\), we could solve it fast. Contradiction!
NP-hard on Directed Networks: Reduction from Disjoint Paths Problem

**Reduction:** *From* joint shortest paths \((s_1,t_1),(s_2,t_2)\) to shortest walk \((s,w,t)\) problem

*Reduction:* To find shortest paths \((s_1,t_1)\), \((s_2,t_2)\), introduce **waypoint** \(w\) and connect \(t_1\) to \(s_2\) via \(w\)....
NP-hard on Directed Networks: Reduction from Disjoint Paths Problem

Reduction: From joint shortest paths \((s_1,t_1),(s_2,t_2)\) to shortest walk \((s,w,t)\) problem

... and ask for shortest waypoint route \((s_1,w,t_2)\)
Reduction from Disjoint Paths Problem

The walk \((s_1, w, t_2)\) walk defines a \((s_1, t_1)\) and a \((s_2, t_2)\) path pair before/after the waypoint! Solves original problem: Contradiction!

Reduction: To find shortest paths \((s_1, t_1)\), \((s_2, t_2)\), introduce **waypoint** \(w\) and connect \(t_1\) to \(s_2\) via \(w\)....

... and ask for **shortest waypoint route** \((s_1, w, t_2)\)
What about waypoint routes on *undirected* networks?
What about waypoint routes on undirected networks?

- Reduction from disjoint paths no longer works: disjoint paths problem not NP-hard on undirected networks
What about waypoint routes on *undirected* networks?

- **Reduction from disjoint paths no longer works**: disjoint paths problem is not NP-hard on undirected networks.

- **Indeed, algorithm exists**: We can reduce to edge-disjoint paths to compute a waypoint route!

---

**Walks**

**Edge-Disjoint Paths**
Indeed, algorithm exists: We can reduce to edge-disjoint paths to compute a waypoint route!

Reduction from disjoint paths no longer works: disjoint paths problem not NP-hard on undirected networks.

Replace capacitated links with parallel links!
What about waypoint routes on \textit{undirected} networks?

- Reduction from disjoint paths no longer works: disjoint paths problem not NP-hard on undirected networks.

- Indeed, \textbf{algorithm exists}: We can reduce to edge-disjoint paths to compute a waypoint route.

- Replace capacitated links with parallel links!
- Shortest paths \((s,w),(w,t)\) will give shortest \((s,w,t)\) path!

Walks \hspace{2cm} \text{Edge-Disjoint Paths}
Fast and Shortest Waypoint Routing on Undirected Networks: Suurballe’s Algorithm
Suurballe’s algorithm: finds two (edge-)disjoint shortest paths between same endpoints:
Suurballe’s algorithm: finds two (edge-)disjoint shortest paths between same endpoints:

Fast and Shortest Waypoint Routing on Undirected Networks: Suurballe’s Algorithm

How to compute a shortest \((s,w,t)\) route with this algorithm??
Waypoint Routing on Steroids

- Reduction to Suurballe’s algorithm:

- To find shortest \((s,w,t)\) route...
Reduction to Suurballe’s algorithm:

- Connect $S^+$ to $s$ and $t$.
- Connect $w$ to $T^+$. 

Waypoint Routing on Steroids
Waypoint Routing on Steroids

- Reduction to Suurballe’s algorithm:

  - Ask Suurballe for 2 disjoint paths from $S^+$ to $T^+$...
Waypoint Routing on Steroids

- Reduction to Suurballe’s algorithm:

... and hence also $(s,w,t)$. 

Diagram:

- Nodes: $s$, $w$, $t$, $G$
- Edges: $s 	o G 	o w 	o t$
Waypoint Routing on Steroids

- Reduction to Suurballe's algorithm:

  \[ S^+ + T^+ \]

  Can’t I use Suurballe to efficiently compute disjoint paths as well?!
Waypoint Routing on Steroids

- Reduction to Suurb's algorithm

No! Solves a much easier problem: 2 routes from \( \{s_1, s_2\} \) to \( \{t_1, t_2\} \).
Remarks: Under the rug...

- **Remark 1:** Suurballe is actually for **directed substrate graphs**, so need gadget to transform problem in right form:

- **Remark 2:** Suurballe: actually vertex disjoint
  - Suurballe & Tarjan: edge disjoint
Remarks: Under the rug...

- **Remark 1:** Suurballe is actually for directed substrate graphs, so need gadget to transform problem in right form:

- **Remark 2:** Suurballe: actually vertex disjoint
  - Suurballe & Tarjan: edge disjoint

Conclusion: exploiting traffic engineering flexibilities is non-trivial!
PART II: Dynamic Embeddings
Real Communication Patterns Change over Time

- E.g., changing bandwidth demand: map reduce application cycles through phases of high and low bandwidth requirements

- E.g., long-running applications (e.g., streaming) change in popularity

- E.g., job churn: jobs terminate, new ones arrive

- E.g., large elephant flow degree, changing over time (cf ProjecToR at SIGCOMM 2016)

Bandwidth utilization of 3 different runs of the same TeraSort workload (without interference)
Real Communication Patterns Change over Time

- E.g., changing bandwidth demand: map reduce application cycles through phases of high and low bandwidth requirements

- E.g., long-running applications (e.g., streaming) change in popularity

- E.g., job churn: jobs terminate, new ones arrive

- E.g., large elephant flow degree, changing over time (cf ProjecToR at SIGCOMM 2016)

Bandwidth utilization of 3 different runs of the same TeraSort workload (without interference)
Real Communication Patterns Change over Time

- E.g., changing bandwidth demand: map reduce application cycles through phases of high and low bandwidth requirements.

- E.g., long-running applications (e.g., streaming) change in popularity.

- E.g., job churn: jobs terminate, new ones arrive.

- E.g., large elephant flow degree, changing over time (cf ProjecToR at SIGCOMM 2016).

Ideally, we want to place frequently communicating nodes close: may require adaptations over time!

Adjust the network! Migrate and collocate!

Data utilization of 3 different runs of the same TeraSort workload (without interference).
Essentially An Online RePartitioning Problem

Communication @ time t:

How to embed pattern across $\ell=4$ servers (or racks, pods, etc.) of size $k=4$?
Essentially An Online RePartitioning Problem

Communication @ time t:
Essentially An Online RePartitioning Problem

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

... little inter-cluster communication.

A classic (hard) combinatorial problem!
Now assume: changes in communication pattern!

- E.g., more communication (1,3), (3,4), (2,5) but less (5,6)
Now assume: changes in communication pattern!

- E.g., more communication \((1,3),(3,4),(2,5)\) but less \((5,6)\)

Makes sense that nodes 1 and 5 change clusters!
Consider a simple network, e.g., a single switch (e.g., a rack):
Consider a simple network, e.g., a single switch (e.g., a rack):

Costs: 0, 1, α
A Simple Model for the Tutorial

Consider a simple network, e.g., a single switch (e.g., a rack):

Objective: minimize total communication and migration cost!
More precisely: competitive ratio $\rho = \text{cost(ON)}/\text{cost(OPT)}$
Consider a simple network, e.g., a single switch (e.g., a rack):

Objective:

minimize total communication and migration cost!

More precisely: \( \text{competitive ratio} \ \rho = \frac{\text{cost(ON)}}{\text{cost(OPT)}} \)

Nice: If competitive ratio is low, there is no need to develop any sophisticated prediction models (which may be wrong anyway)! The guarantee holds in the worst-case.
Consider a simple network, e.g., a single switch (e.g., a rack):

Objective: minimize total communication and migration cost!
More precisely: competitive ratio $\rho = \text{cost(ON)}/\text{cost(OPT)}$

"Prediction is difficult, especially about the future."

Nils Bohr

Nice: If competitive ratio is low, there is no need to develop any sophisticated prediction models (which may be wrong anyway)! The guarantee holds in the worst-case.
Adversarial Models

Weak adversary

- Chooses \textit{request distribution} $D$
- Requests \textit{sampled i.i.d.} from $D$
- Cannot react to online algo

Strong adversary

- Can generate \textit{arbitrary request sequence} $\sigma$
- Knows and can react to online algo
Adversarial 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: Algorithmic Challenges
- Do not know $D$ resp. $\sigma$ ahead of time
- Upon each communication request $(u,v)$:
  - Migrate $u$ and $v$ together? «Rent-or-buy»: migration cost should be amortized
  - 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 !

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

OFF:
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:**

```
  [ ]  [ ]
```

**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)!

E.g., ON can even collocate all except for one!

OFF:  

ON:
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)!

E.g., ON can even collocate all except for one!

What is the achievable competitive ratio?
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:  

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

Ouch! Cost 1 for each request.
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:  

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 !

- 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:  

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

What is the cost of OFF?

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 !

- 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:**

Ouch! Cost 1 for each request.
It is hard to compete under !

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

  \[ \text{OFF:} \]

  \[ \text{ON:} \]

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

  Ouch! Cost 1 for each request.

Move to configuration $i \in \{1, \ldots, k\}$ which is asked the least.

Averaging argument: At least $k$ times less communication cost!

Lower bound of $\Omega(k)$ for competitive ratio, despite big augmentation!
A Simple $O(n^2)$ Upper Bound

At least it does not depend on time! 😊
A Simple $O(n^2)$ Upper Bound

- Algorithm DET:
  - 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 DET:
  - 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
Algorithm DET:
- 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
A Simple $O(n^2)$ Upper Bound

- **Algorithm DET:**
  - 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 $O(n^2)$ Upper Bound

- **Algorithm DET:**
  - 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!
    - If this is not possible: phase ends

Components cannot be partitioned perfectly (first component alone too large)!
A Simple $O(n^2)$ Upper Bound

- **Algorithm DET:**
  - 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*!
    - If this is not possible: phase ends

**Competitive ratio?**
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)
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

Impossible: $u$ and $v$ already collocated by definition!
A Simple O(n^2) Upper Bound

- Analysis (costs per phase):
  - Observe: edge weights always ≤ α: once reach α, their endpoints will always be collocated (by algorithm definition)
  - α-edges form a forest (so at most n many!): once two nodes (u,v) are connected by a path of α-edges, they are in a single cluster and will no longer communicate across clusters
  - Thus: ON cost per phase:
    - At most 1 reorganization per α-edge (at most n α-edges), so n times reconfig cost n·α, so n^2α
    - Communication cost: at most α per edge (at most n^2 many), so also at most n^2α
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 $n \cdot \alpha$-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 rematch“): 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
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 \textit{whp}
- Then move to the best location \textit{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 with high probability (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

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

- Define **conditions** for configurations: if met, **never go back** to it (we can afford it w.h.p.: seen enough samples)
- Mantra of our algorithm: Rotate!
  - Rotate early, but not too early!
  - And: rotate **locally**
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)!
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)!

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.
PART III: Embeddings over Time
A VNet seldom comes alone!

Which ones to accept and embed? Admission control!
A VNet seldom comes alone!

Different flavors of VNets:
- Node mappings given or subject to optimization
- Different routing and traffic models
- Different prices and durations
- ...

Which ones to accept and embed? Admission control!
A VNet seldom comes alone!

- Different flavors of VNets:
  - Node mappings given or subject to optimization
  - Different routing and traffic models
  - Different prices and durations
  - ...

Can be solved by online approximation framework by Buchbinder and Naor.
Different VNet Flavors

- Traffic models
  - Customer Pipe
    - Traffic matrix: Bandwidth per VM pair \((u,v)\)
  - Hose Model
    - Per VM bandwidth: polytope of traffic matrices.
  - Aggregate Ingress
    - Only ingress specified: e.g., support multicast etc.

- Routing models
  - Tree
    - Steiner tree embedding
  - Single Path
    - Unsplittable paths
  - Multi-Path
    - Splittable paths (more capacity)

Relay costs: e.g., depending on packet rate
**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row $e : A_{e,(j,\ell)} \neq 0$ do
      $$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/ce} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/ce} - 1).$$
   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0.$
Primal and Dual

Applying Buchbinder & Naor

Formulate the packing (dual) LP: Maximize profit (Note: dynamic LP!)

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. \( f_{j,\ell} \leftarrow \arg\min\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
       \( x_e \leftarrow x_e + \frac{2^{A_{e,(j,\ell)}}}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}} - 1) \).
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
Primal and Dual

Applying Buchbinder & Naor

\[
\begin{align*}
\text{(I)} & \quad \begin{array}{ll}
\min & Z_j^T \cdot 1 + X^T \cdot C \\
\text{s.t.} & Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& X, Z_j \geq 0 
\end{array} \\
\text{(II)} & \quad \begin{array}{ll}
\max & B_j^T \cdot Y_j \\
\text{s.t.} & A_j \cdot Y_j \leq C \\
& D_j \cdot Y_j \leq 1 \\
& Y_j \geq 0
\end{array}
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \(j\)th round:

1. \(f_{j, \ell} \leftarrow \arg\min \{ \gamma(j, \ell) : f_{j, \ell} \in \Delta_j \} \) (oracle procedure)
2. If \(\gamma(j, \ell) < b_j\) then, (accept)
   (a) \(y_{j, \ell} \leftarrow 1\).
   (b) For each row \(e\) : If \(A_{e,(j,\ell)} \neq 0\) do
      \[
x_e \leftarrow x_e \cdot \frac{2^{A_{e,(j,\ell)} / c_e}}{w(j, \ell)} + \frac{1}{w(j, \ell)} \cdot \left(2^{A_{e,(j,\ell)} / c_e} - 1\right).
\]
   (c) \(z_j \leftarrow b_j - \gamma(j, \ell)\).
3. Else, (reject)
   (a) \(z_j \leftarrow 0\).
Primal and Dual

Applying Buchbinder & Naor

\[
\begin{align*}
\text{(I)} & & \min & Z_j^T \cdot 1 + X^T \cdot C & \text{ s.t. } & Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& & & X, Z_j \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{(II)} & & \max & B_j^T \cdot Y_j & \text{ s.t. } & A_j \cdot Y_j \leq C \\
& & & D_j \cdot Y_j \leq 1 \\
& & & Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-orNothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   \begin{enumerate}
   \item \( y_{j,\ell} \leftarrow 1. \)
   \item For each row \( e : A_{e,(j,\ell)} \neq 0 \) do
       \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1). \]
   \end{enumerate}
3. Else, (reject)
   \begin{enumerate}
   \item \( z_j \leftarrow b_j - \gamma(j, \ell). \)
   \end{enumerate}
**Primal and Dual**

**Applying Buchbinder & Naor**

<table>
<thead>
<tr>
<th>(I)</th>
<th>(II)</th>
</tr>
</thead>
</table>
| \[
\begin{align*}
\min Z_j^T \cdot 1 + X^T \cdot C & \quad \text{s.t.} \\
Z_j^T \cdot D_j + X^T \cdot A_j & \geq B_j^T \\
X, Z_j & \geq 0
\end{align*}
| \[
\begin{align*}
\max B_j^T \cdot Y_j & \quad \text{s.t.} \\
A_j \cdot Y_j & \leq C \\
D_j \cdot Y_j & \leq 1 \\
Y_j & \geq 0
\end{align*}
|

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

**Algorithm**

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \text{argmin} \{ \gamma(j, \ell) : f_{j,\ell} \in \Delta_j \} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e : A_{e,(j,\ell)} \neq 0 \) do
      \[
      x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)} / c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)} / c_e} - 1).
      \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
Primal and Dual

Applying Buchbinder&Naor

\[
\begin{align*}
\text{min } & Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& X, Z_j \geq 0
\end{align*}
\]

(I)

\[
\begin{align*}
\text{max } & B_j^T \cdot Y_j \quad \text{s.t.} \\
& A_j \cdot Y_j \leq C \\
& D_j \cdot Y_j \leq 1 \\
& Y_j \geq 0
\end{align*}
\]

(II)

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

**Algorithm**

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \(j\)th round:

1. \(f_{j,\ell} = \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in A_j\}\) (oracle procedure)
2. If \(\gamma(j, \ell) < b_j\) then, (accept)
   - \((a)\) \(y_{j,\ell} = 1\).
   - \((b)\) For each row \(e\) : If \(A_{e,(j,\ell)} \neq 0\) do
     \[
     x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).
     \]
   - \((c)\) \(z_j = b_j - \gamma(j, \ell)\).
3. Else, (reject)
   - \((a)\) \(z_j \leftarrow 0\).

Embedding cost vs profit?
Primal and Dual

Applying Buchbinder & Naor

\[
\begin{align*}
\text{(I)} & \quad \min Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& \quad Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& \quad X, Z_j \geq 0
\\
\text{(II)} & \quad \max B_j^T \cdot Y_j \quad \text{s.t.} \\
& \quad A_j \cdot Y_j \leq C \\
& \quad D_j \cdot Y_j \leq 1 \\
& \quad Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
       \[
       x_e \leftarrow x_e \cdot \frac{2^{A_{e,(j,\ell)}/c_e}}{w(j,\ell)} \cdot \left( \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1) \right).
       \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).

If cheap: accept and update primal variables (always feasible solution)
Primal and Dual

Applying Buchbinder & Naor

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j, \ell} \leftarrow \arg \min \{ \gamma(j, \ell) : f_{j, \ell} \in A_j \}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j, \ell} \leftarrow 1$.
   (b) For each row $e$ : If $A_{e,(j,\ell)} \neq 0$ do

   $$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0.$

Else reject
Primal and Dual

Applying Buchbinder & Naor

\[
\begin{align*}
\text{(I)} & \quad \min & \quad Z_j^T \cdot 1 + X^T \cdot C & \text{s.t.} \\
& & \quad Z_j^T \cdot D_j + X^T \cdot A_j & \geq B_j^T \\
& & \quad X, Z_j \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{(II)} & \quad \max & \quad B_j^T \cdot Y_j & \text{s.t.} \\
& & \quad A_j \cdot Y_j & \leq C \\
& & \quad D_j \cdot Y_j & \leq 1 \\
& & \quad Y_j & \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \(j\)th round:

1. \(f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}\) (oracle procedure)
2. If \(\gamma(j, \ell) < b_j\) then, (accept)
   (a) \(y_{j,\ell} \leftarrow 1\).
   (b) For each row \(e\) : If \(A_{e,(j,\ell)} \neq 0\) do
   \[
   x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).
   \]
   (c) \(z_j \leftarrow b_j - \gamma(j, \ell)\).
3. Else, (reject)
   (a) \(z_j \leftarrow 0\).

Computationally hard!
Primal and Dual

Applying Buchbinder&Naor

\[
\begin{align*}
\text{(I)} & \quad \min Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& \quad Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& \quad X, Z_j \geq 0
\\
\text{(II)} & \quad \max B_j^T \cdot Y_j \quad \text{s.t.} \\
& \quad A_j \cdot Y_j \leq C \\
& \quad D_j \cdot Y_j \leq 1 \\
& \quad Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j, \ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j, \ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j, \ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_e,(j,\ell) \neq 0 \) do
       \[
       x_e \leftarrow x_e \cdot 2^{A_e,(j,\ell)/c_e} + \frac{1}{w(j, \ell)} 
       \cdot (2^{A_e,(j,\ell)/c_e} - 1).
       \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).

Computationally hard! Use your favorite approximation algorithm! If competitive ratio \( \rho \) and approximation \( r \), overall competitive ratio \( \rho \ast r \).
Online VNet Admission Control

Algorithm comes in 2 flavors:

- **Bicriteria guarantees:** Obtain constant fraction of the optimal benefit while augmenting resources by a logarithmic factor.

- **Fractional guarantees resp. limited resource consumption:** The online algorithm achieves a logarithmic competitive ratio without resource augmentation, but either:
  - may serve a fraction of a request (associated benefit is assumed to be the same fraction of the benefit of the request)
  - the allowed traffic patterns of a request consumes at most a logarithmic fraction of every resource (in which case the algorithm rejects the request or fully embeds it)
Summary

- Predictable performance requires isolation of all resources
- Static embeddings
- Reconfiguring embeddings
- Embeddings over time
Further Reading

Network Hypervisor Performance:
- Logically Isolated, Actually Unpredictable? Measuring Hypervisor Performance in Multi-Tenant SDNs

Virtual Network Embedding:
- Beyond the Stars: Revisiting Virtual Cluster Embeddings
- Service Chain and Virtual Network Embeddings: Approximations using Randomized Rounding
- Charting the Complexity Landscape of Waypoint Routing
  Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid.
  ArXiv Technical Report, May 2017
- Competitive and Deterministic Embeddings of Virtual Networks

Online Collocation:
- Online Balanced Repartitioning
  Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid.
- Competitive Clustering of Stochastic Communication Patterns on the Ring
  5th International Conference on Networked Systems (NETYS), Marrakech, Morocco, May 2017

Network Design:
- Demand-Aware Network Designs of Bounded Degree
  Chen Avin, Kaushik Mondal, and Stefan Schmid.
Further Reading

Network Hypervisor Performance:

Virtual Network Embedding:

Online Collocation:

Network Design:

Hypervisor performance and interference analysis
- Including decomposability (study of integrality gap)
- Star vs Hose embedding
- Relationship to disjoint paths
- Online primal-dual
- Adaptive VM migration to minimize communication costs
- A model and analysis motivated by ProjectToR