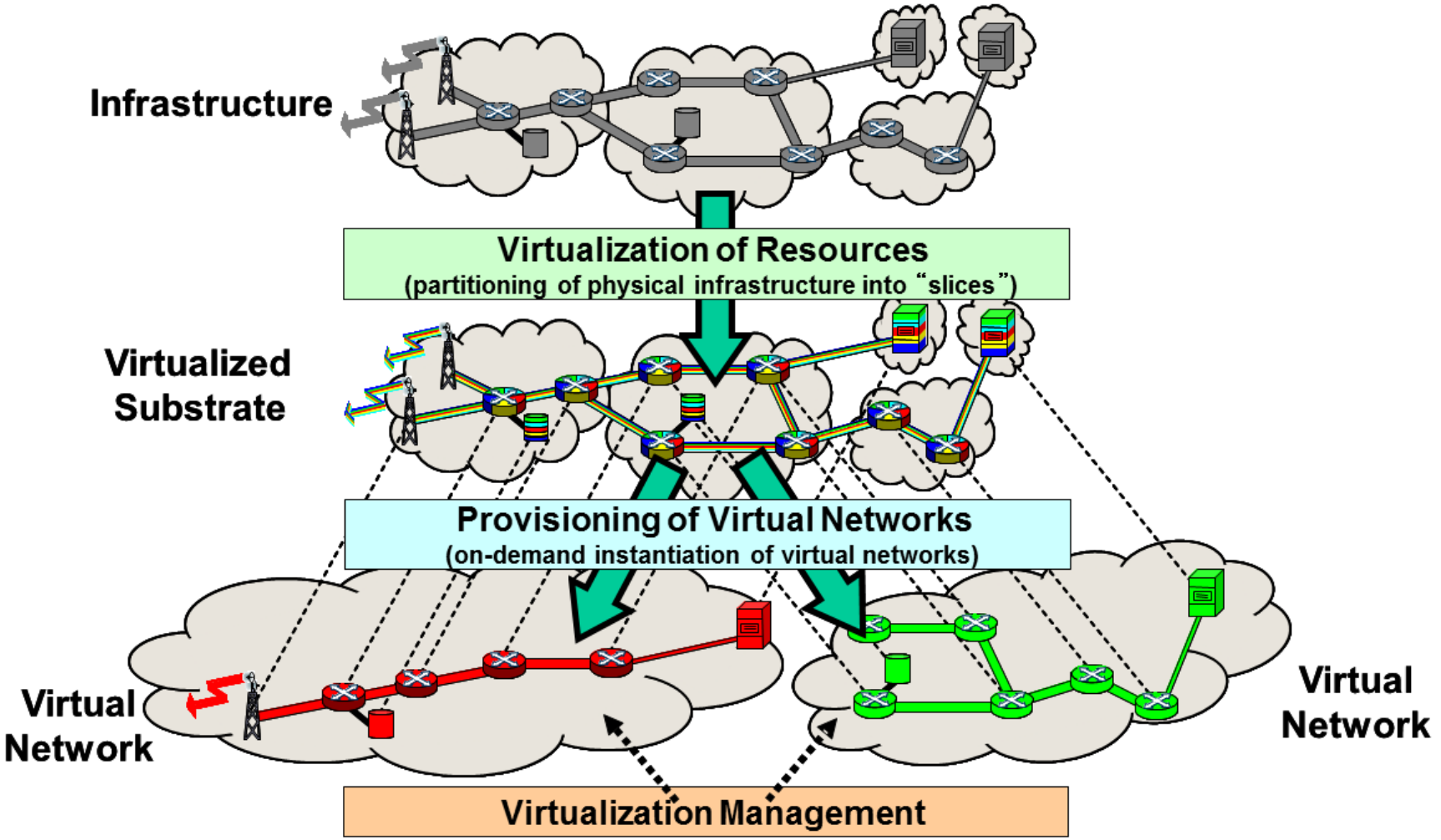
# Virtual Network Embeddings with Good and Bad Intentions

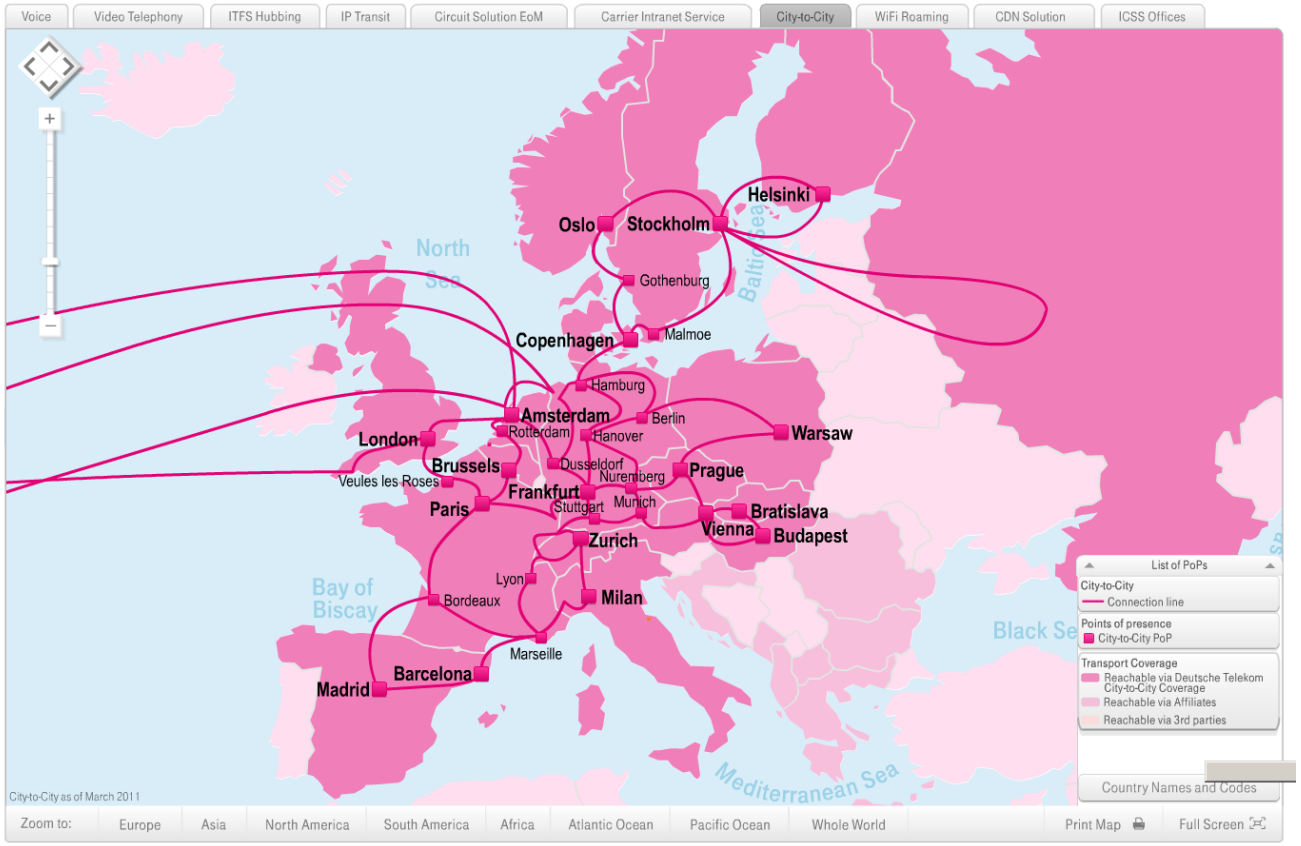

Stefan Schmid

February, 2014

# VNets: Virtual Networking Cloud Resources.

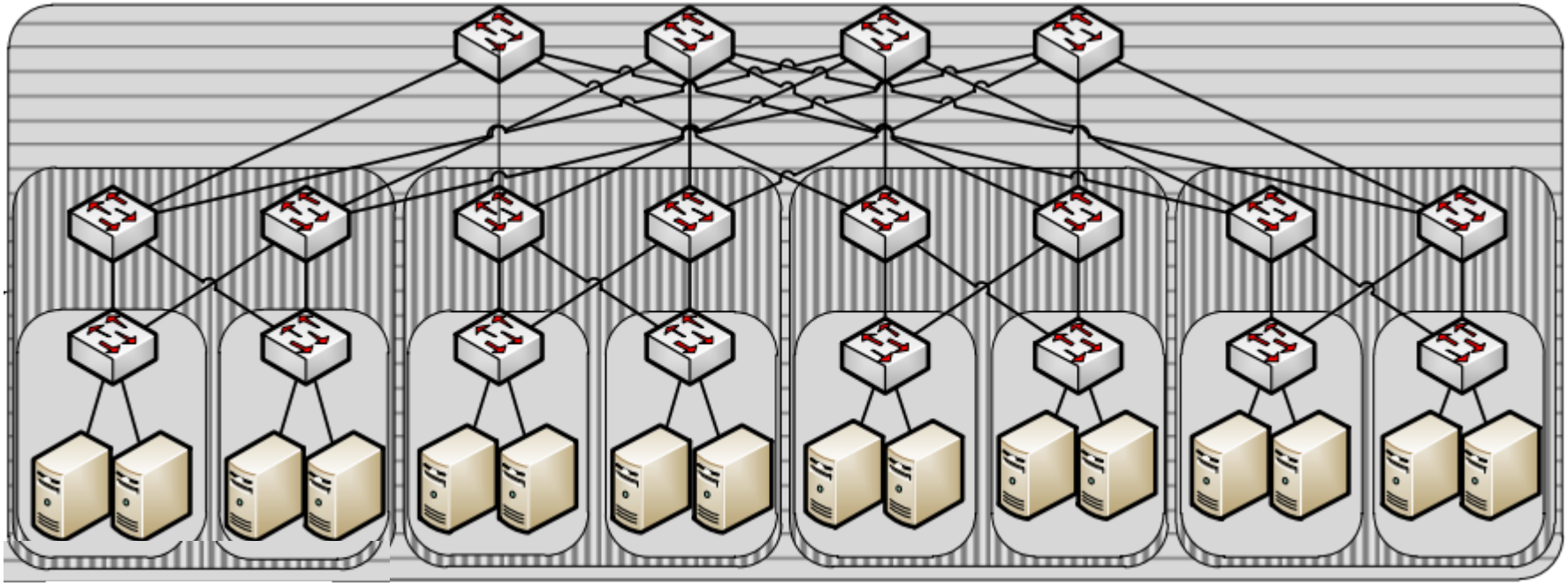


# Wide-Area VNet: Distributed Cloud.



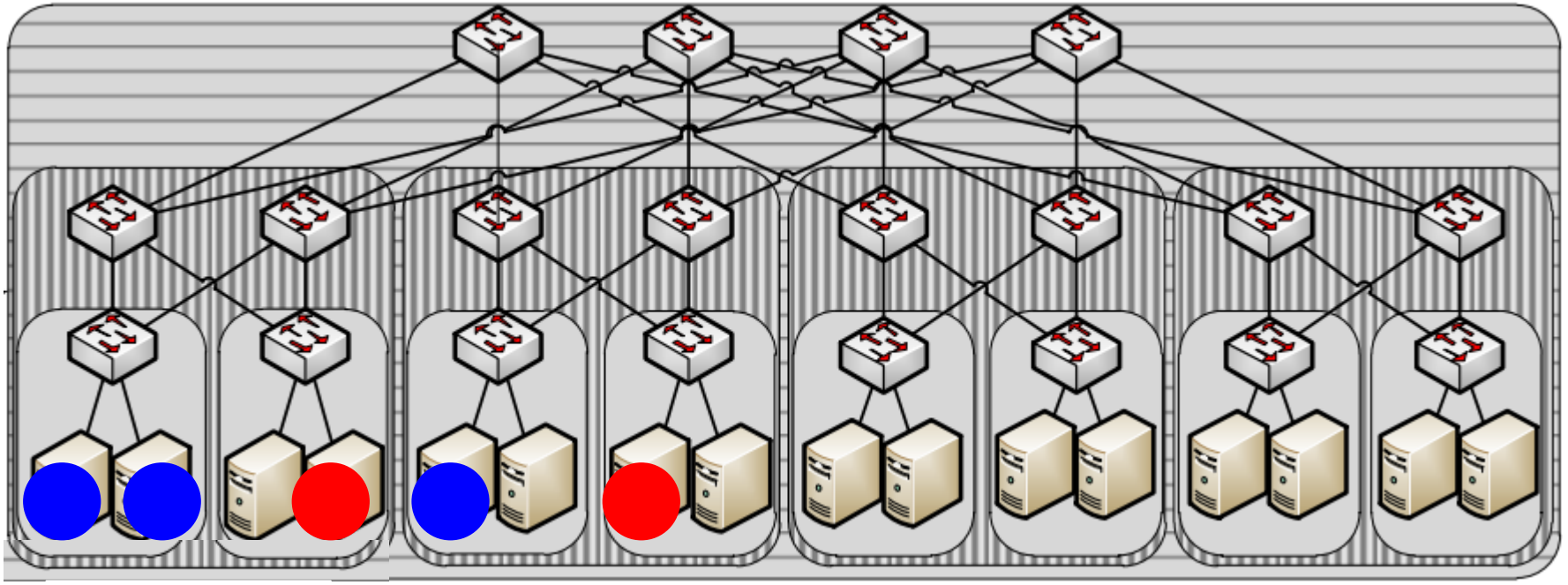

Service deployment, bandwidth guarantees, ...

# Datacenter VNet: Predictable Performance.



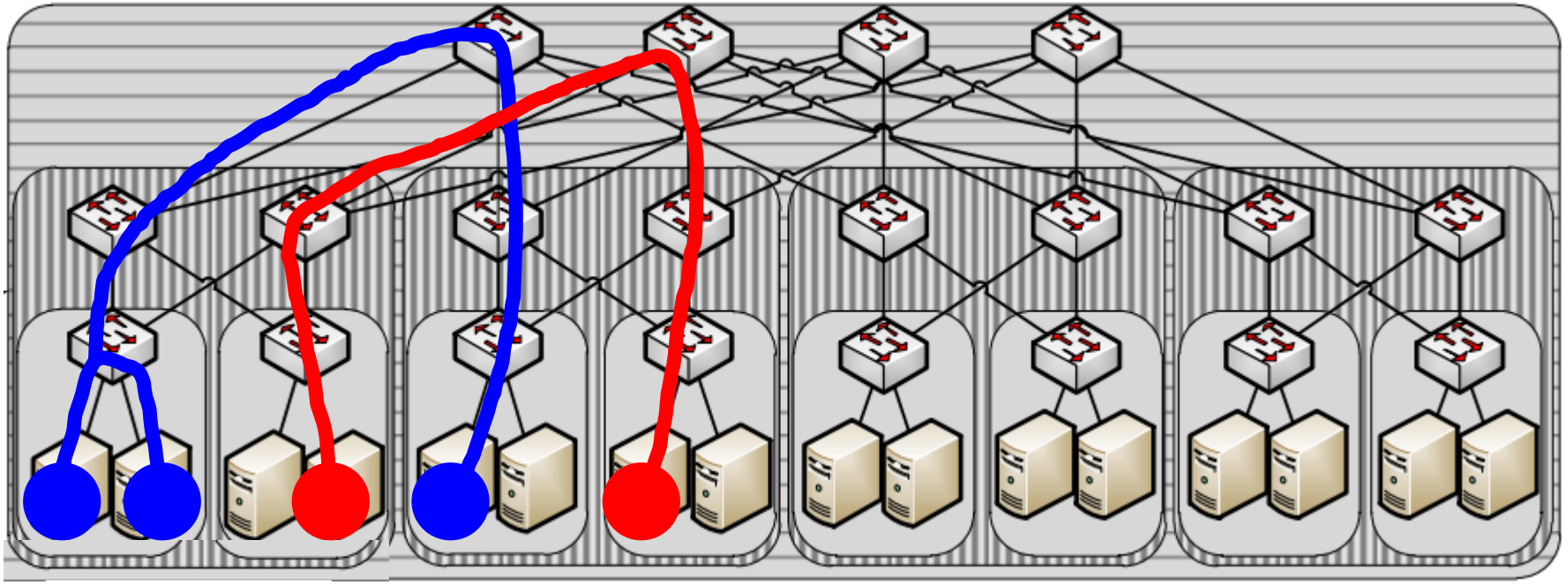
Today: only VMs come with performance isolation (if at all).

# Datacenter VNet: Predictable Performance.



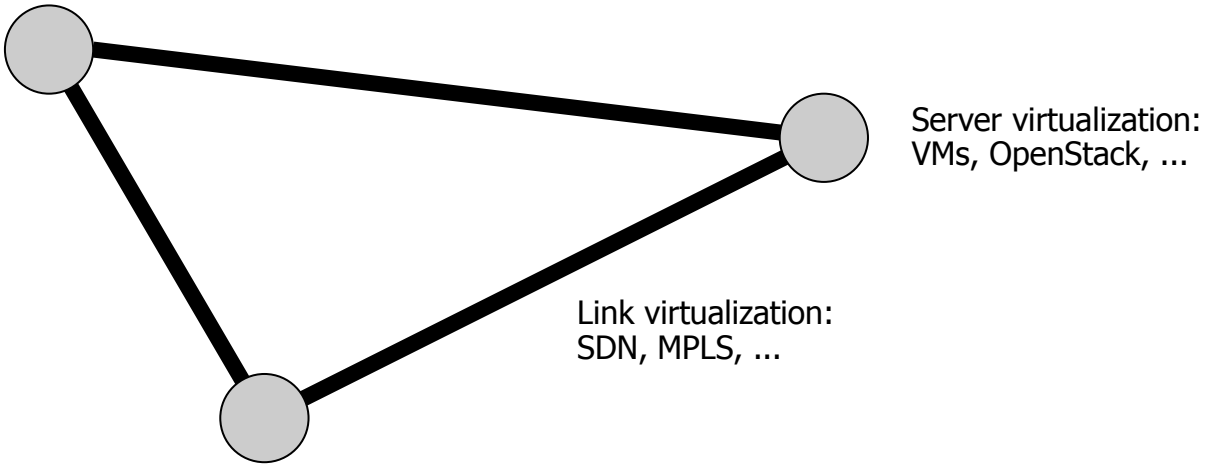
Today: only VMs come with performance isolation (if at all).

# Datacenter VNet: Predictable Performance.



Today: only VMs come with performance isolation (if at all).
Without network guarantees: unpredictable, varying, costly application performance.

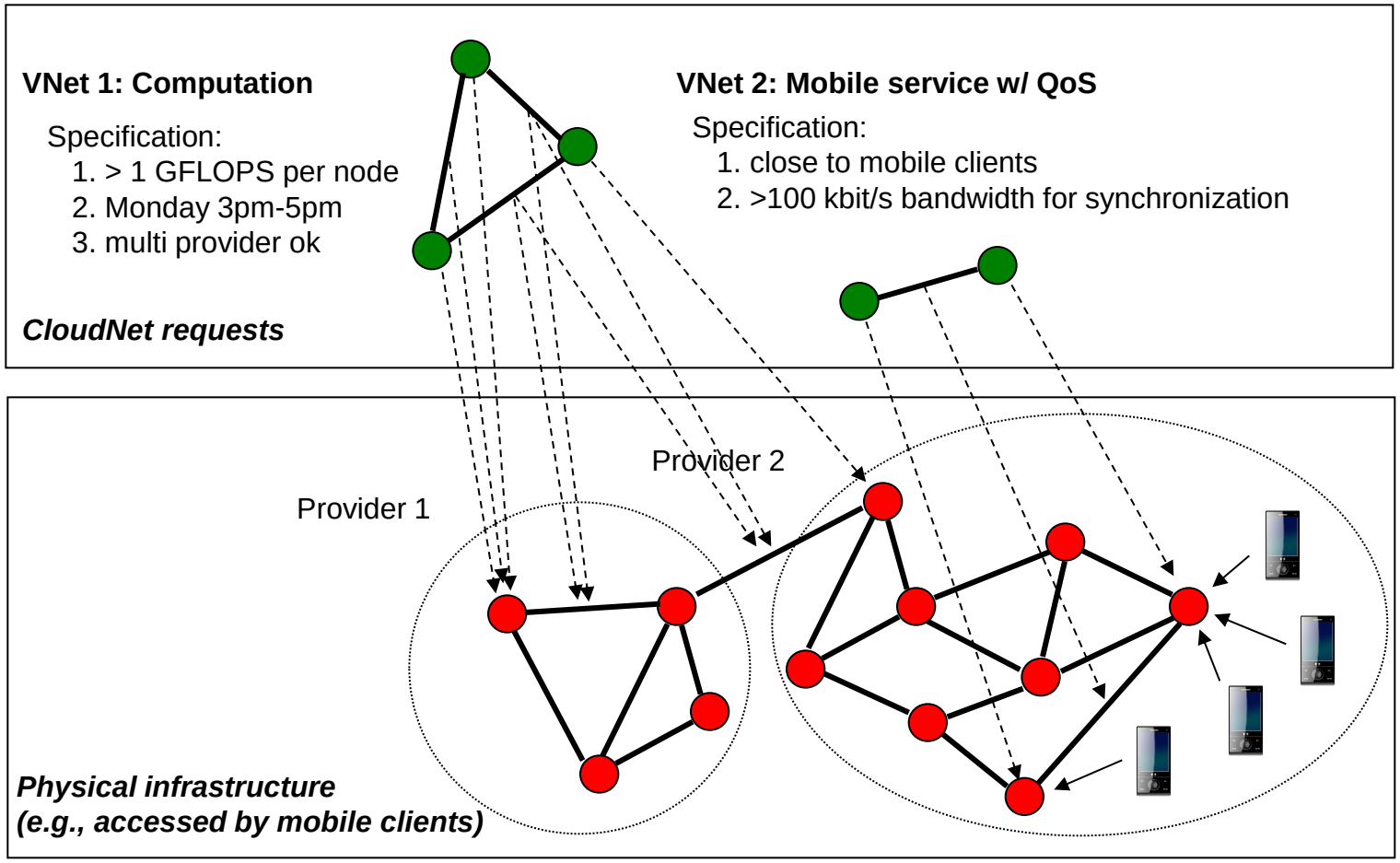
# Virtualization and VNet



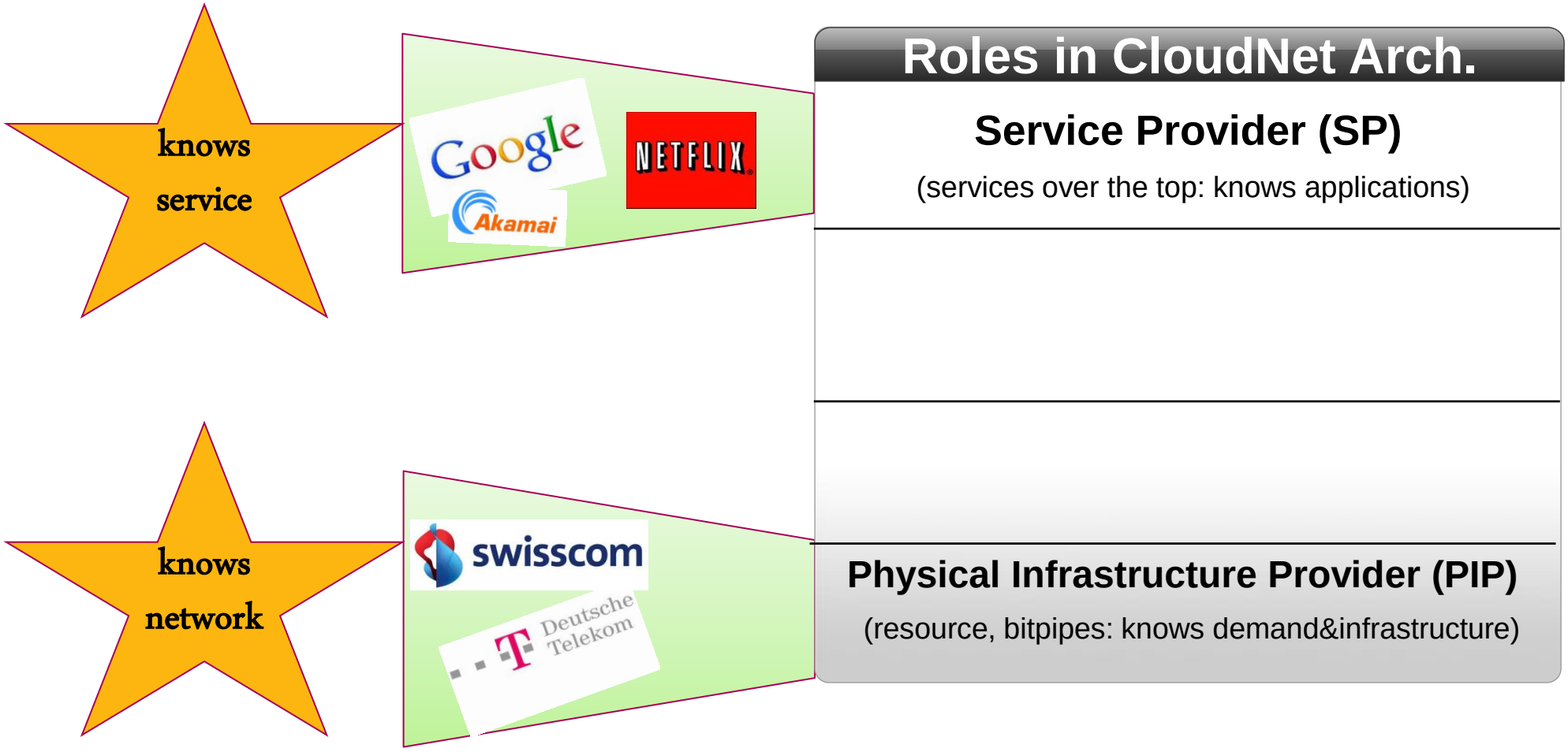

Virtualization trend starts to spill-over to the network.

Benefit: decouple service from physical constraints, supports flexible embeddings and seamless migrations.

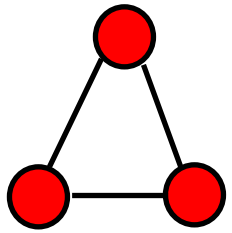
# A Graph Embedding Problem!



# Our Prototype Architecture.



# Our Prototype Architecture.




**Provide L2 topology: resource and management interfaces, provides indirection layer, across PIPs!**

**Can be recursive.**

## Roles in CloudNet Arch.

**Service Provider (SP)**

(services over the top: knows applications)

**Virtual Network Provider (VNP)**

(resource broker, compiles resources)

**Physical Infrastructure Provider (PIP)**

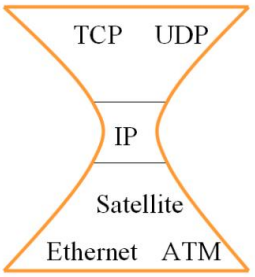
(resource, bitpipes: knows demand&infrastructure)

# Our Prototype Architecture.

**Build upon layer 2: clean slate!**

**Tailored towards application (OSN, …): routing, addressing, multi-path/redundancy…**

**E.g., today's Internet.**


IP hourglass

## Roles in CloudNet Arch.

**Service Provider (SP)**

(services over the top: knows applications)

**Virtual Network Operator (VNO)**

(operates VNet, Layer 3+, triggers migration)

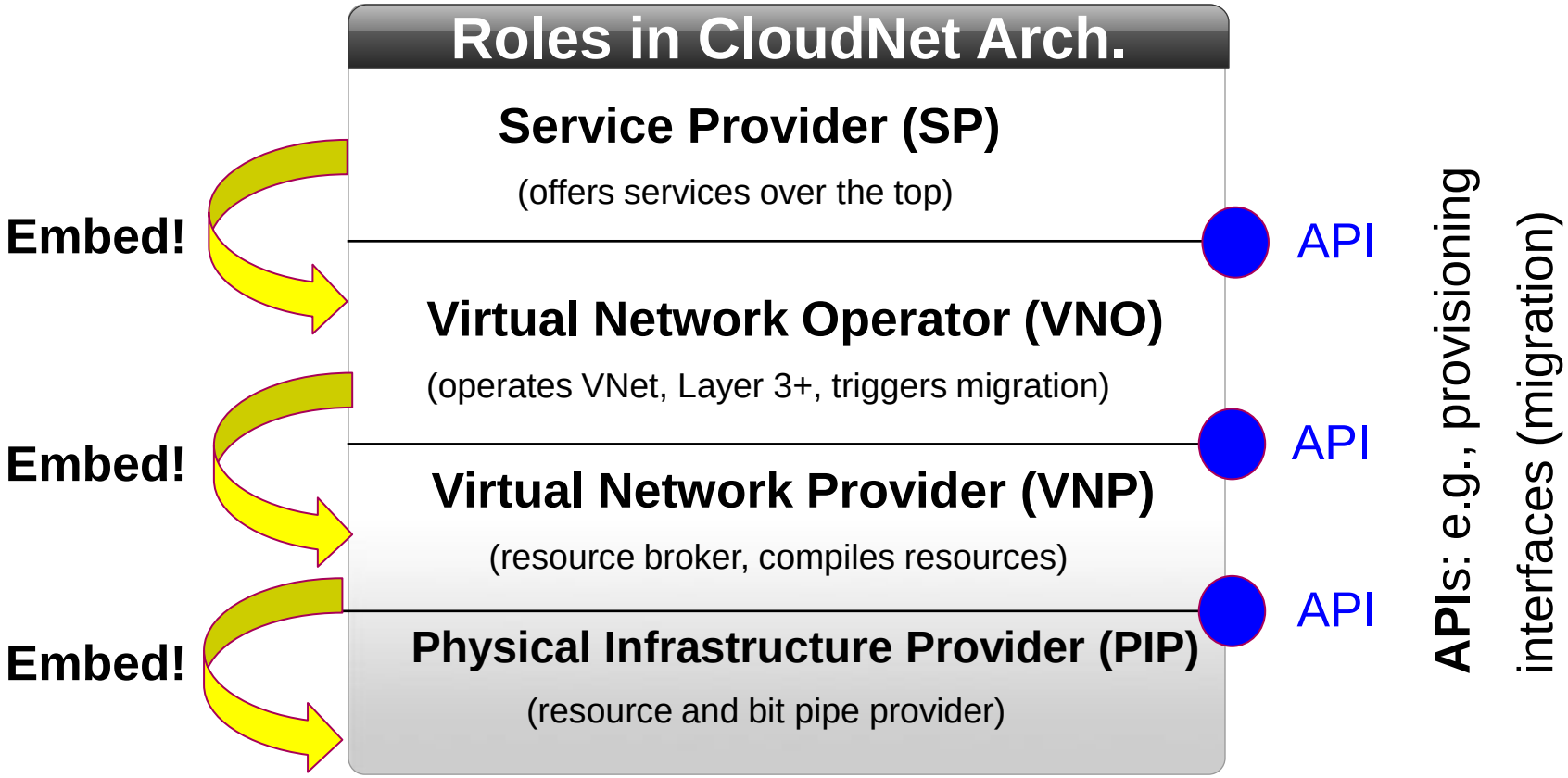
**Virtual Network Provider (VNP)**

(resource broker, compiles resources)

**Physical Infrastructure Provider (PIP)**

(resource, bitpipes: knows demand&infrastructure)

# Our Prototype Architecture.



# Our Prototype Architecture.

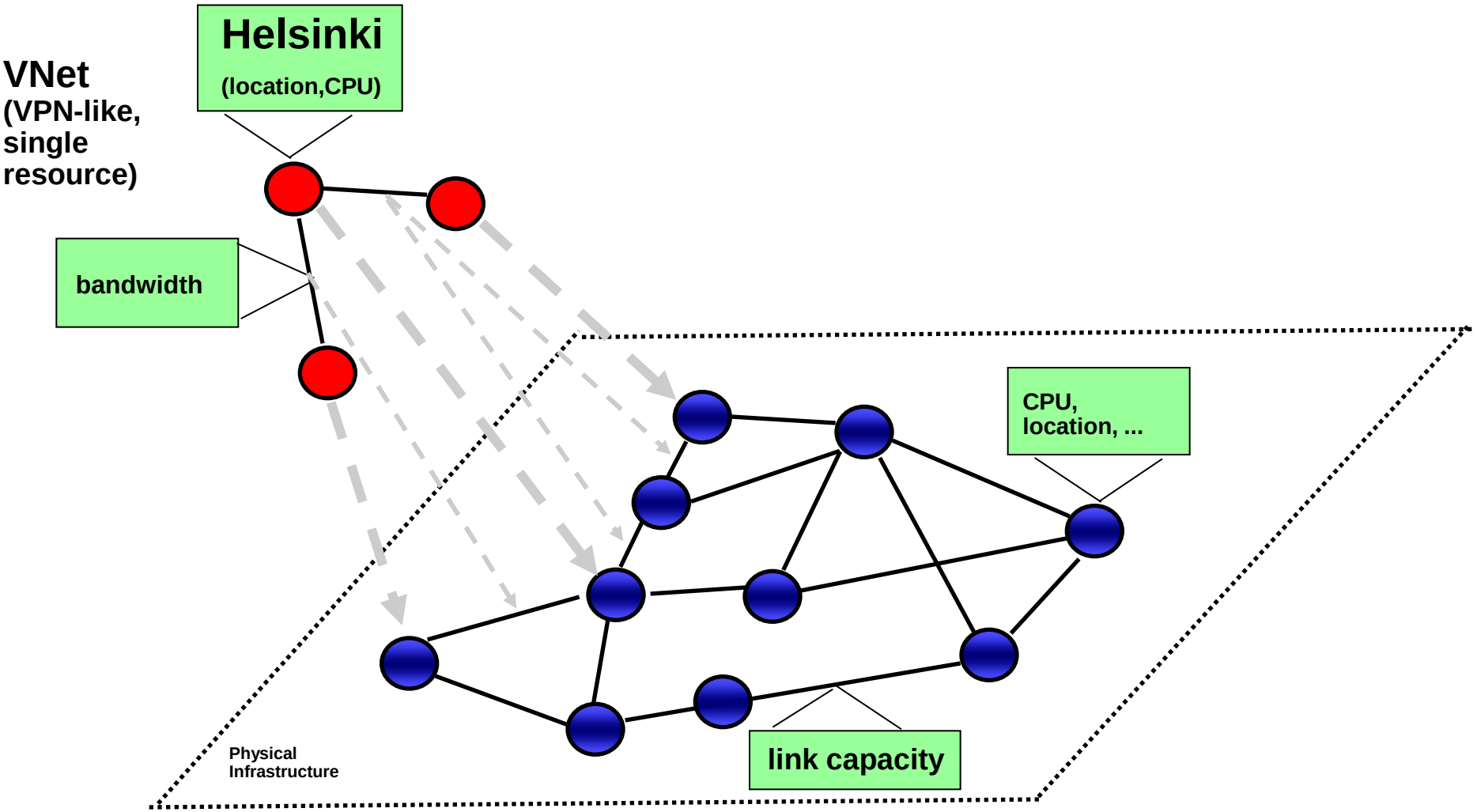
(resource and bit pipe provider)


Internships...

# Talk Outline:

1. **Which VNets to accept? (And how to embed?)**
2. Threats: VNet embeddings with bad intentions?
3. Migrating VNets
4. VNets with in-network processing

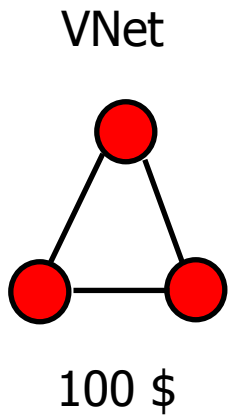
# Competitive Access Control: Model (1)



# Competitive Access Control: Model (2)

Specification of CloudNet request:

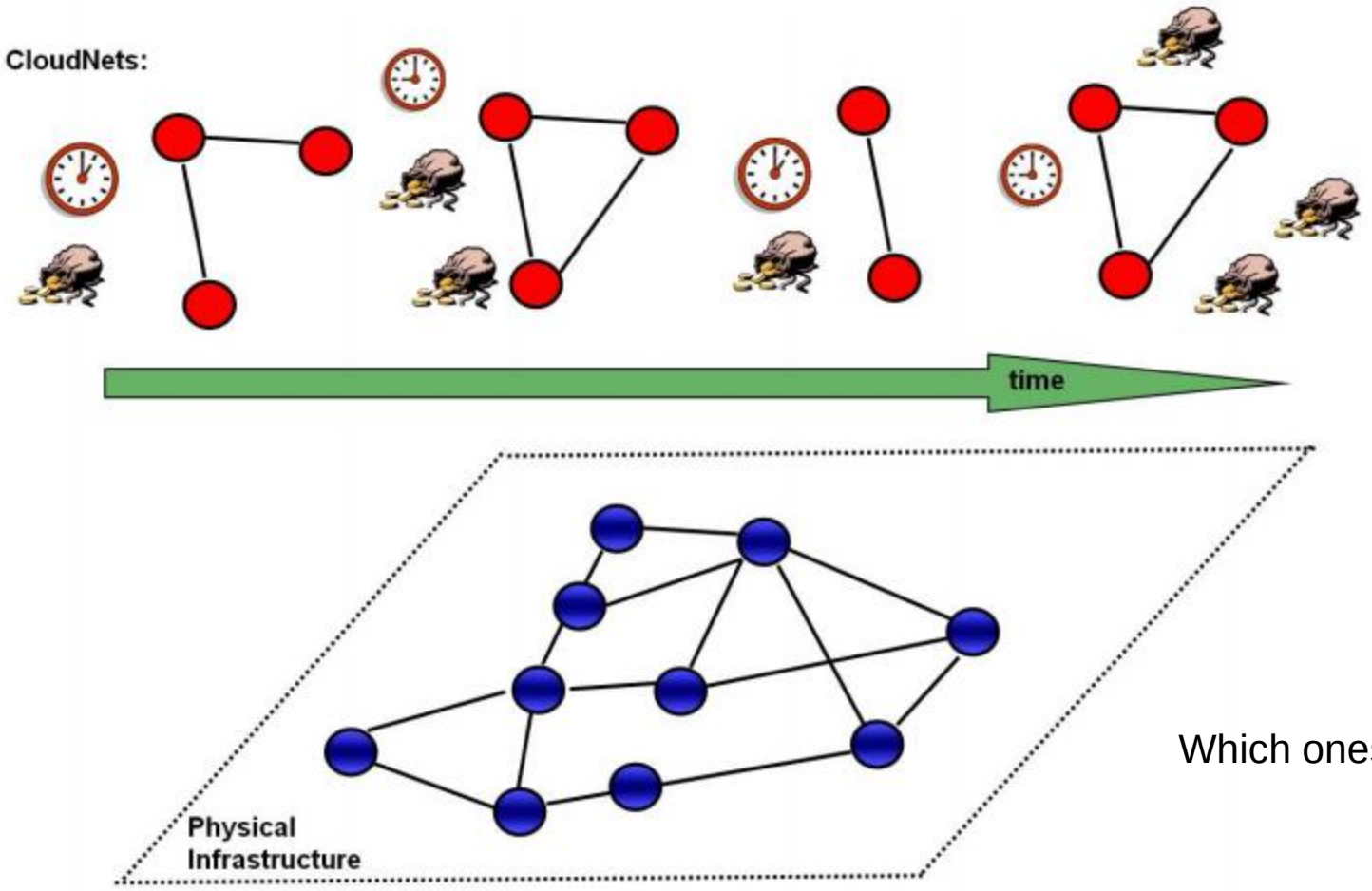
- terminal locations to be connected
- benefit if CloudNet accepted (all-or-nothing, no preemption)
- desired bandwidth and allowed traffic patterns
- a routing model
- duration (from when until when?)




If VNets with these specifications arrive over time, which ones to accept online?

Objective: maximize **sum of benefits** of accepted VNets

# Competitive Access Control: Model (3)

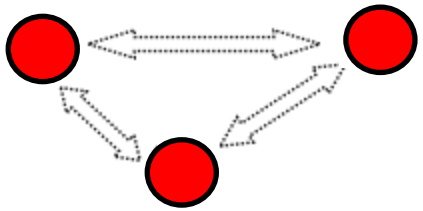



Which ones to accept?

# VNet Specifications (1): Traffic Model.

## Customer Pipe

Every pair (u,v) of nodes requires a certain bandwidth.



Detailed constraints, only this traffic matrix needs to be fulfilled!

## Hose Model

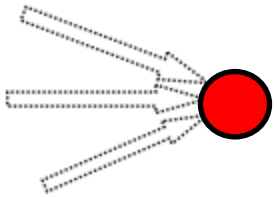
Each node v has max ingress and max egress bandwidth: each traffic matrix fulfilling them must be served.

More flexible, must support many traffic matrices!

«virtual switch»

## Aggregate Ingress Model

Sum of ingress bandwidths must be at most a parameter I.

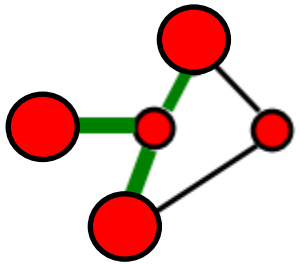


Simple and flexible! Good for multicasts etc.: no overhead, duplicate packets for output links, not input links already!

# VNet Specifications (2): Routing Model.

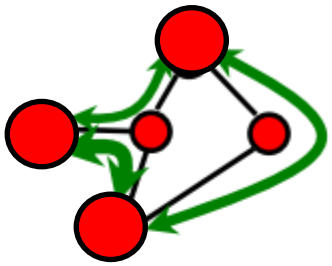
## Tree

VNet is embedded as Steiner tree:



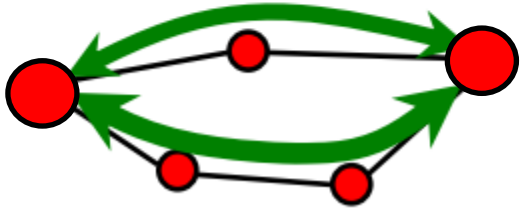
## Single Path

Each pair of nodes communicates along a single path.



## Multi Path

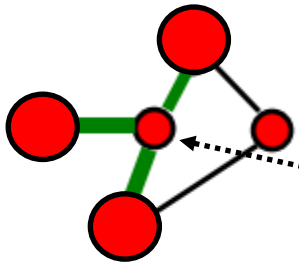
A linear combination specifies split of traffic between two nodes.



# VNet Specifications (2): Routing Model.

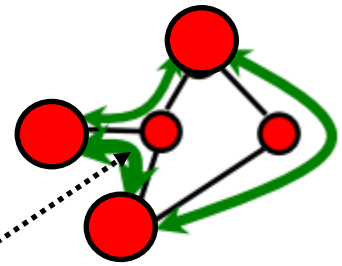
## Tree

VNet is embedded as Steiner tree:



## Single Path

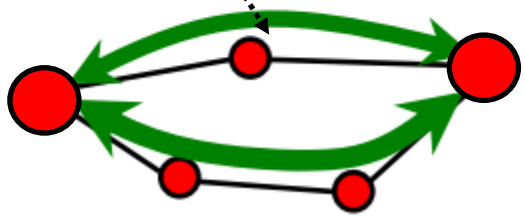
Each pair of nodes communicates along a single path.



Relay nodes may add to embedding costs!
(resources depend, e.g., on packet rate)

## Multi Path

A linear combination specifies split of traffic between two nodes.



# Competitive Embeddings.

Competitive analysis framework:

## Online Algorithm

Online algorithms make decisions at time t without any knowledge of inputs / requests at times t'>t.

## Competitive Ratio

Competitive ratio r,

r = Cost(ALG) / cost(OPT)

The price of not knowing the future!

## Competitive Analysis

An *r-competitive online algorithm* ALG gives a worst-case performance guarantee: the performance is at most a factor r worse than an optimal offline algorithm OPT!

No need for complex predictions but still good!

# Buchbinder&Naor: Primal-Dual Approach.

Algorithm design and analysis follows online primal-dual approach by Buchbinder&Naor!
(Application to general VNet embeddings, traffic&routing models, router loads, duration, approx oracles, ...)

**1. Formulate dynamic primal (covering) and dual (packing) LP**

| $\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ | $\max B_j^T \cdot Y_j \;\; s.t.$ |
|---|---|
| $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ | $A_j \cdot Y_j \leq C$ |
| $X, Z_j \geq \mathbf{0}$ | $D_j \cdot Y_j \leq \mathbf{1}$ |
| | $Y_j \geq \mathbf{0}$ |
| (I) | (II) |

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

**2. Derive algorithm which always produces feasible primal solutions and where Primal >= 2*Dual**

**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) $z_j \leftarrow 0$.

# Result.

## Theorem

The presented online algorithm log-competitive
in the amount of resources in the physical network.
(If capacities can be exceeded by a log factor, it is even constant competitive.)

However, competitive ratio also depends on max benefit.

# Algorithm and Proof Sketch (1).

**Embedding oracle: algo invokes an oracle procedure to determine cost of VNet embedding!**

**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) $z_j \leftarrow 0$.

# Algorithm and Proof Sketch (1).

**If resource cost lower than benefit: accept!**

**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) $z_j \leftarrow 0$.

# Algorithm and Proof Sketch (1).

**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) $z_j \leftarrow 0$.

**update allocations for accepted VNet...**

# Algorithm and Proof Sketch (1).

**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) $z_j \leftarrow 0$.

**otherwise reject**
**(no change in substrate)**

# Algorithm and Proof Sketch (1).

**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) $z_j \leftarrow 0$.

**otherwise reject**
**(no change in substrate)**

**Algorithm efficient... except for oracle (static, optimal embedding)!**
**What if we only use a suboptimal embedding here?**

# Algorithm and Proof Sketch (2).

Problem: computation of optimal embeddings NP-hard!
Thus: use approximate embeddings! (E.g., Steiner tree)

**GIPO:**

**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) $z_j \leftarrow 0$.

Competitive ratio ρ

**Embedding approx.:**

**<insert your favorite approx algo>**

**Approx ratio r**

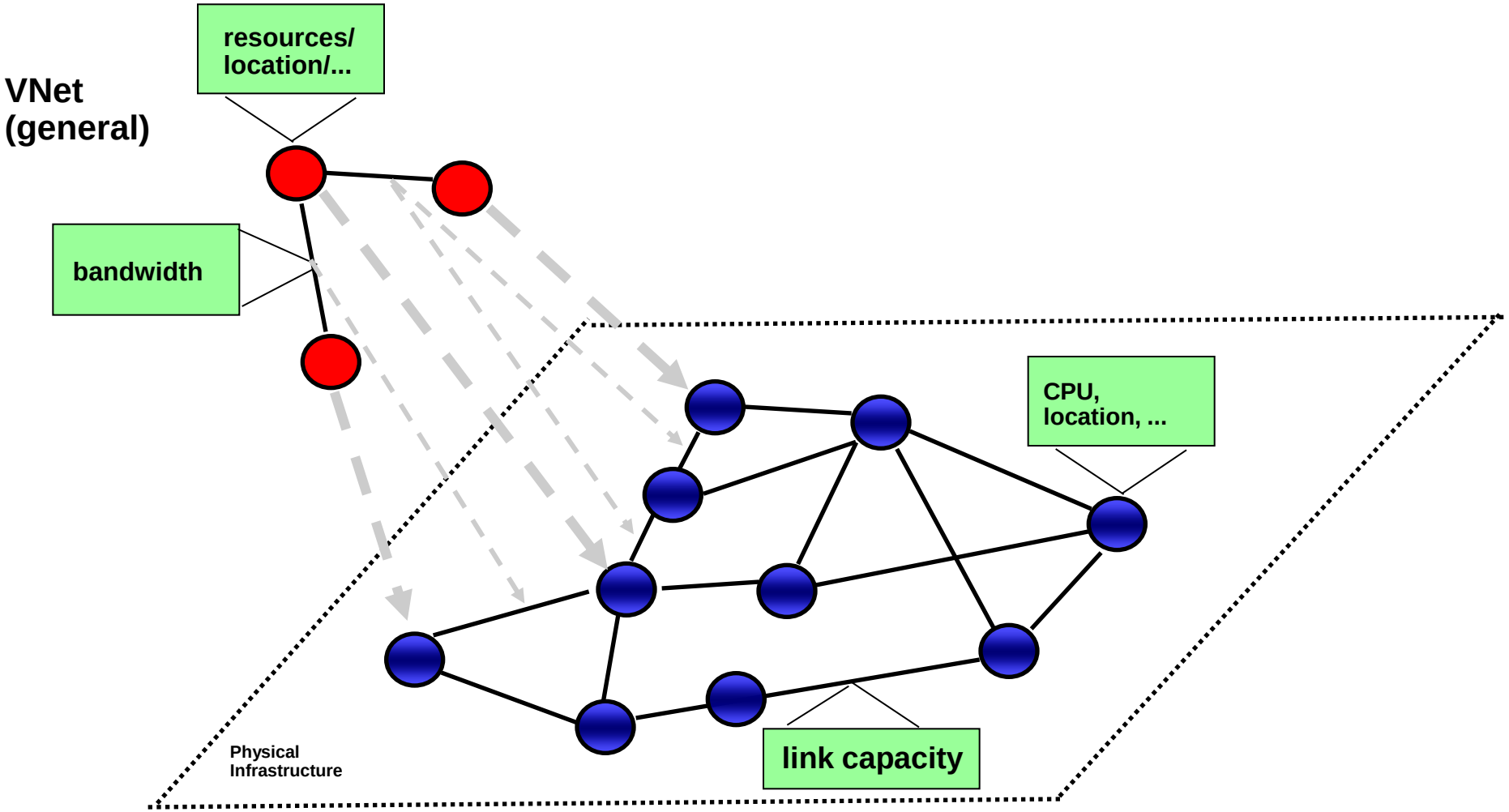
## Lemma

**The approximation does not reduce the overall competitive ratio by much: we get ρ*r ratio!**

# Talk Outline:

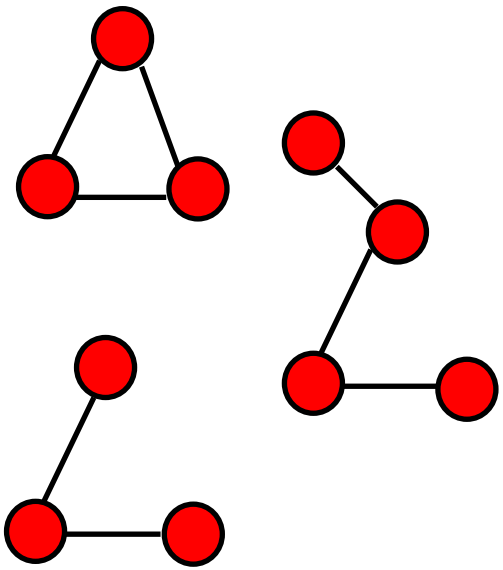
1. Which VNets to accept? (And how to embed?)
2. **Threats: VNet embeddings with bad intentions?**
3. Migrating VNets
4. VNets with in-network processing

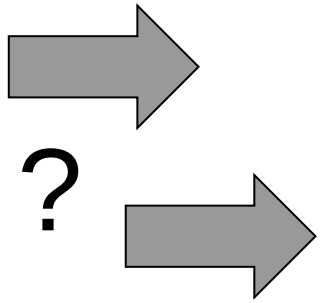
# Flexible Embeddings: Beyond VPN Model.

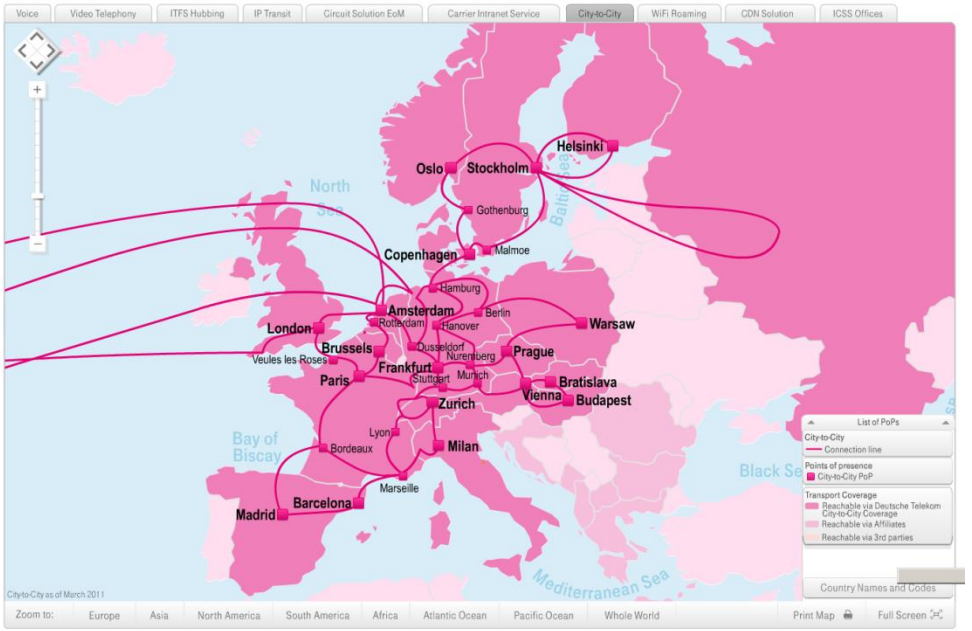


# Security Issues.

- Are VNet embeddings a threat for ISPs?
- Do embeddings leak information about infrastructure?

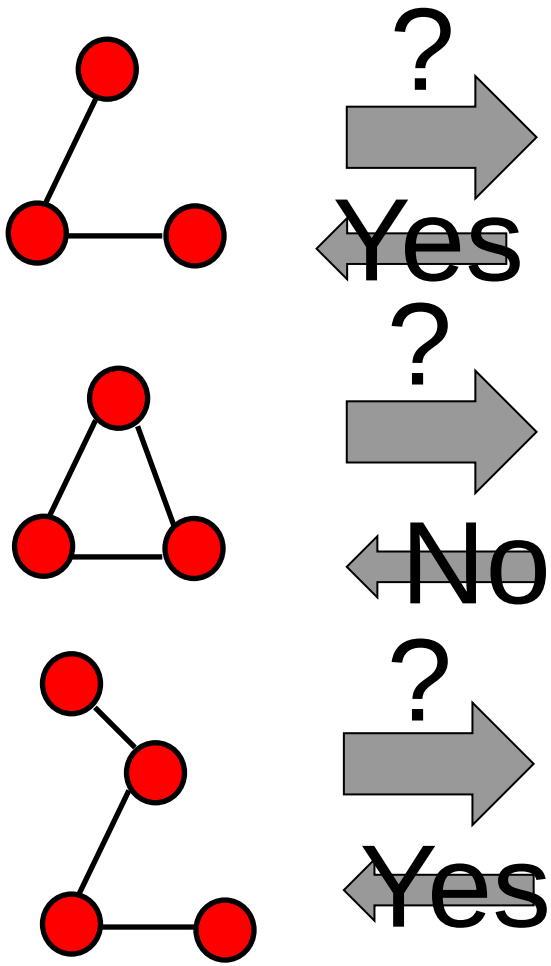




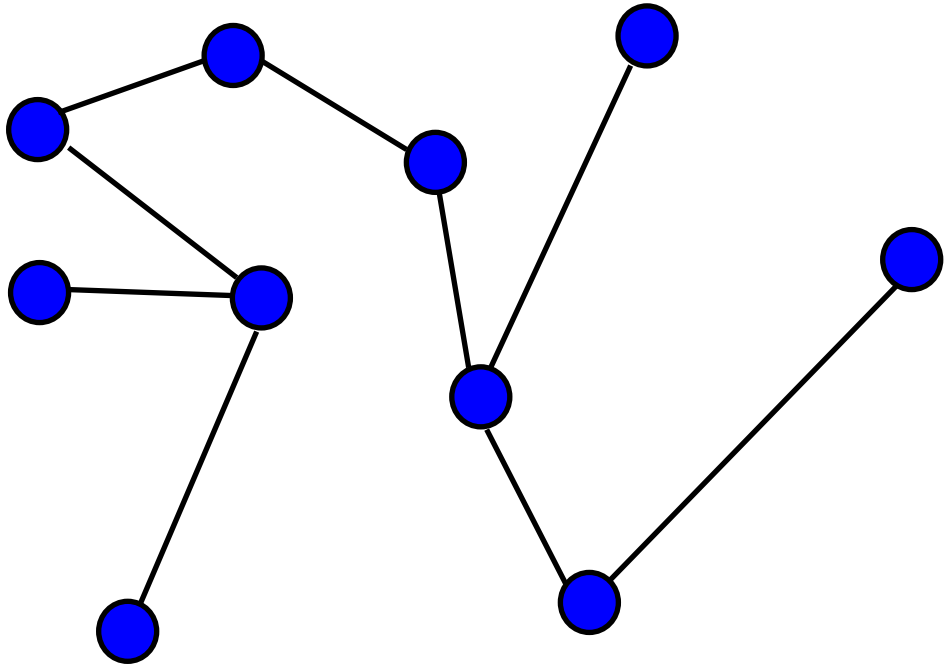
# Request Complexity.

Are VNet embeddings a threat for ISPs?

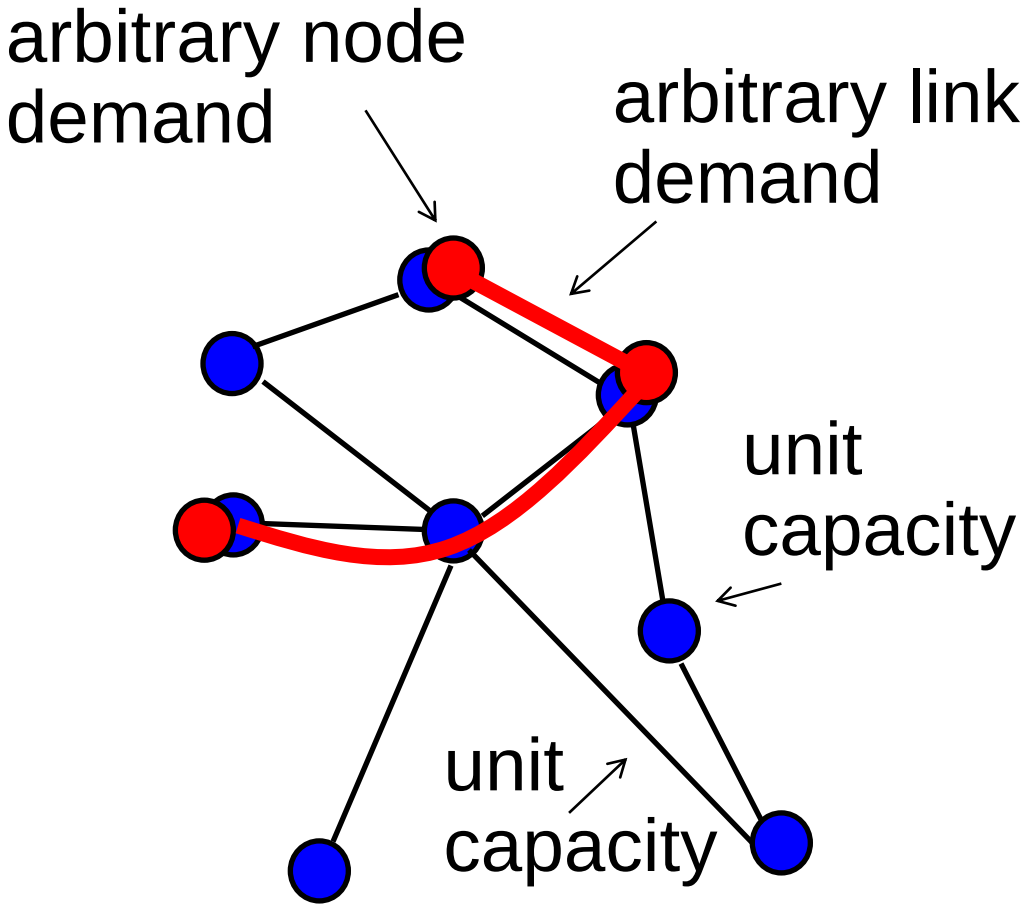



## Request Complexity

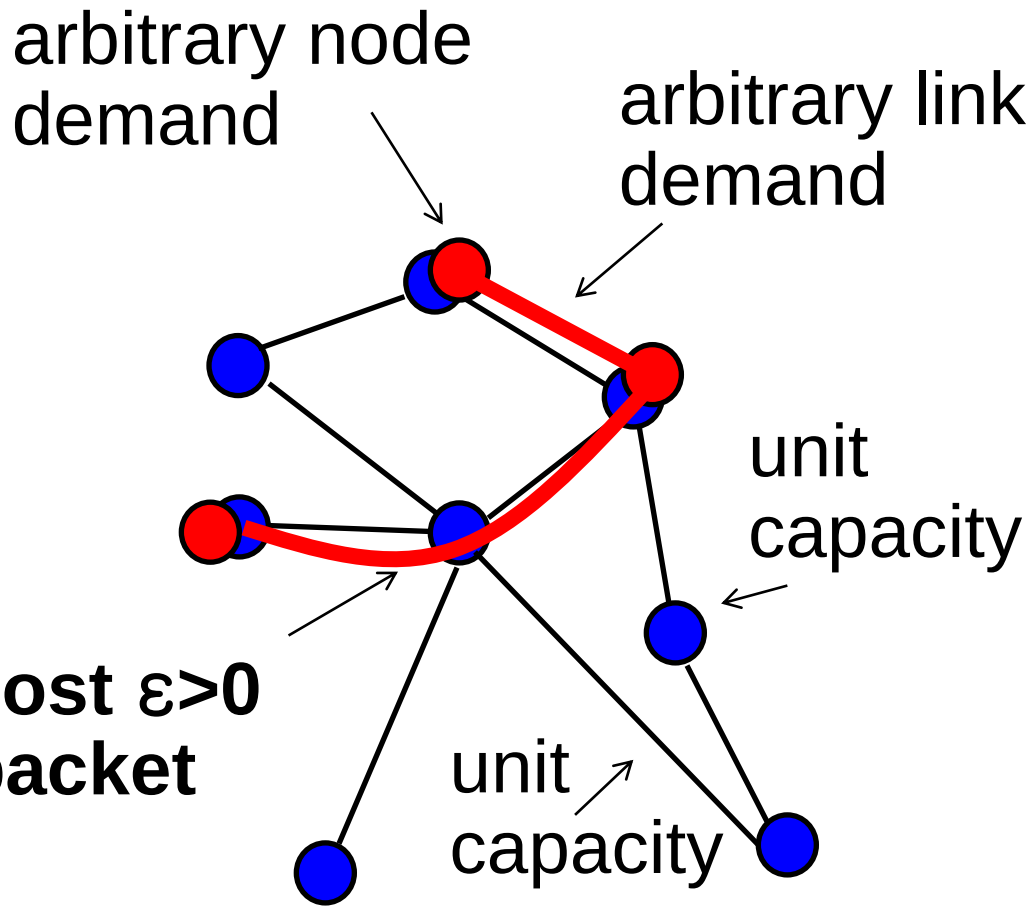
How many embeddings needed to fully reveal topology?



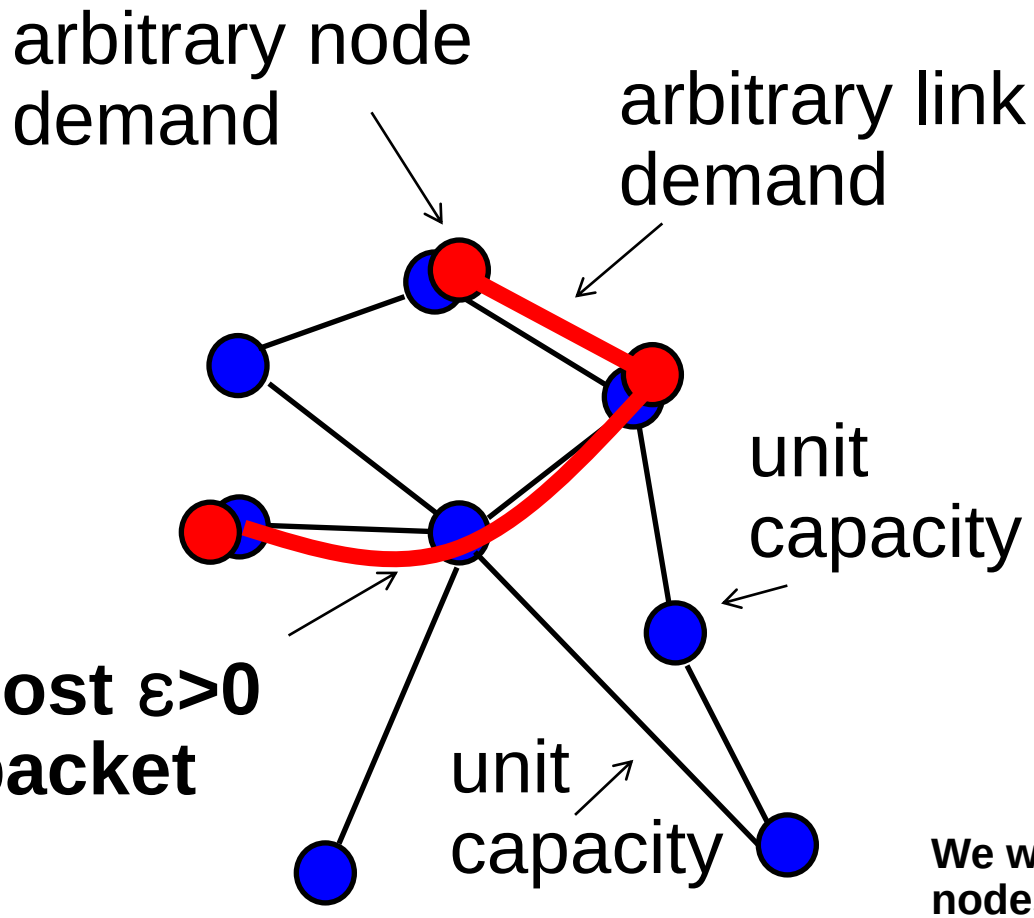
# Embedding Model.



# Embedding Model.



# Embedding Model.

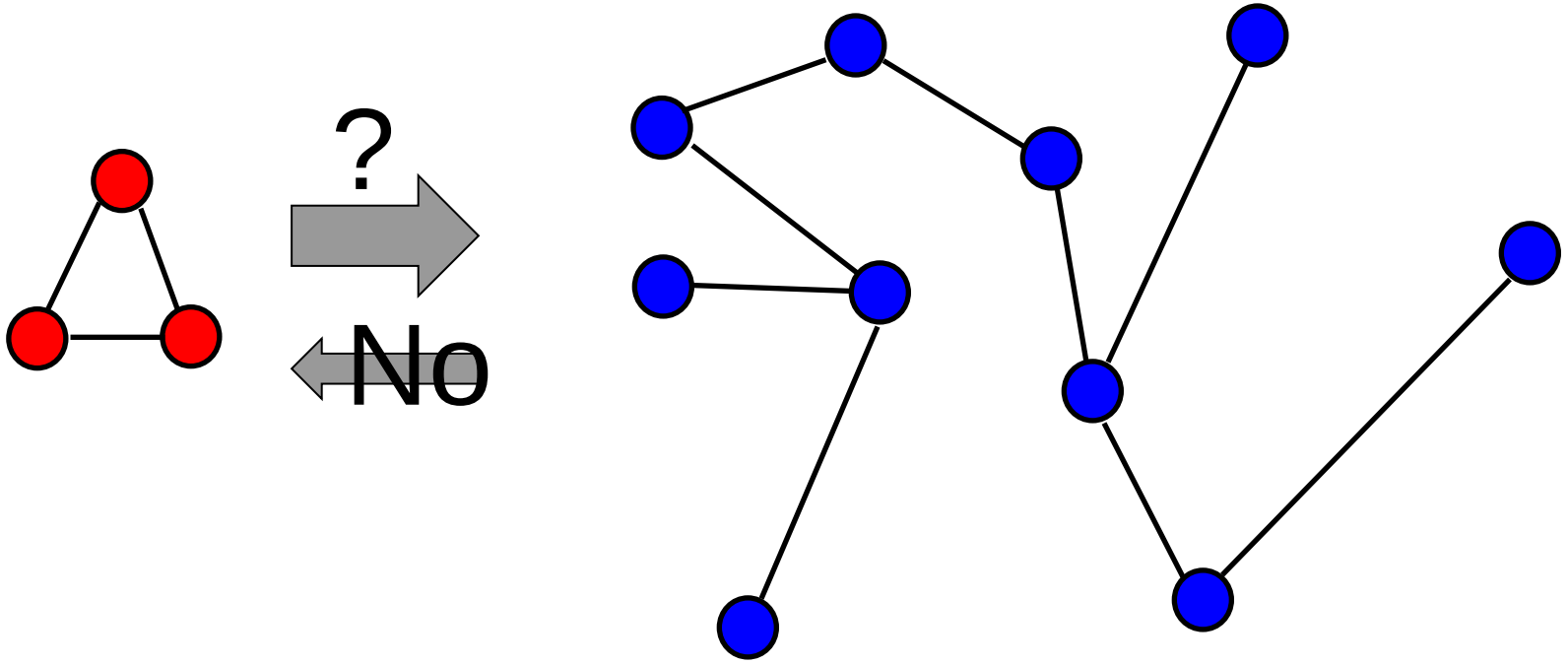



**relay cost ε>0 (e.g., packet rate)**

**We will ask for unit capacities on nodes and links!**
**Essentially a graph immersion problem:** disjoint paths for virtual links...

# Some Properties Simple...

«Is the network 2-connected?»

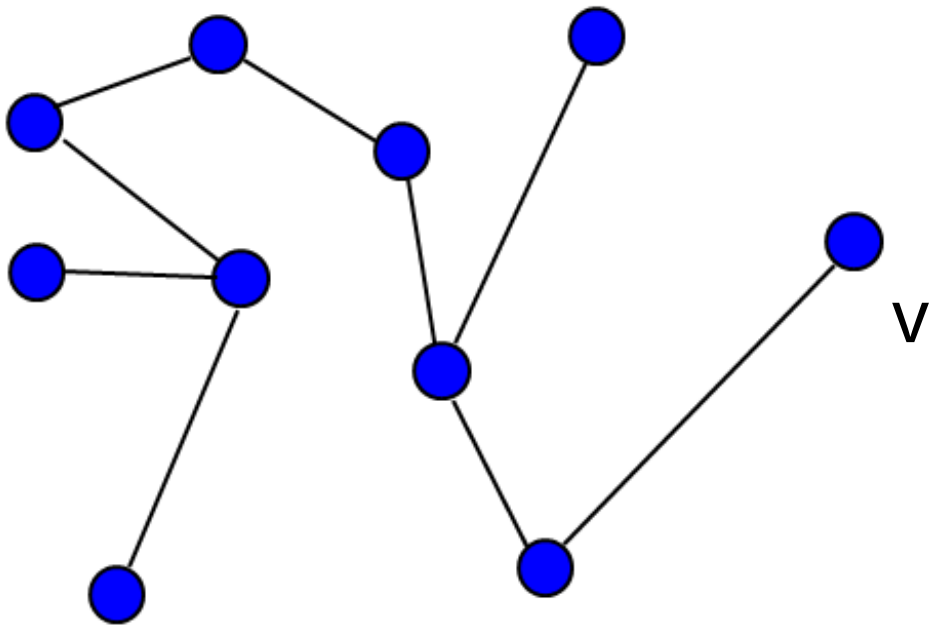


# Example: Tree.

**How to discover a tree?**

Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node

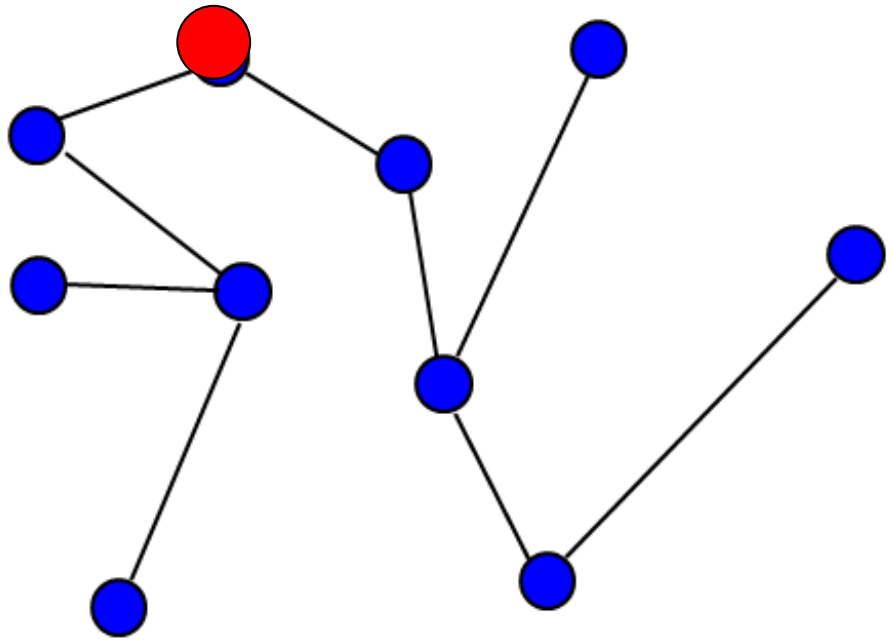


# Example: Tree.

**How to discover a tree?**

Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node

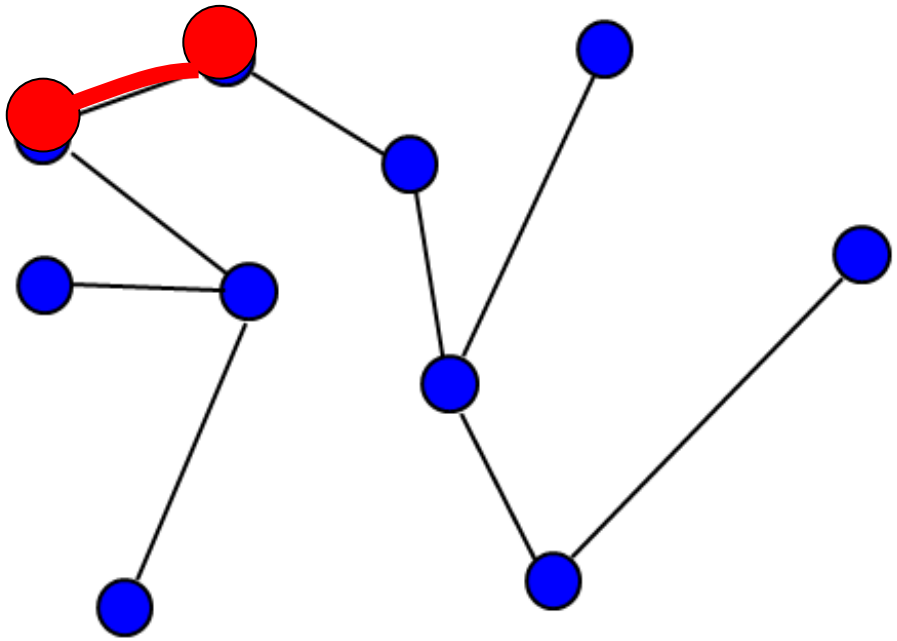


# Example: Tree.

**How to discover a tree?**

Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node

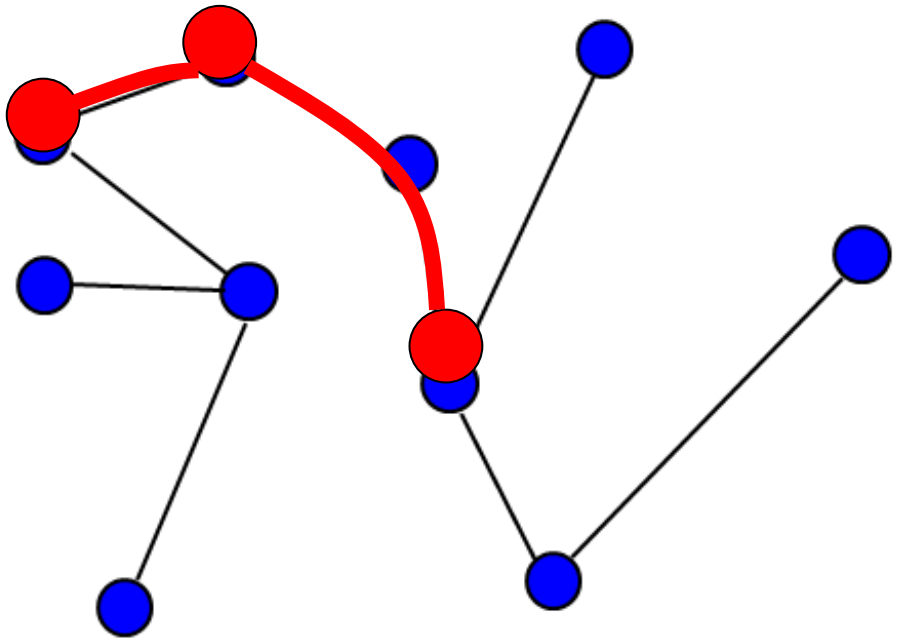


# Example: Tree.

**How to discover a tree?**

Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node

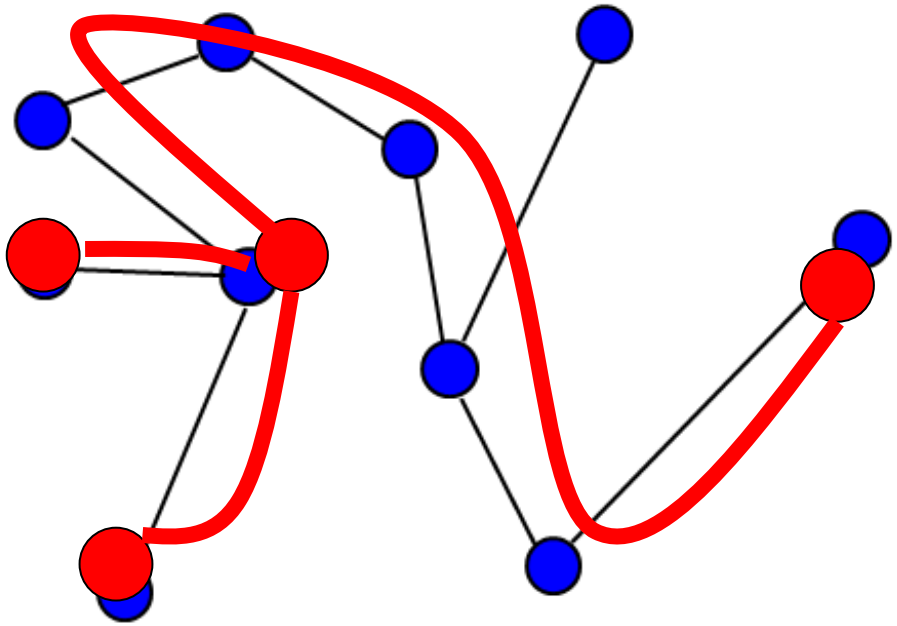


# Example: Tree.

**How to discover a tree?**

Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node

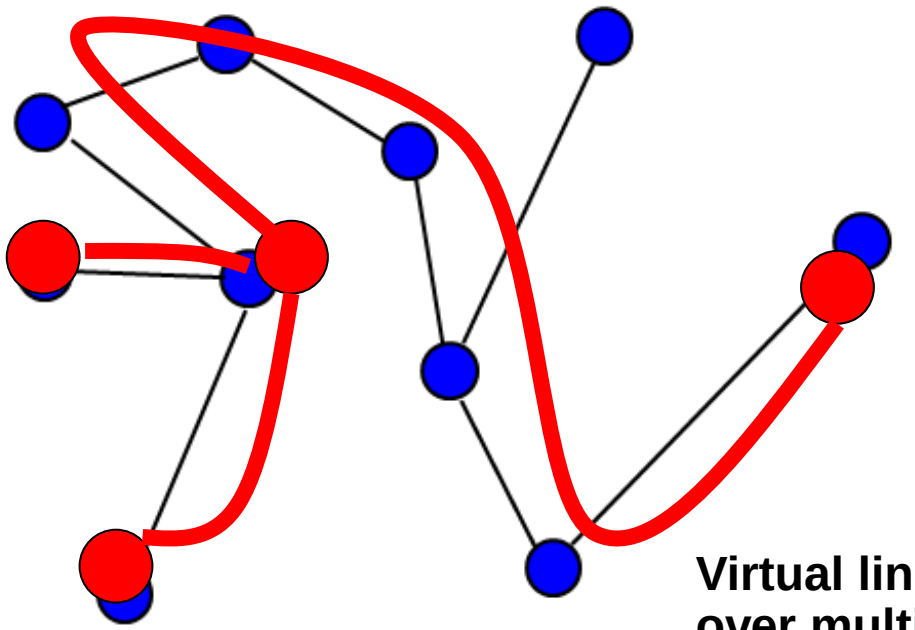


# Example: Tree.

**How to discover a tree?**

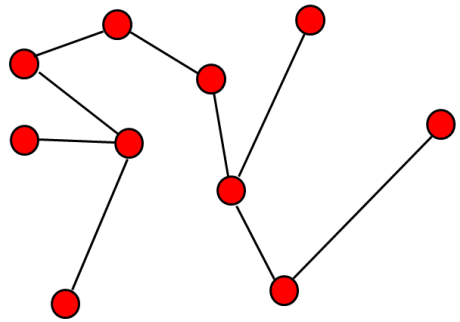
Graph growing:

1. Test whether triangle fits? (loop-free)
2. Try to add neighbors to node as long as possible, then continue with other node



**Virtual links may be embedded over multiple physical links!**

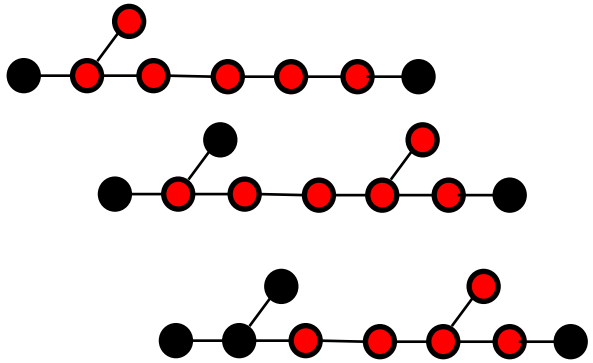
# Tree Solution: Graph Growing.



**TREE ALGORITHM: line strategy**

1. Binary search on longest path («anchor»):

2. Last and first node *explored*, explore «branches» at pending nodes



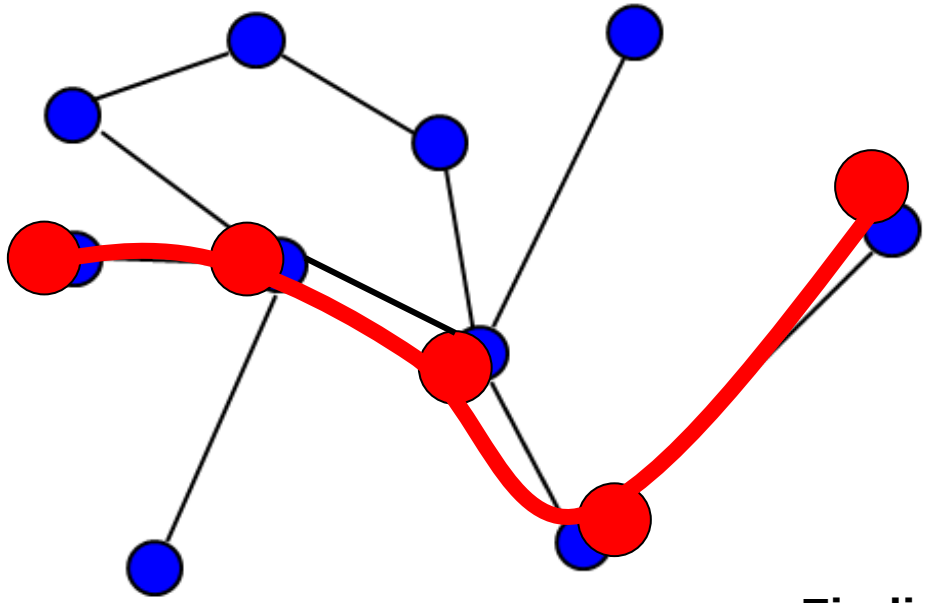
**Amortized Analysis**:
Per discovered physical link at most one query, plus at most one per physical node (no incident links).

**Algorithm 1** Tree Discovery: TREE

1: $G := \{\{v\}, \emptyset\}$ /* current request graph */
2: $\mathcal{P} := \{v\}$ /* pending set of unexplored nodes*/
3: **while** $\mathcal{P} \neq \emptyset$ **do**
4: choose $v \in \mathcal{P}$, $S :=$*exploreSequence*$(v)$
5: **if** $S \neq \emptyset$ **then**
6: $G := GvS$, add all nodes of $S$ to $\mathcal{P}$
7: **else**
8: remove $v$ from $\mathcal{P}$

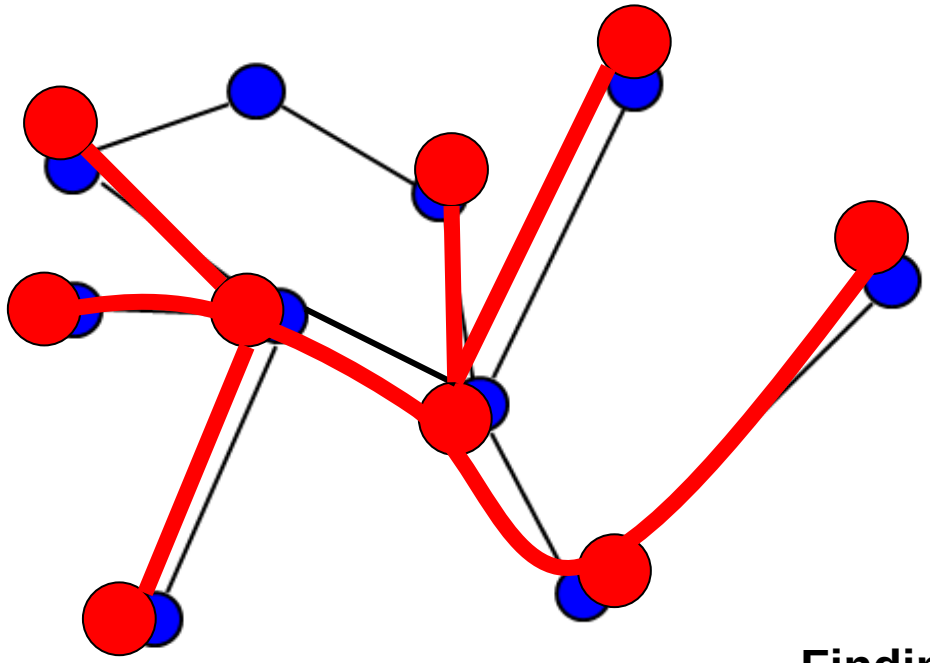
*exploreSequence*$(v)$
1: $S := \emptyset$
2: **if** `request`$(GvC, H)$ **then**
3: find max $j$ s.t. $GvC^j \mapsto H$ (binary search)
4: $S := C^j$
5: **return** $S$

# Greedy Graph Growing on General Graphs? (1)



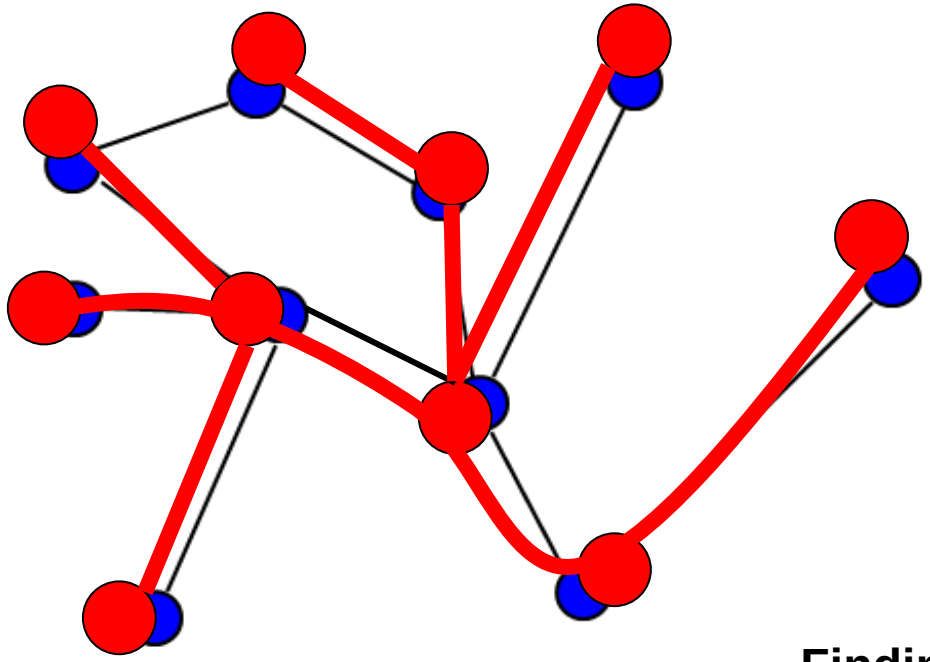
**Finding path...**

# Greedy Graph Growing on General Graphs? (1)



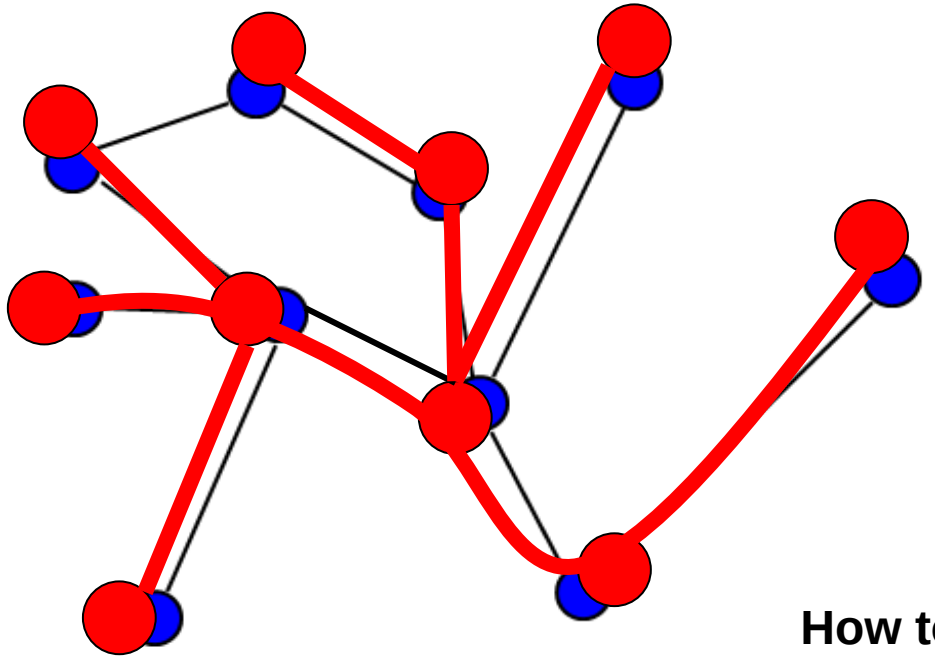
**Finding neighbors...**

# Greedy Graph Growing on General Graphs? (1)



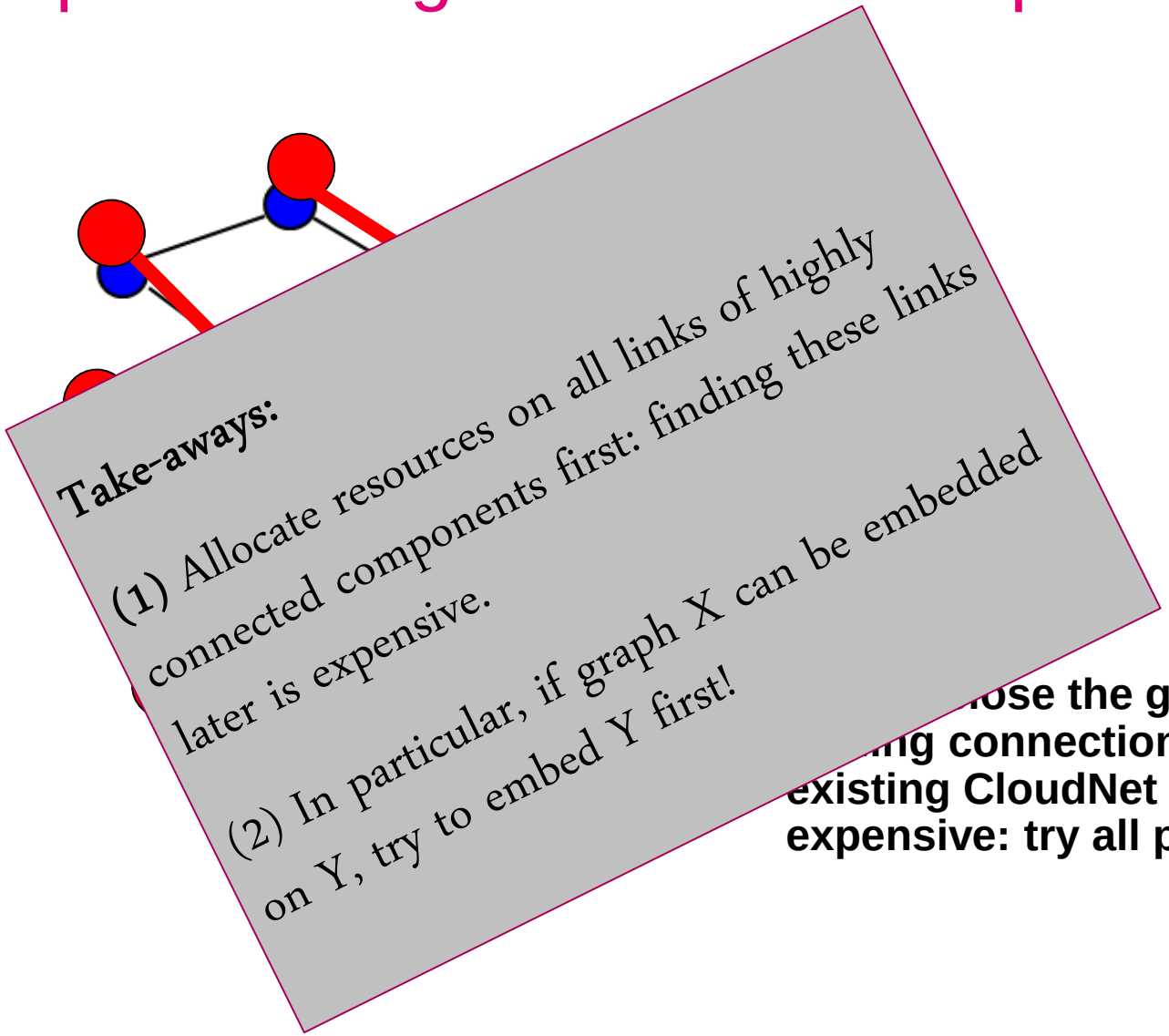
**Finding more neighbors...**

# Greedy Graph Growing on General Graphs? (1)



**How to close the gap?**
**Adding connections between existing CloudNet nodes is expensive: try all pairs!**

# Greedy Graph Growing on General Graphs? (1)

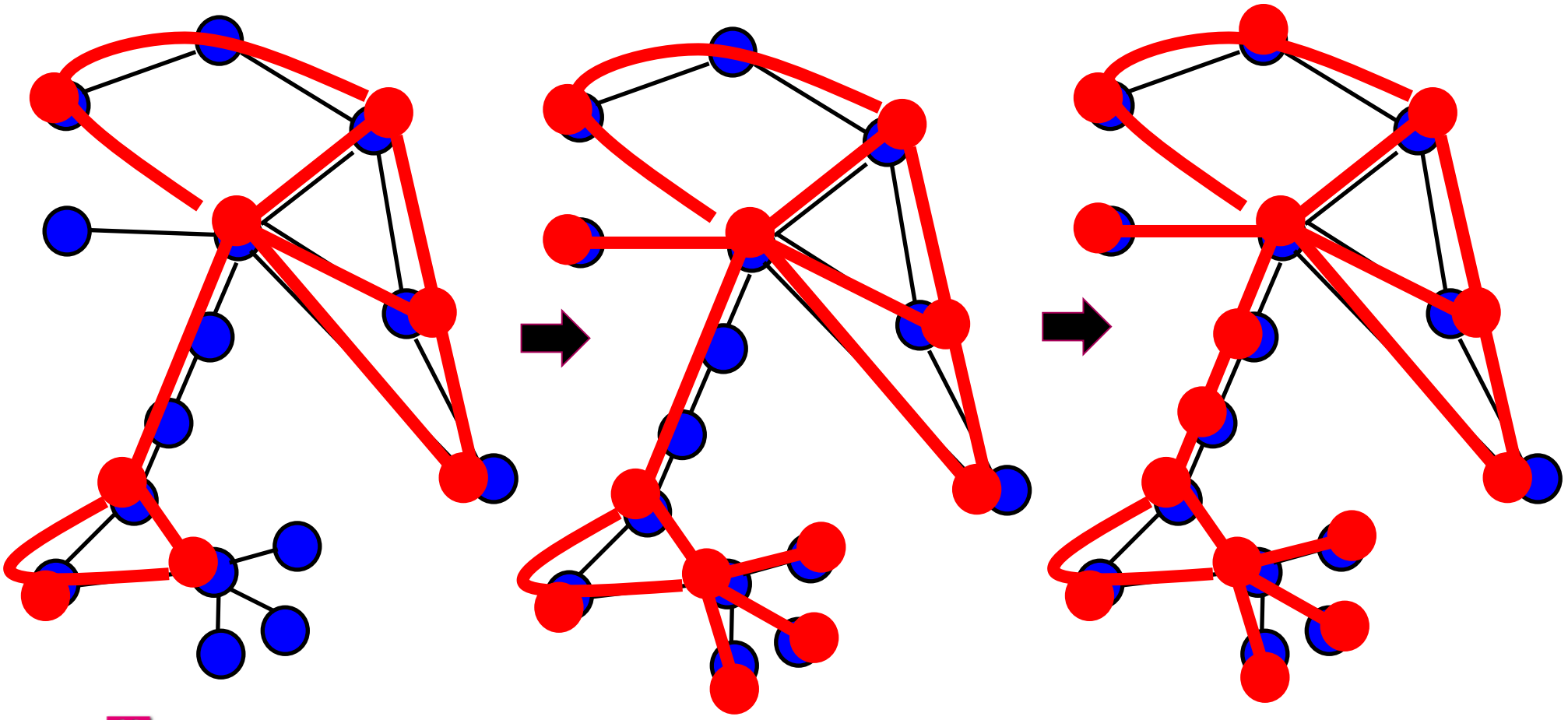



**…lose the gap?**
**…ing connections between existing CloudNet nodes is expensive: try all pairs!**

# Greedy Graph Growing on General Graphs? (2)

Simple solution: First try to find the «knitting»!

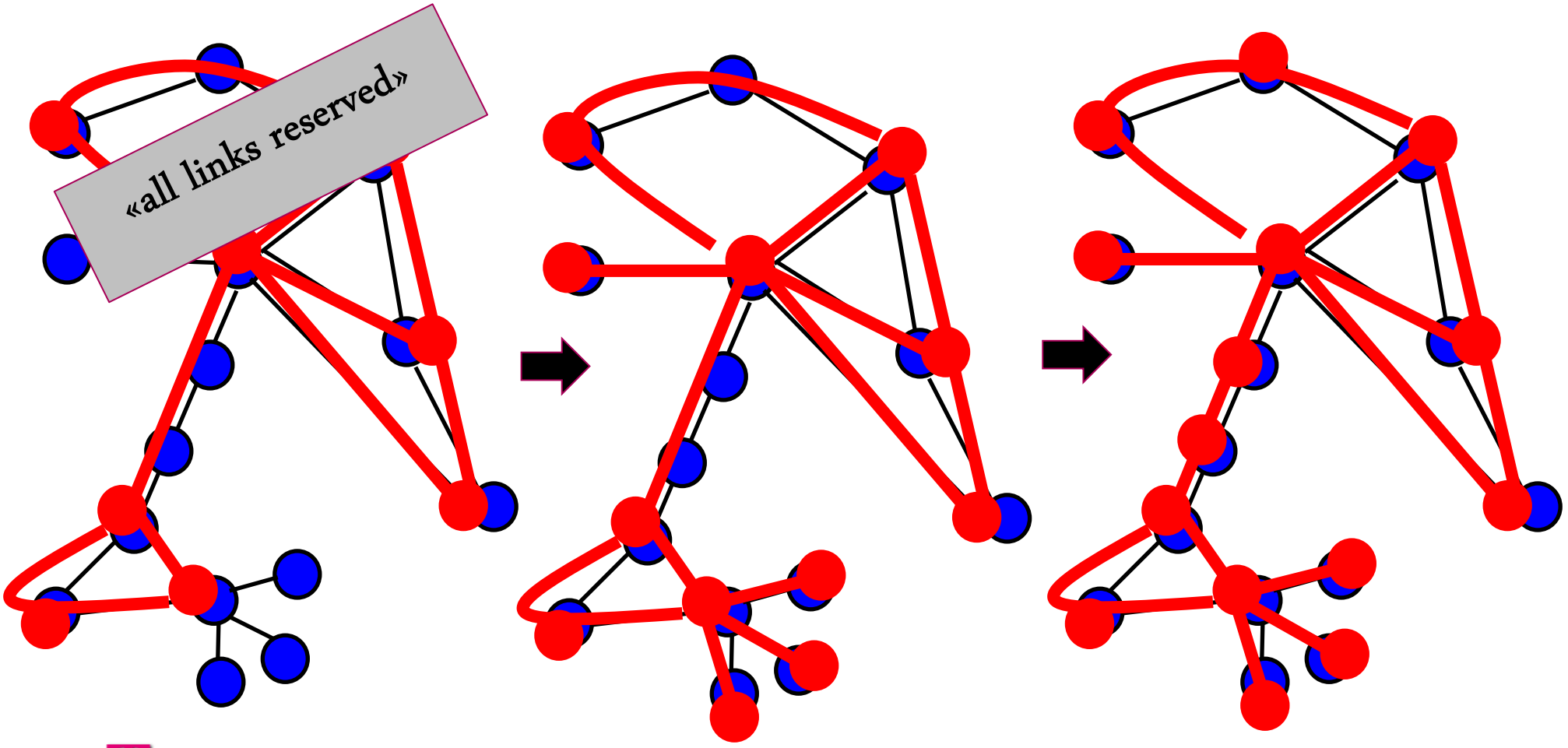
- The «two-or-more» connected components
- Later «expand nodes» and «expand edges»



# Greedy Graph Growing on General Graphs? (2)

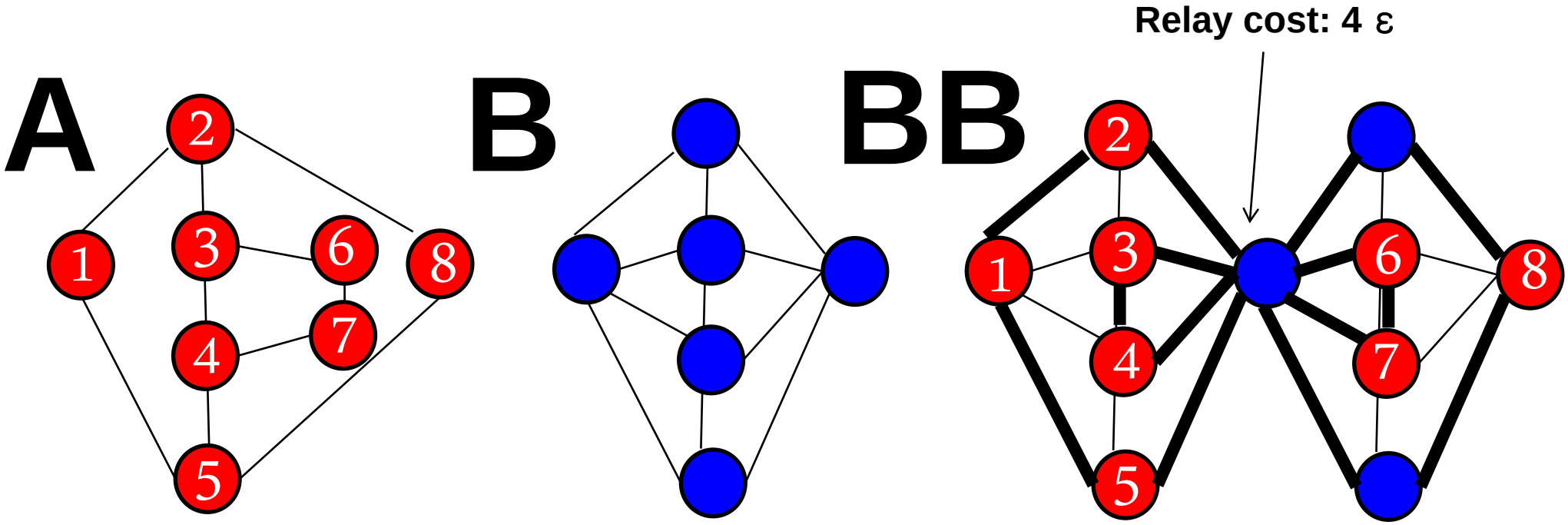
Simple solution: First try to find the «knitting»!

- The «two-or-more» connected components
- Later «expand nodes» and «expand edges»



# Greedy Graph Growing on General Graphs? (3)

**Idea: Ask graph «motif» only if it's guaranteed that it cannot be embedded over a more highly connected subgraph! (And connectivity has to be added later.)**




Careful: What goes first also depends on entire motif sequences!

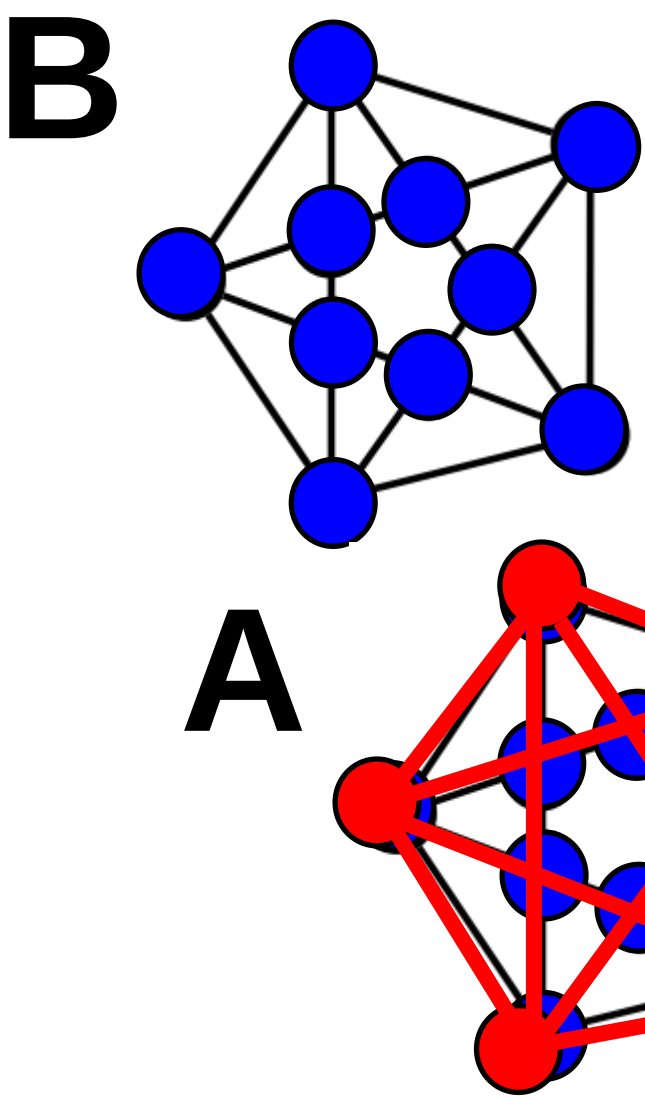
# Remark.

## Minor vs embedding:

Even with unit link capacity, for small epsilon, graph A may be embeddable (→) into graph B although A is not a minor of B!

### Graph Minor

Graph A is a minor of B if A can be obtained from B by (1) deleting nodes, (2) deleting edges, or (3) contracting two nodes along edges.




Planar graph (and hence K5-minor free):
But K5 can be embedded here!

# Dictionary Attack: Expansion Framework.

## Motif

Basic "knittings" of the graph.

## Dictionary

Define an order on motif sequences: Constraints on which sequence to ask first in order not to overlook a part of the topology. (E.g., by embedding links across multiple hops.)

## Poset

Poset = partially ordered set
(1) Reflexive: G $\rightarrow$ G
(2) Transitive: G $\rightarrow$ G' and G' $\rightarrow$ G'', then G $\rightarrow$ G''
(3) Antisymmetric: G $\rightarrow$ G' and G' $\rightarrow$ G implies G=G' (isomorphic)

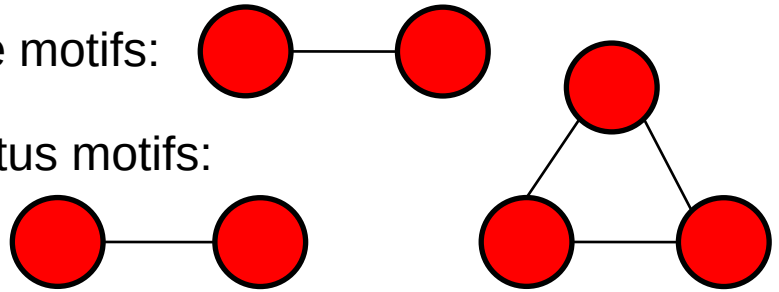
## Framework

**Explore branches according to dictionary order, exploiting poset property.**

**Examples**

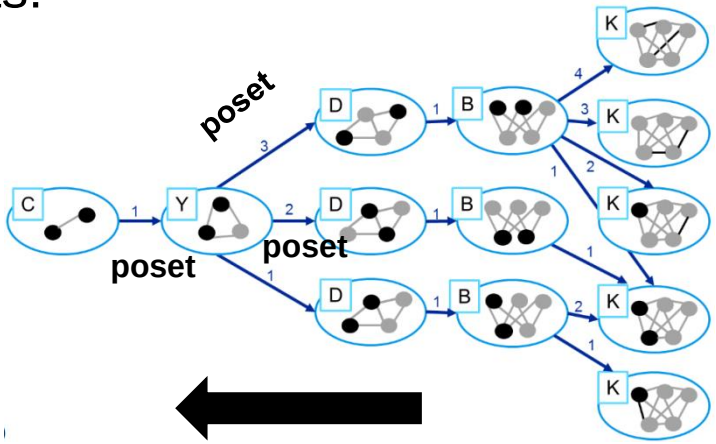
Tree motifs:

Cactus motifs:

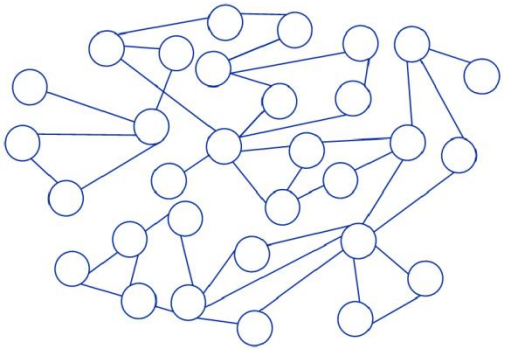


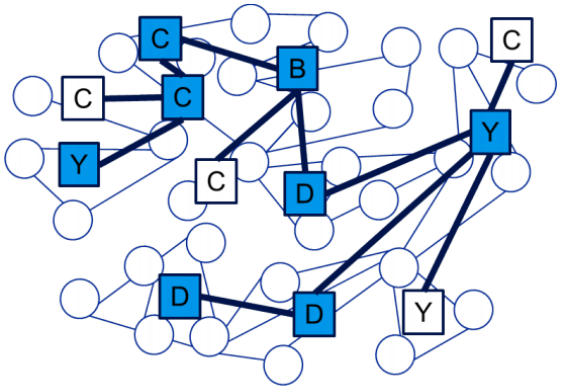
# Dictionary Attack: Expansion Framework.

**Dictionary dag** (for chain C, cycle Y, diamond D, ...) with attachment points:



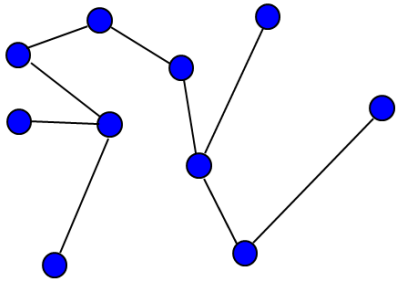

**solves, e.g.,**






**Complexity:**
Depends on dictionary depth and number of attachment points

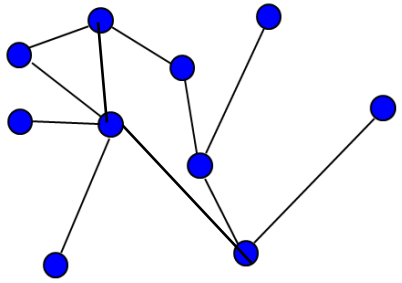
# Overview of Results.



## Tree

**Can be explored in O(n) requests. This is optimal!**

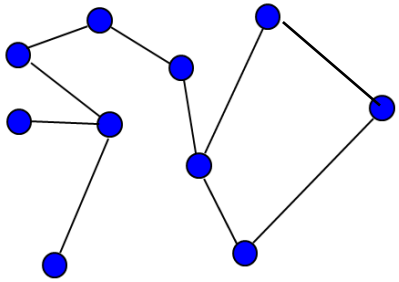
Lower bound: via number of possible trees and binary information.



## General Graph

**Can be explored in $O(n^2)$ requests. This is optimal!**

Idea: Make spanning tree and then try all edges.
(Edges directly does not work!)

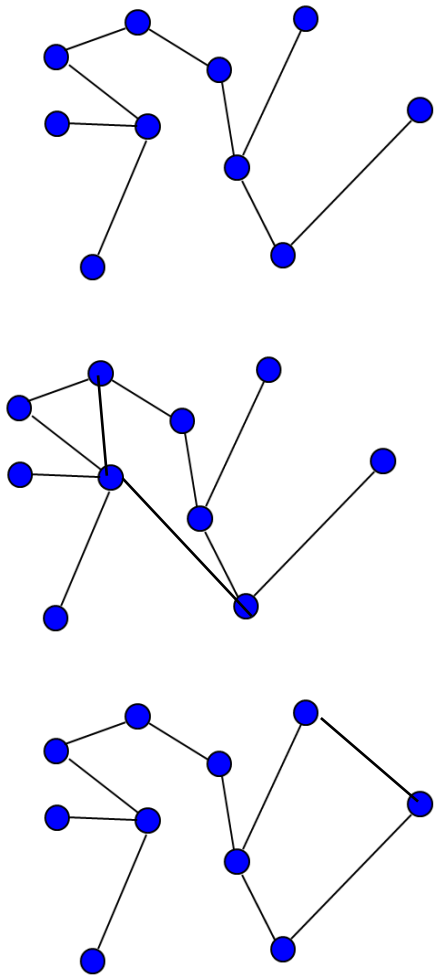


## Cactus Graph

**Can be explored in O(n) requests. This is optimal!**

Via «graph motifs»!
A general framework exploiting poset relation.

# Overview of Results.



**Algorithm 2** Cactus Discovery: CAC

```
1: G := {{v}, ∅} /* current request graph */
2: P := {v} /* pending set of unexplored nodes*/
3: while P ≠ ∅ do
4:     choose v ∈ P, S :=exploreSequence(v)
5:     if S ≠ ∅ then
6:         G := GvS, add all nodes of S to P
7:         for all e ∈ S do edgeExpansion(e)
8:     else
9:         remove v from P
```

*exploreSequence*$(v)$

```
1: S := ∅
2: if GvYCY ↦ H then
3:     find max j s.t. GvY^jCY ↦ H
4:     S := Y^jCY, P' := {C}
5:     while P' ≠ ∅ do
6:         for all C_i ∈ P' do
7:             A := prefix(C_i, S), B := postfix(C_i, S);
8:             if GvACYCB ↦ H then
9:                 find max j,k s.t. GvAC(Y^jC)^kB ↦ H
10:                for l := 1,...,k do
11:                    P'' := P'' ∪ {C_l}
12:                S := AC(Y^jC)^kB
13:        P' := P'', P'' := ∅
14: if request(GvSY, H) then
15:     find max j s.t. GvSY^j ↦ H
16:     S := SY^j
17: if request(GvSC, H) then
18:     S := SC
19: return S
```

*edgeExpansion*$(e)$

```
1: let u, v be the endpoints of edge e, remove e from G
2: find max j s.t. GvC^ju ↦ H
3: G := GvC^ju, add newly discovered nodes to P
```

ower bound: via number of
ossible trees and binary
nformation.

Idea: Make spanning tree and then try all edges.
(Edges directly does not work!)

Via «graph motifs»!
A general framework exploiting poset relation.

# Dictionary Attacks: Expand Framework.

## Motif

Basic "knitti

## Poset

Partially orde
relation fulfills
symmetry, tra

## Framev

**Explore bra
to dictiona
exploiting**

**Algorithm 5** Motif Graph Discovery DICT

1: $H' := \{\{v\}, \emptyset\}$ /* current request graph */
2: $\mathcal{P} := \{v\}$ /* pending set of unexplored nodes*/
3: **while** $\mathcal{P} \neq \emptyset$ **do**
4: choose $v \in \mathcal{P}, T := find\_motif\_sequence(v, \emptyset, \emptyset)$
5: **if** $T \neq \emptyset$ **then**
6: $H' := H'vT$, add all nodes of $T$ to $\mathcal{P}$
7: **for all** $e \in T$ **do** $edgeExpansion(e)$
8: **else**
9: remove $v$ from $\mathcal{P}$

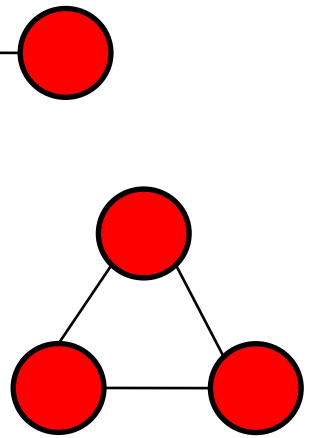
*find_motif_sequence*$(v, T^<, T^>)$

1: find max $i, j$, BF, AF s.t.
$H'v\ (T^<)$ BF $(D[i])^j$ AF $(T^>) \mapsto H$ where
BF, AF $\in \{\emptyset, C\}^2$ /*issue requests*/
2: **if** $(i, j, \text{BF}, \text{AF}) = (0, 0, C, \emptyset)$ **then**
3: **return** $T^<CT^>$
4: **if** BF $= C$ **then**
5: BF $= find\_motif\_sequence(v, T^<, (D[i])^j \text{ AF } T^>)$
6: **if** AF $= C$ **then**
7: AF $= find\_motif\_sequence(v, T^< \text{ BF } (D[i])^j, T^>)$
8: **return** BF $(D[i])^j$ AF

*edge_expansion*$(e)$

1: let $u, v$ be the endpoints of edge $e$, remove $e$ from $H'$
2: find max $j$ s.t. $H'vC^ju \mapsto H$ /*issue requests*/
3: $H' := H'vC^ju$, add newly discovered nodes to $\mathcal{P}$

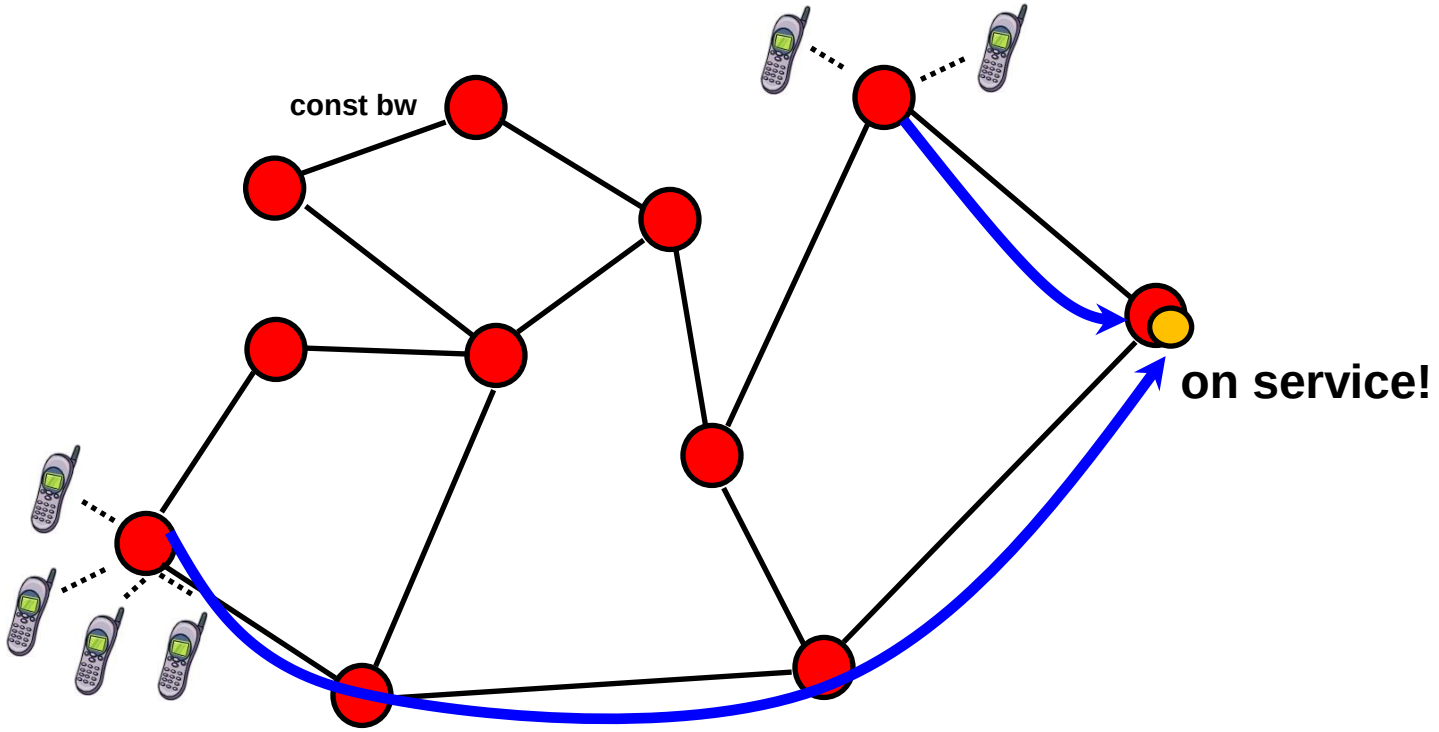
equences:
ence to ask first
part of the
ling links across



# Talk Outline:

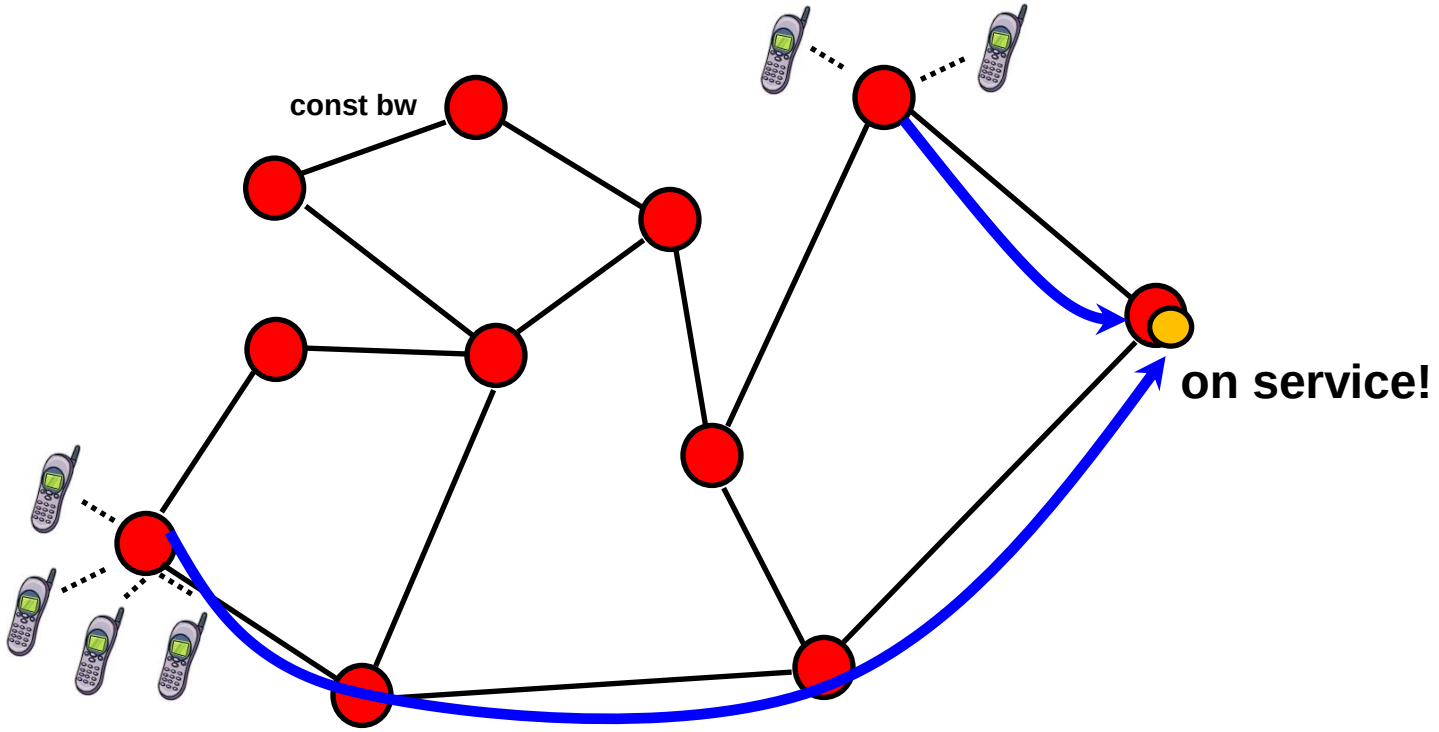
1. Which VNets to accept? (And how to embed?)
2. Threats: VNet embeddings with bad intentions?
3. **Migrating VNets**
4. VNets with in-network processing

# The Virtual Service Migration Problem.




**Given a virtual network with guaranteed bandwidth: where to migrate service?**

# The Virtual Service Migration Problem.

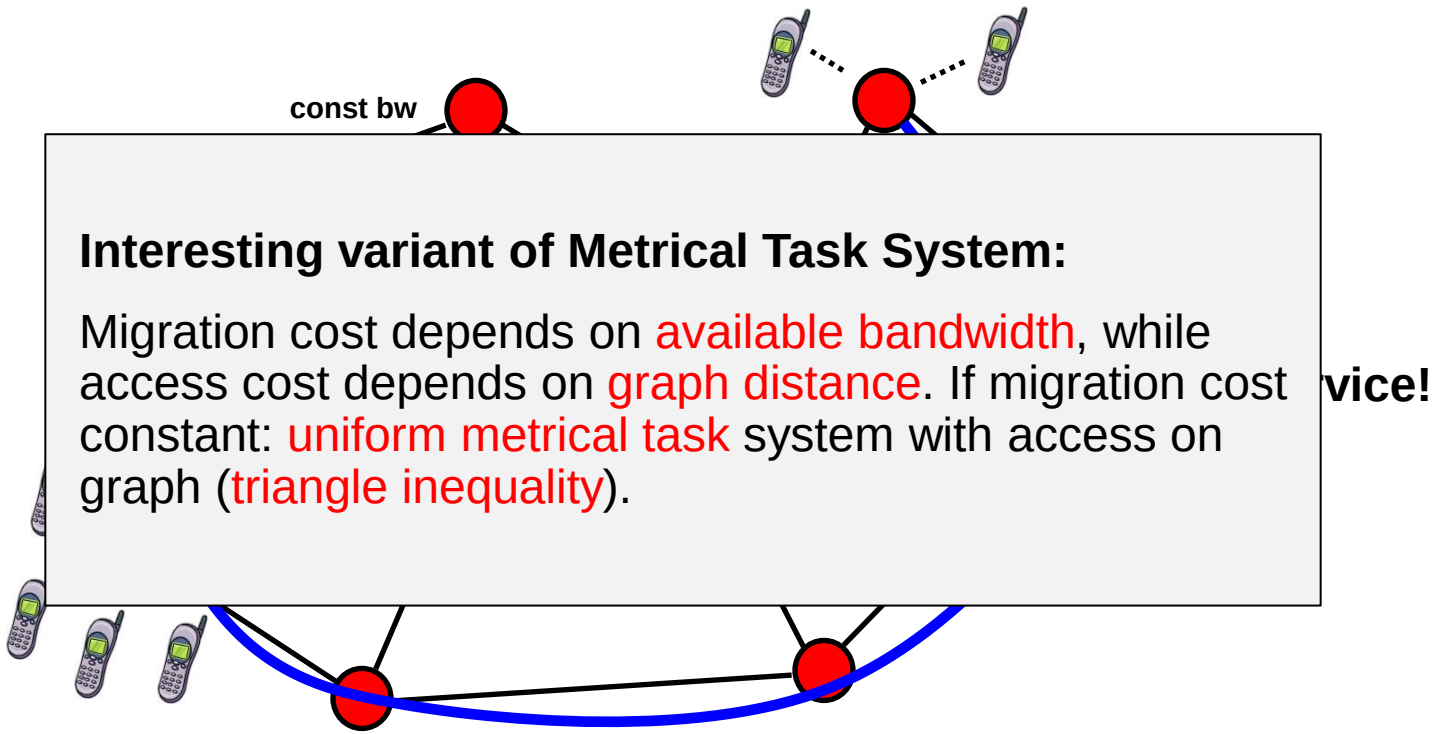



**Given a virtual network with guaranteed bandwidth: where to migrate service?**

Simple model: one service, constant migration cost (interruption), access along graph.

Cost: m * # migrations + sum of access latency.

# The Virtual Service Migration Problem.




**Given a virtual network with guaranteed bandwidth: where to migrate service?**

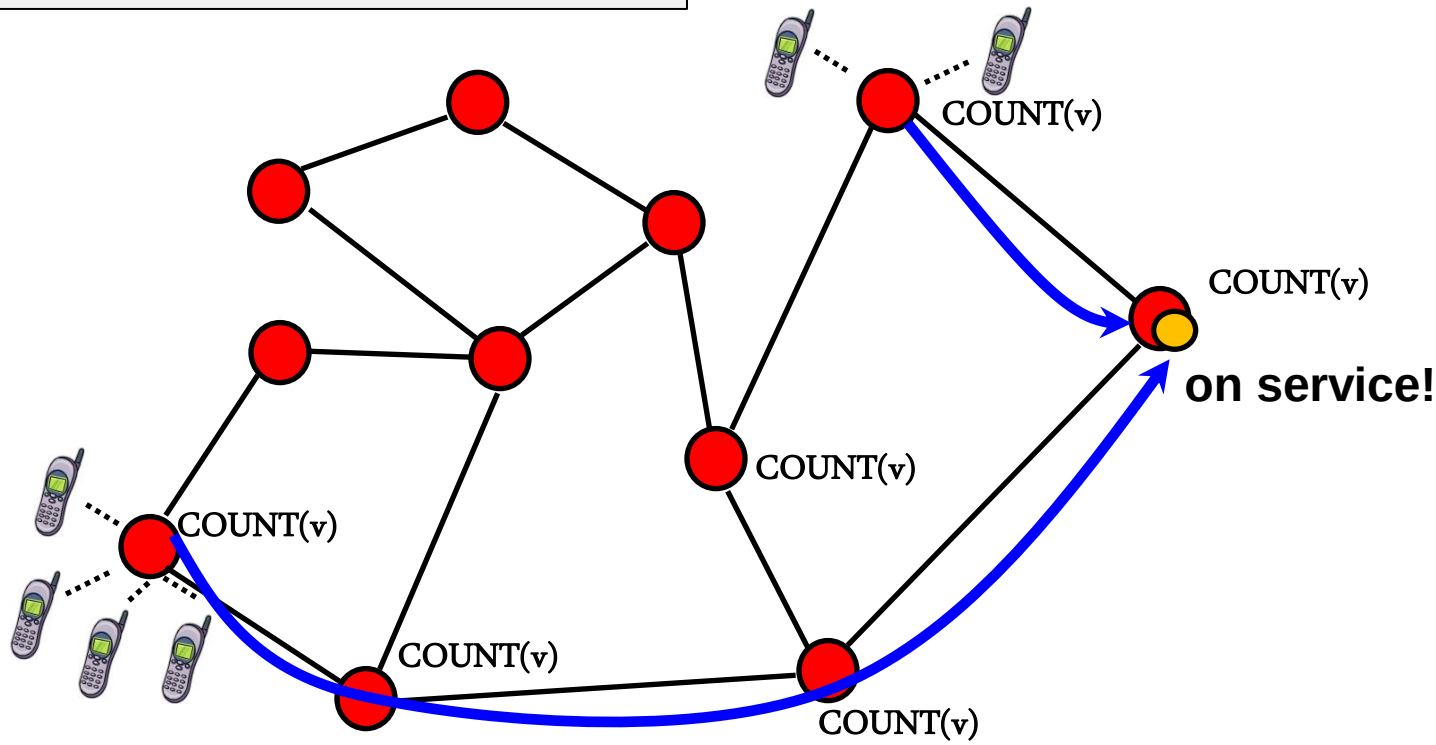
Simple model: one service, constant migration cost (interruption), access along graph.

Cost: m * # migrations + sum of access latency.

# Algorithms?

**Randomized Algo:**

1. Access cost counters at each node (if service there)
2. When counter exceeds m, migrate to random node with counter lower than m.
3. When no node left, epoch ends. Reset and restart.



**Randomized Algo:**

1. Access cost counters at each node (if service there)
2. When counter exceeds m, migrate to random node with counter lower than m.
3. When no node left, epoch ends. Reset and restart.

## Analysis: log(n)-competitive

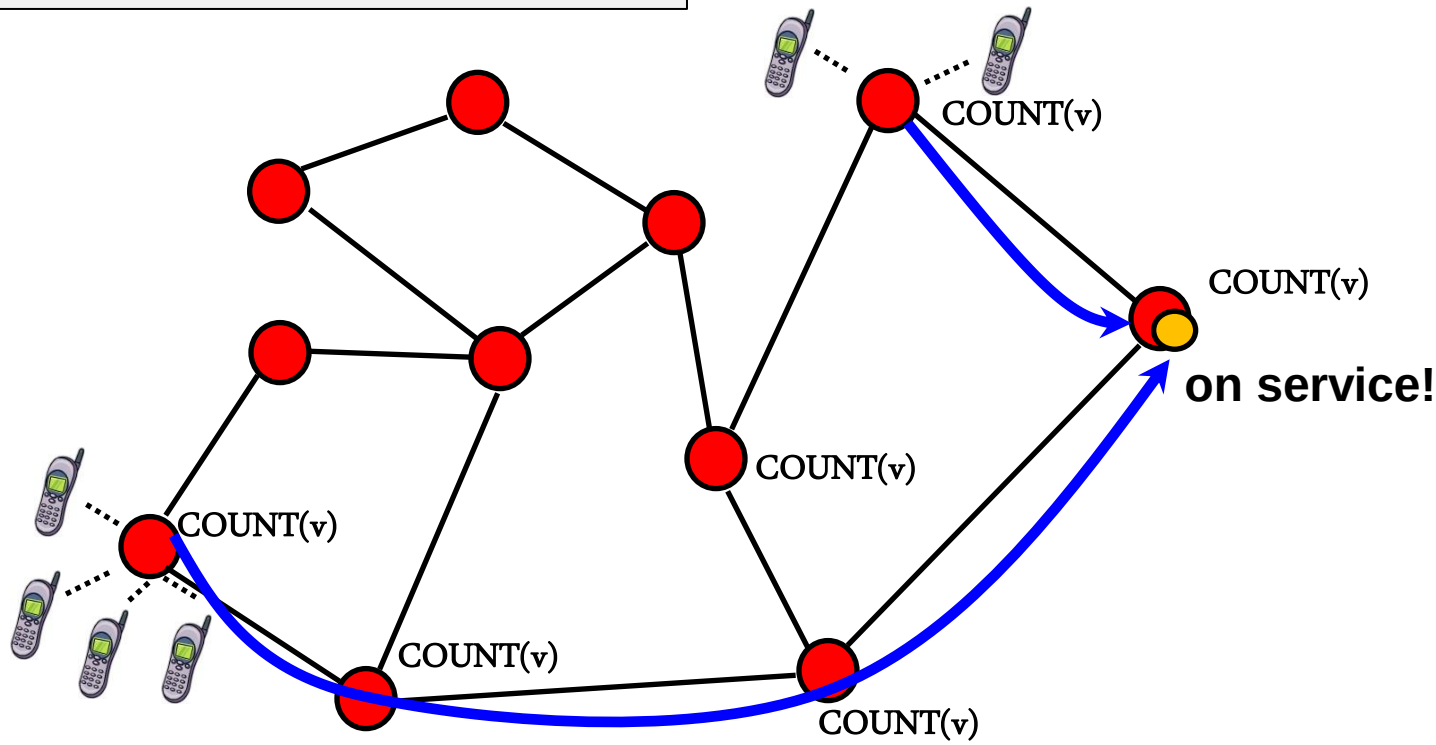
Offline cost per epoch:

at least m (migrate or access cost)

Online cost per epoch:

Per phase at most 2m (access plus migration).

At most log(n) phases: go to random node in remaining order.



**Deterministic Algo:**

1. Access cost counters at each node (if service there)

2. When counter exceeds m, deactivate nodes with counters > m/40, migrate to active node in center of active component.

3. When no node left, epoch ends. Reset and restart.

## Analysis: log(n)-competitive

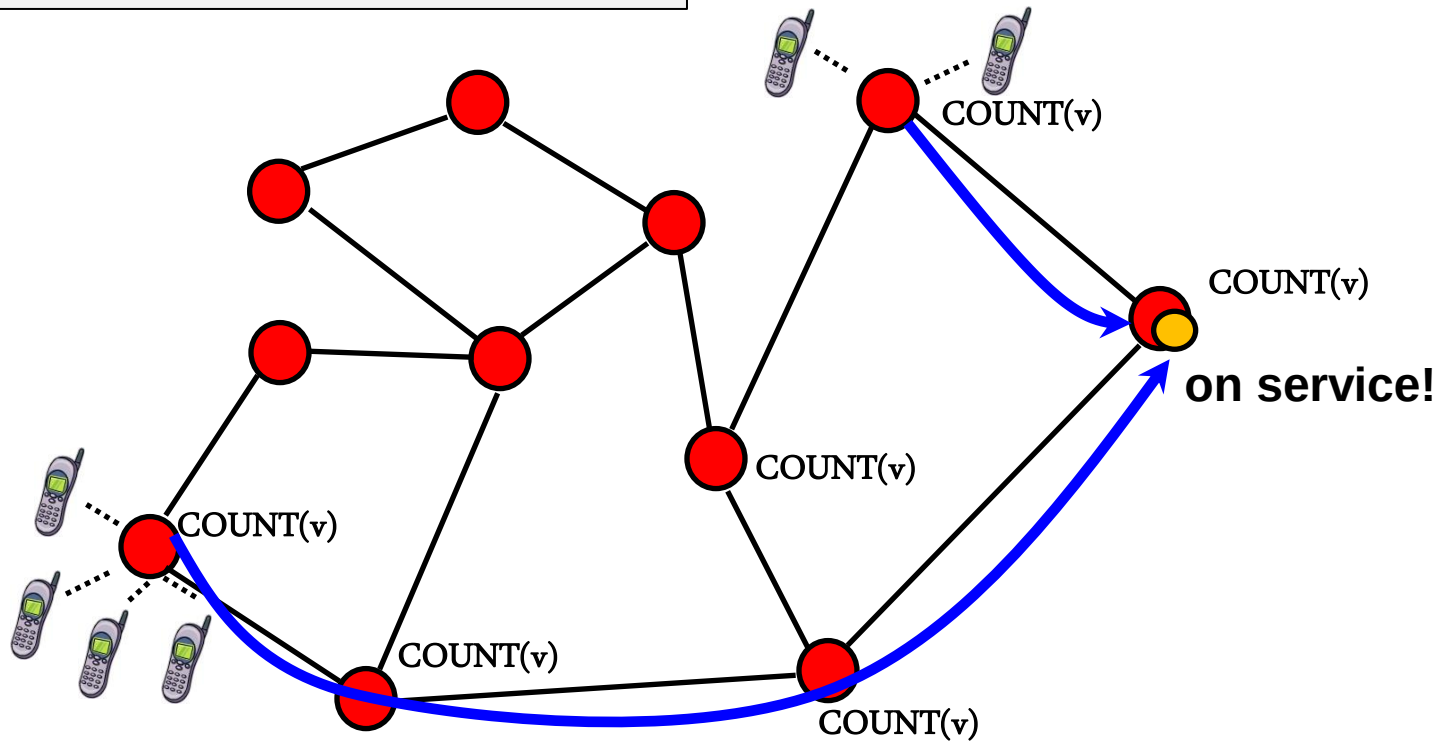
Offline cost per epoch:

at least m (migrate or access cost)

Online cost per epoch:

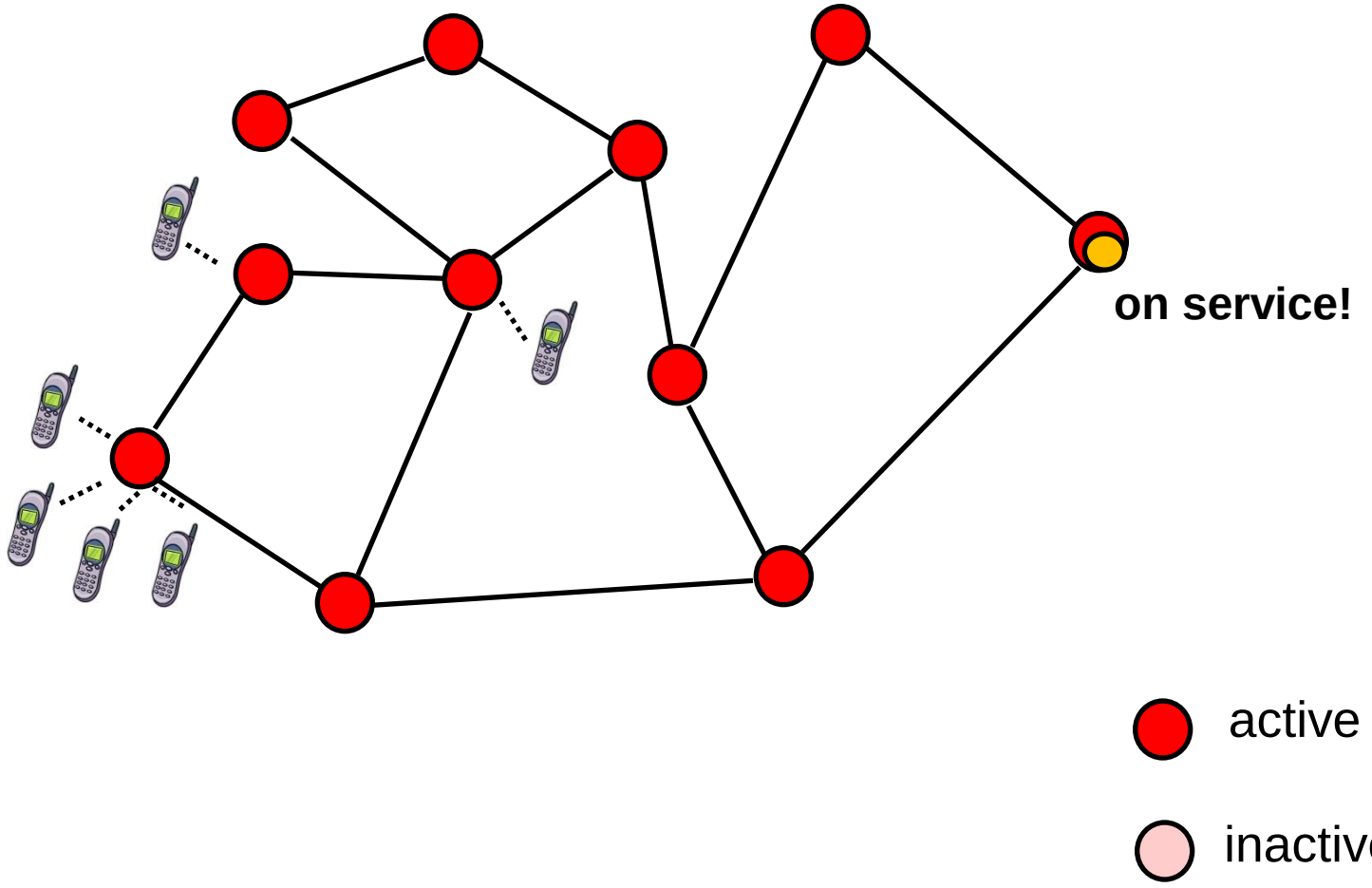
Per phase at most 2m (access plus migration).

At most log(n) phases: exploit triangle inequality.



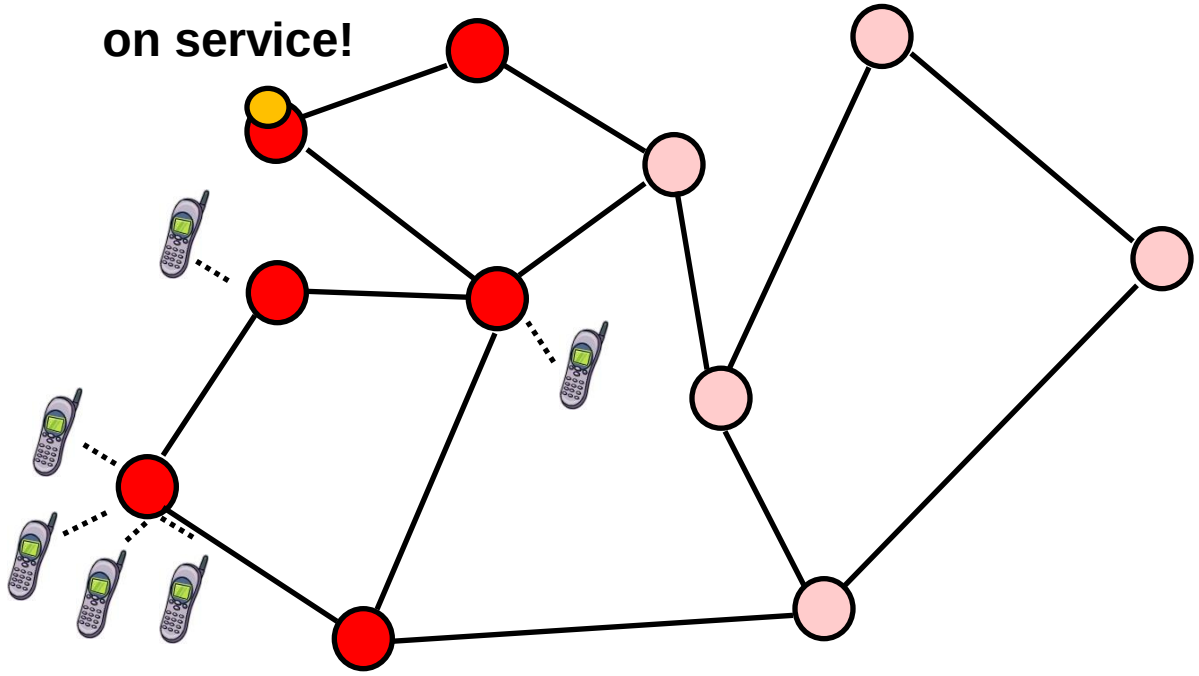
# Center-of-Gravity Algo: Example.

Before phase 1:



# Center-of-Gravity Algo: Example.

Before phase 2:

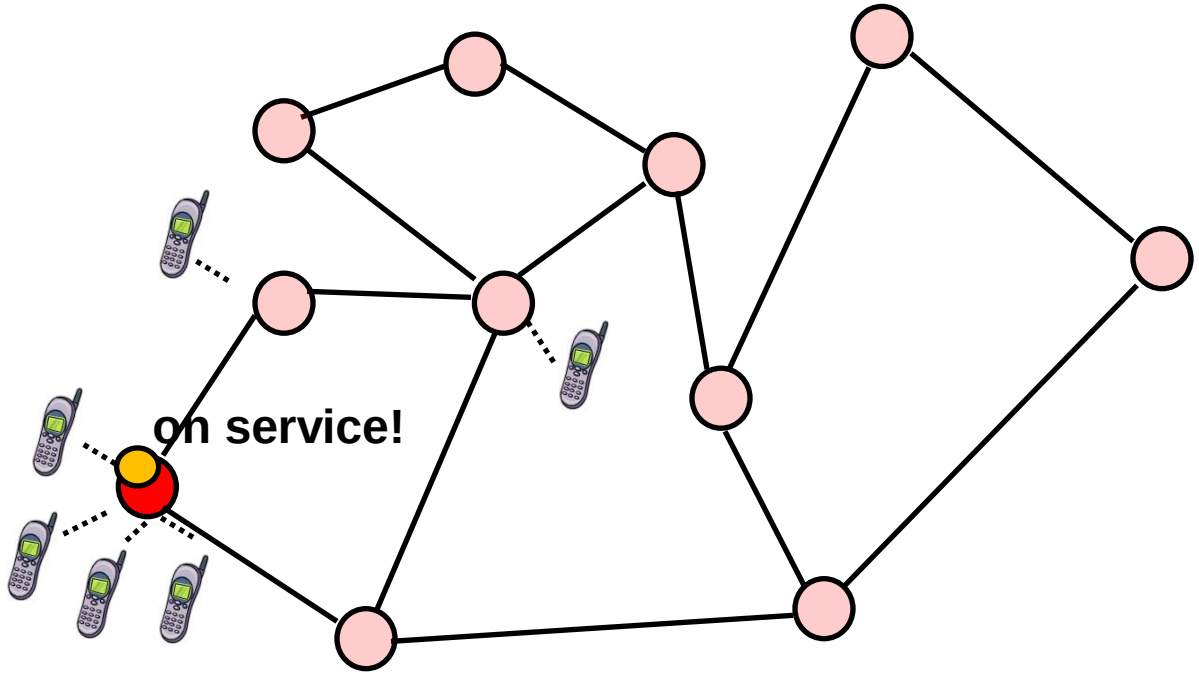



active

inactive

# Center-of-Gravity Algo: Example.

End of epoch:




active

inactive

# Center-of-Gravity Algo: Result.

Competitive analysis? Assume constant bandwidths!

**r = ALG / OPT ?**

Lower bound cost of OPT:

In an **epoch**, each node has at least access cost ***m***, or there was a migration of cost ***m***.

Upper bound cost of ALG:

We can show that each **phase** has cost at most ***2m*** (access plus migration), and there are at most ***log(n)*** many phases per **epoch**!

**Theorem**

**ALG is log(n) competitive!**

**A special uniform metrical task system (graph metric for access)!**

# Optimality?

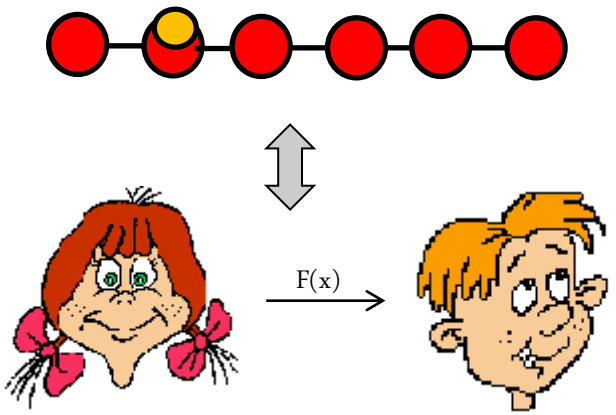
**Theorem**

**«Center of Gravity» algorithm is log(n) competitive!**

Also a much simpler randomized algorithm achieves this!

**log(n)/loglog(n) lower bound follows from online function tracking reduction!**

**on service!**




**Online function tracking** *with linear penalties*: Alice observes values $x_t$ and Bob has representative value F(x). Upon new value, either Alice transmits (migration cost) or pays difference $|x_t-y|$ (access cost).

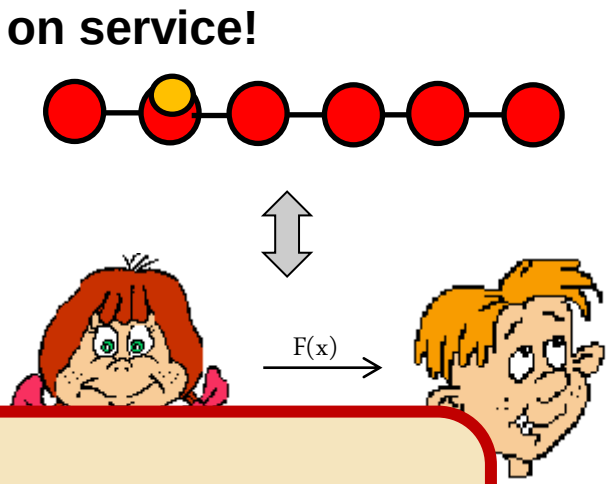
# Optimality?

**Theorem**

**«Center of Gravity» algorithm is log(n) competitive!**

Also a much simpler randomized algorithm achieves this!

**log(n)/loglog(n) lower bound follows from online function tracking reduction!**




**Can be achieved with a refined analysis!**

# The Online Algorithm FOLLOWER.

Concepts:

- **Learn from the past:** migrate to center of gravity of best location *in the past*
- **Amortize:** migrate only when access cost at current node is as high as migration cost!

**Algorithm *Follower***

1: $i := 0$; $k_0 := 0$ $\forall j$: $F_j = \{\}$ {The server starts at an arbitrary node $f_0$}
Upon a new request $r$ do:
2: Serve request $r$ with server at $f_i$
3: $F_i := F_i \cup r$
4: $f' :=$ arbitrary $u \in CG(F_i)$
5: $x' := d(f_i, f')$ {for `co.di.`, and $x' := 1$ for `co.nb.m.`}
6: **if** $C(f_i, F_i) \geq g(x'|k_i)$ **then**
7: $f_{i+1} := f'$; $x_i := x'$
8: $y(w) := d(f_i, w) + d(w, f_{i+1})$ {for `co.di.`, and for `co.nb.m.` $y(w) := 2$ for $w \neq f_{i+1}$ and $y(w) := 1$ otherwise }
9: $slack(w \in V) := g(y(w)|k_i) - C(f_i, F_i)$
10: $w_i :=$ Node $w$ with minimum $slack(w)$ such that $slack(w) \geq 0$
11: Move server to $w_i$ and if $w_i \neq f_{i+1}$ onto $f_{i+1}$
12: $k_{i+1} := k_i + y(w_i)$
13: $i := i + 1$
14: **end if**

## Simplified Follower

1. $F_i$ are requests handled while service at $f_i$
2. to compute $f_{i+1}$ (new pos), Follower only takes into account requests during $f_i$: $F_i$
3. migrate to center of gravity of $F_i$, as soon as migration costs there are amortized (and «reset counters» immediately)!

# The Online Algorithm FOLLOWER.

Concepts:

- **Learn from the past:** migrate to center of gravity of best location *in the past*
- **Amortize:** migrate only when access cost at current node is as high as migration cost!

**Algorithm *Follower***

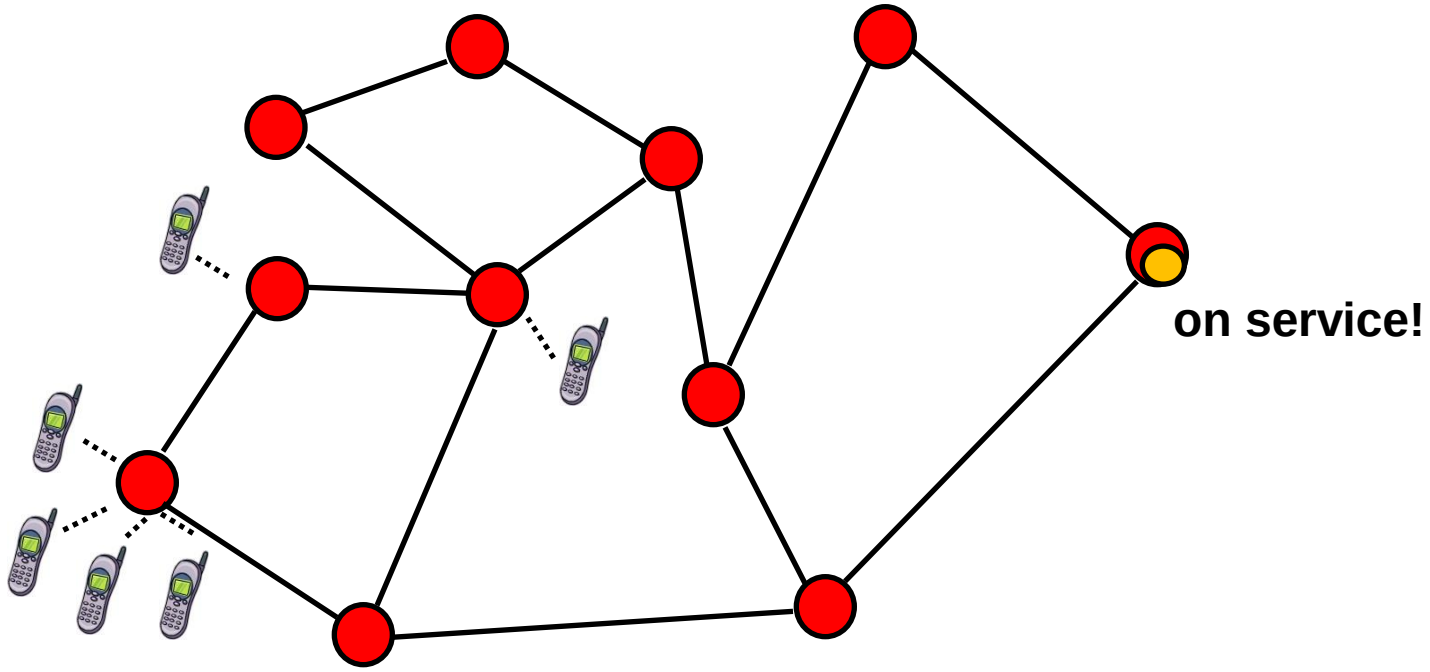
1: $i := 0$; $k_0 := 0$ $\forall j$: $F_j = \{\}$ {The server starts at an arbitrary node $f_0$}
Upon a new request $r$ do:
2: Serve request $r$ with server at $f_i$
3: $F_i := F_i \cup r$
4: $f' :=$ arbitrary $u \in CG(F_i)$
5: $x' := d(f_i, f')$ {for `co.di.`, and $x' := 1$ for `co.nb.m.`}
6: **if** $C(f_i, F_i) \geq g(x'|k_i)$ **then**
7: $f_{i+1} := f'$; $x_i := x'$
8: $y(w) := d(f_i, w) + d(w, f_{i+1})$ {for `co.di.`, and for `co.nb.m.` $y(w) := 2$ for $w \neq f_{i+1}$ and $y(w) := 1$ otherwise }
9: $slack(w \in V) := g(y(w)|k_i) - C(f_i, F_i)$
10: $w_i :=$ Node $w$ with minimum $slack(w)$ such that $slack(w) \geq 0$
11: Move server to $w_i$ and if $w_i \neq f_{i+1}$ onto $f_{i+1}$
12: $k_{i+1} := k_i + y(w_i)$
13: $i := i + 1$
14: **end if**

## Simplified Follower

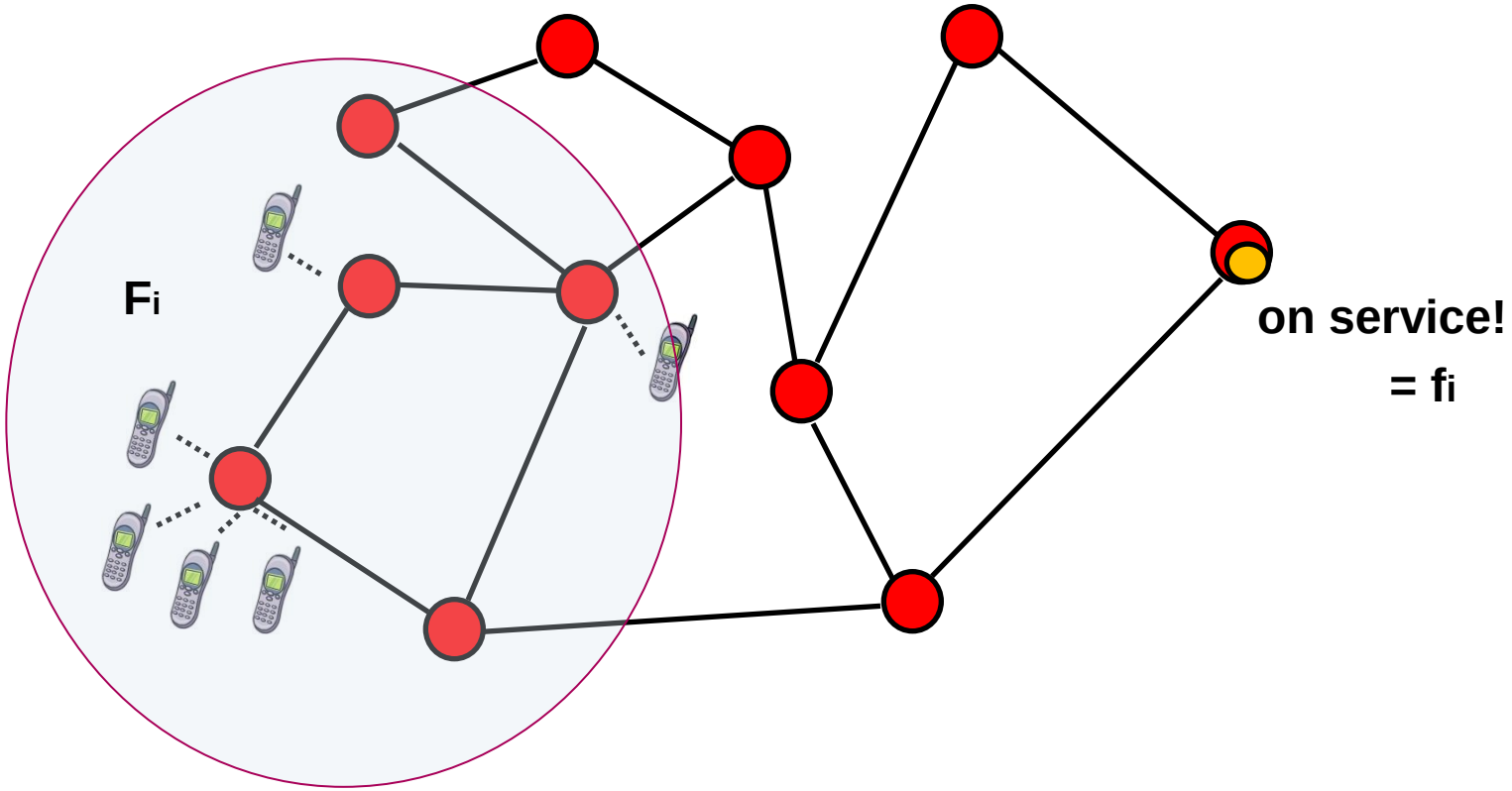
1. $F_i$ are requests handled while service at $f_i$
2. to compute $f_{i+1}$ (new pos), Follower only takes into account requests during $f_i$: $F_i$
3. migrate to center of gravity of $F_i$, as soon as migration costs there are amortized (and «reset counters» immediately)!

Also works for migrations with discount! (Reseller/broker gives discount!)

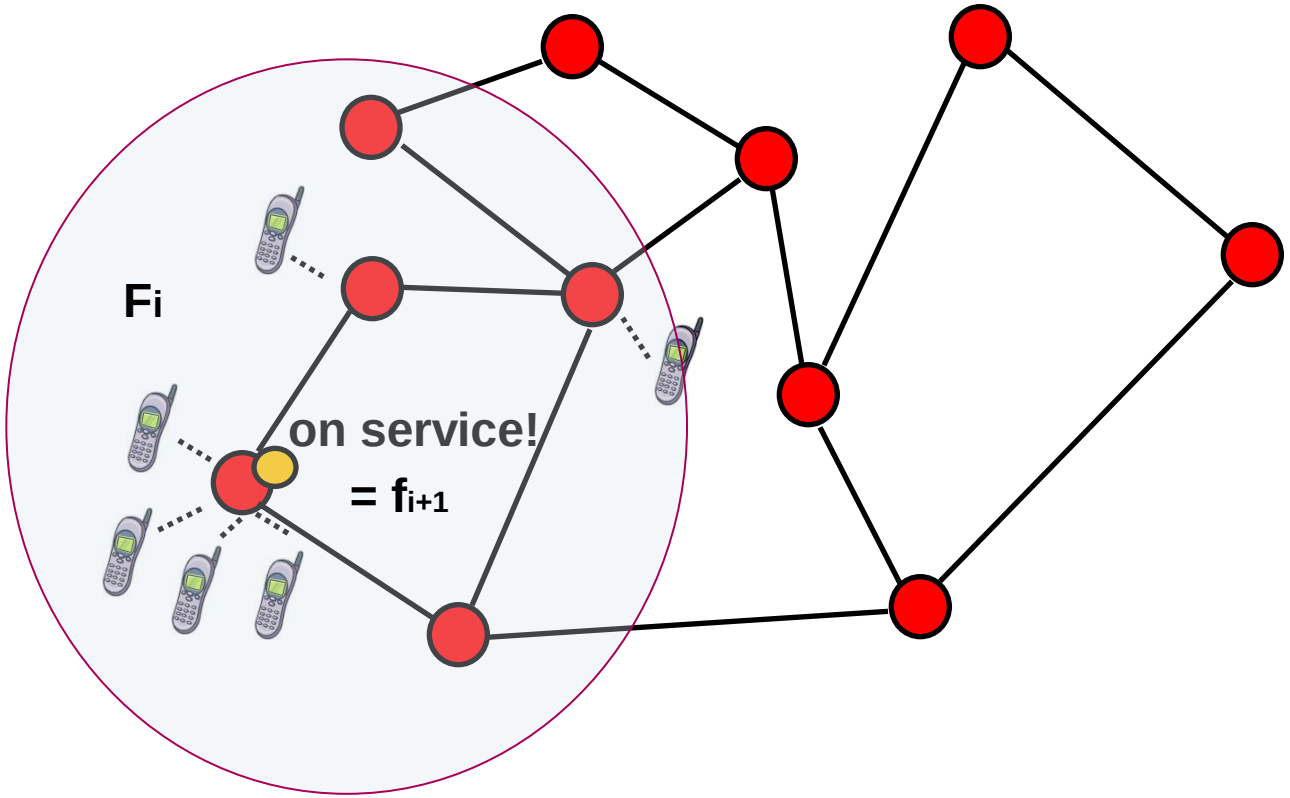
# Intuition.



# Intuition.



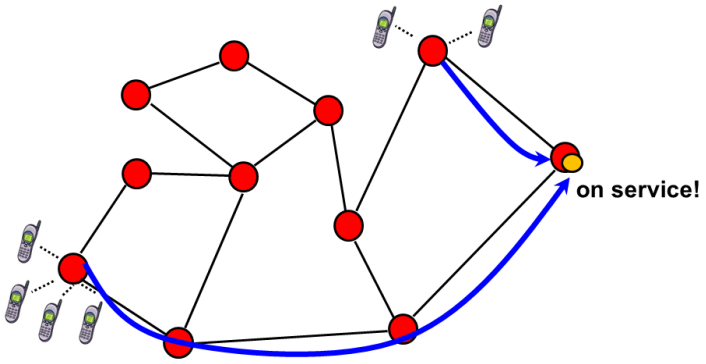
# Intuition.



# Modeling Access and Migration Costs.

## Access Costs

Latency along shortest path in graph.
(Graph distances, and in particular: metric!)




## Migration Costs

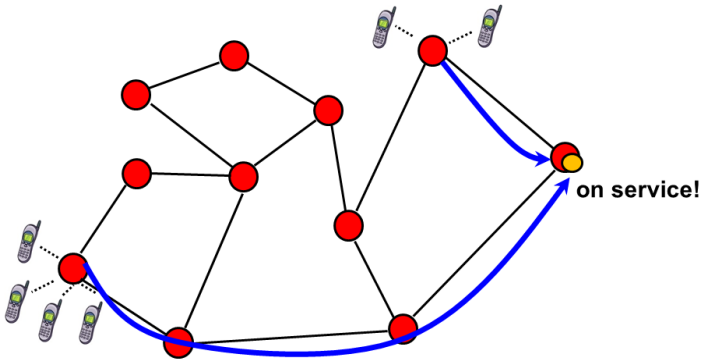
Generalized models:

- E.g., depends on bandwidth along path (duration of service interruption)
- E.g., depends on distance travelled (latency)
- Discount: e.g., VNP (number of migrations, distance travelled, ...)

# Modeling Access and Migration Costs.

## Access Costs

Latency along shortest path in graph.
(Graph distances, and in particular: metric!)




## Migration Co

Generalized mo

- E.g.,

elled, ...)

General cost function g(x|y): cost of migrating distance x given already travelled y

Or g(1|y): cost of migration given we already migrated y times

# Competitive Ratio of FOLLOWER.

Competitive analysis? **FOLLOWER / OPT?**

**Theorem**

**If no discounts are given,**
**Follower is log(n)/loglog(n) competitive!**

**Simple model with *migration costs* = *bandwidth*, and *homogeneous***

**Page migration model with *migration costs = distance,* but discounts**

**Theorem**

**If migration costs depend on travelled distance (page migration), competitive ratio is O(1), even with discounts.**

# Related Work.

- Metrical Task Systems:
  - Classical online problem where server at certain location («state») serves requests at certain costs; state transitions also come at certain costs («migration»)
  - Depending on migration cost function more general (we have graph access costs) and less general (we allow for migration discounts)
  - E.g., uniform space metrical task system: migration costs constant, but access costs more general than graph distances! Lower bound of log(n) vs log(n)/loglog(n) upper bound in our case.

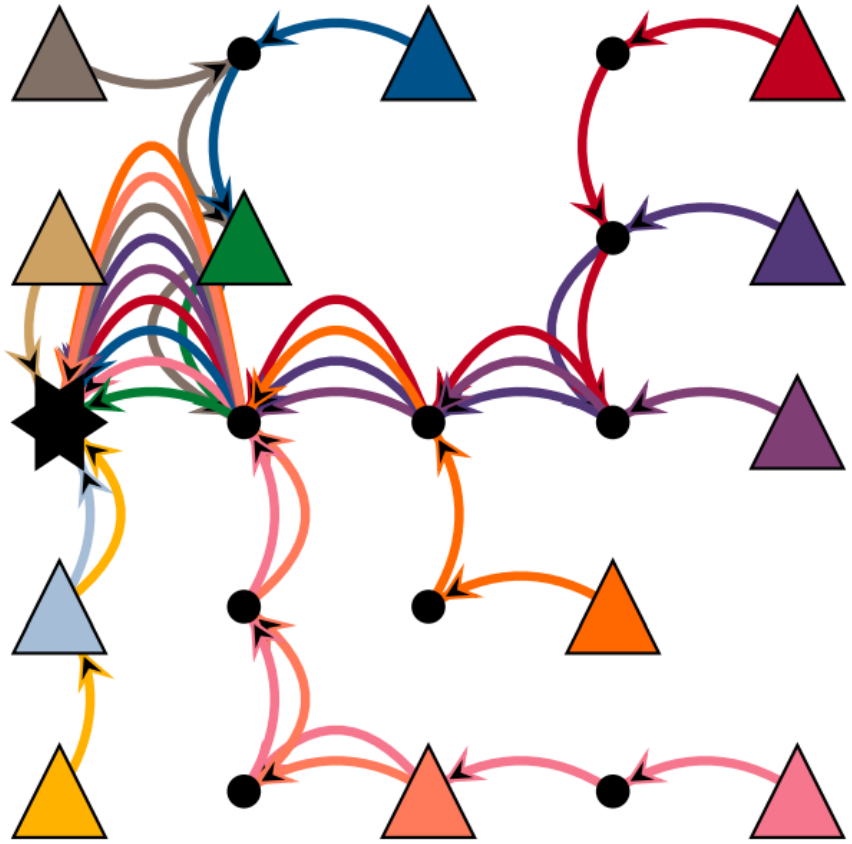
- Online Page Migration
  - Classical online problem from the 80ies; we generalize cost function to distance discounts, while keeping O(1)-competitive

**Our work lies between!**

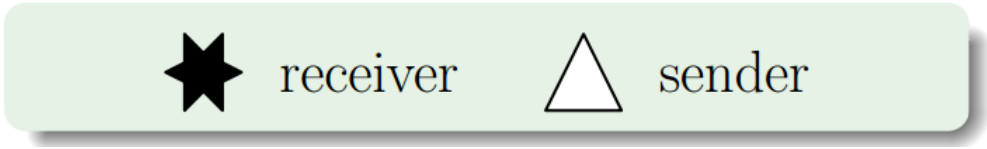
# Talk Outline:

1. Which VNets to accept? (And how to embed?)
2. Threats: VNet embeddings with bad intentions?
3. Migrating VNets
4. **VNets with in-network processing**

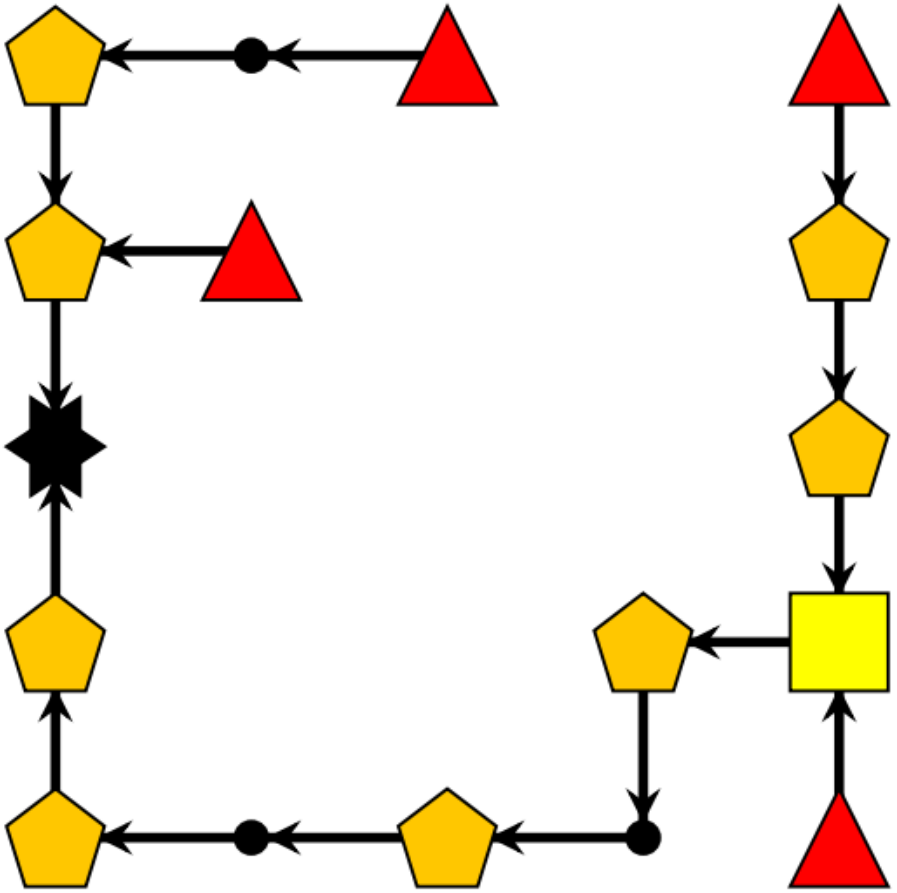
# VNet with Processing



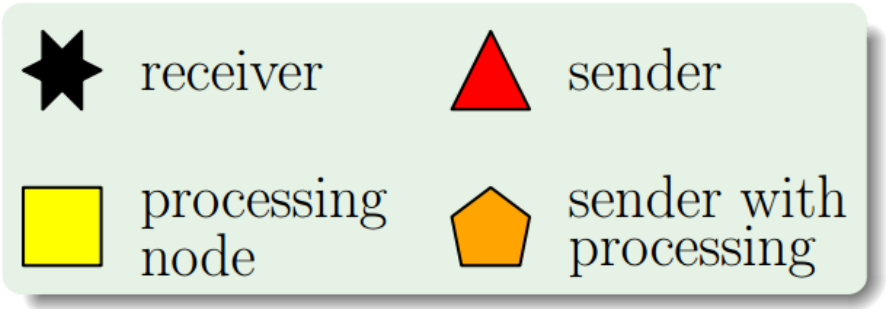
**Unicast:** one connection to each receiver (same for aggregation)!



# VNet with Processing

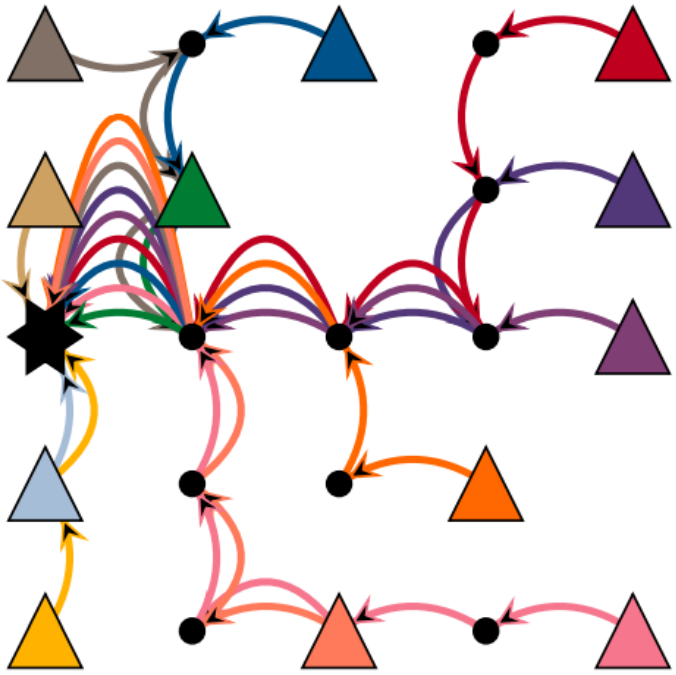


**Multicast:** processing / splitting at each node



# VirtuCast: New Tradeoff?

## Unicast



Solution Method

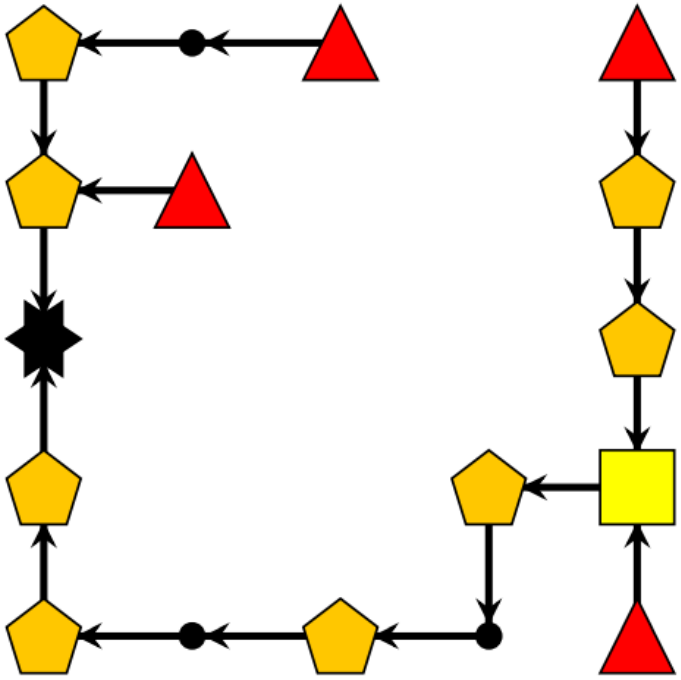
- minimal cost flow

Solution uses

- 43 edges
- 0 processing nodes

vs.

## Multicast



Solution Method

- Steiner arborescence

Solution uses

- 16 edges
- 9 processing nodes

# VirtuCast: Optimal Tradeoff

**Solution uses**

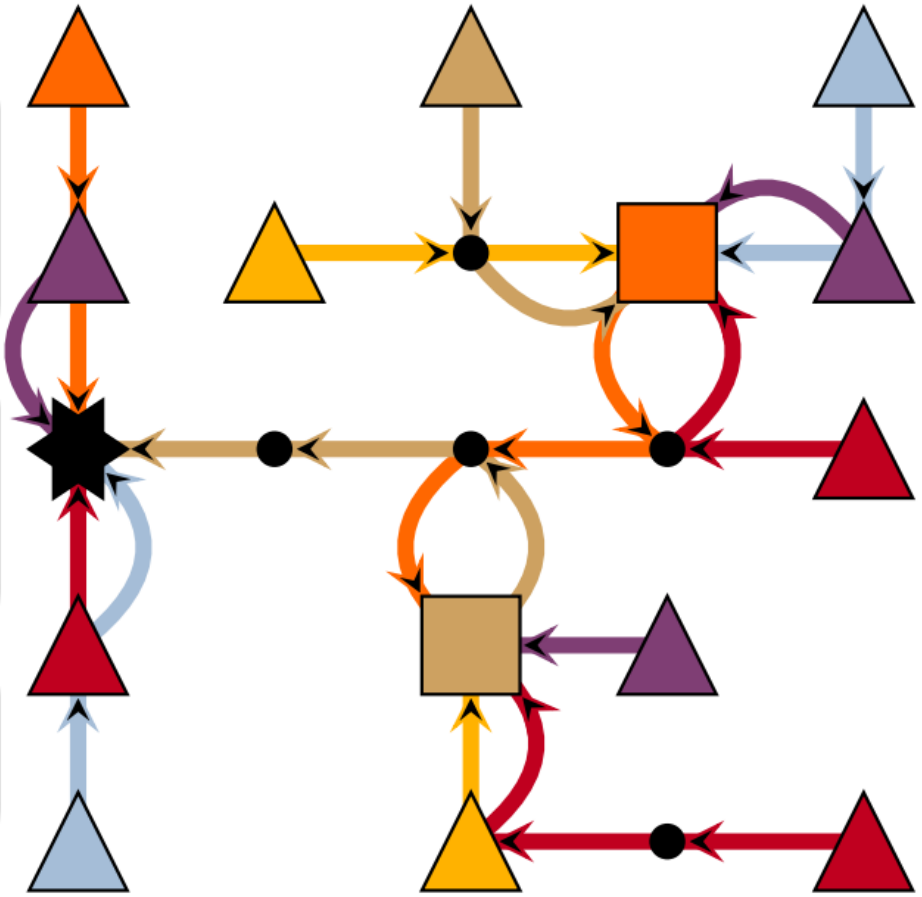
- 26 edges
- 2 processing nodes

**New Model**

Constrained Virtual Steiner Arboresence Problem (CVSAP)

**New Solution Method**

VirtuCast algorithm



# A Single-Commodity Algorithm

**Example: 6000 edges and 200 Steiner sites**

- Single-commodity: 6000 integer variables
- Multi-commodity: 1,200,000 binary variables

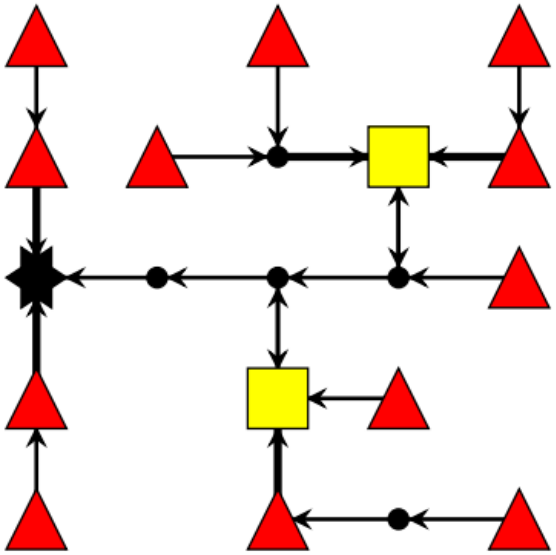


Figure: Single-commodity

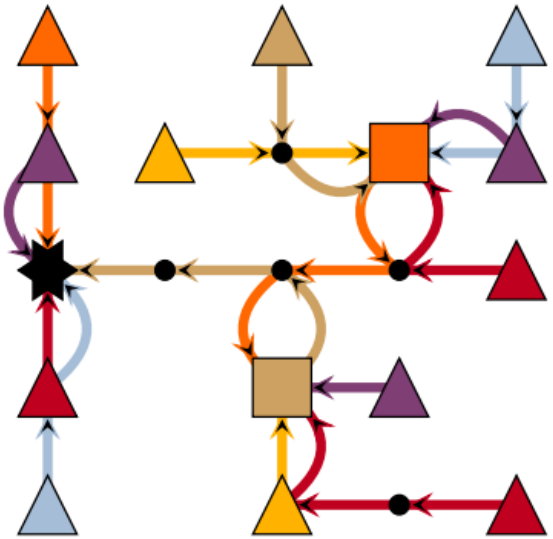


Figure: Multi-commodity

# A Single-Commodity Algorithm

**Example: 6000 edges and 200 Steiner sites**

- Single-commodity: 6000 integer variables
- Multi-commodity: 1,200,000 binary variables

**VirtuCast: 2 Stages**

1. Compute single-commodity MIP
2. Make flow decomposition: find path

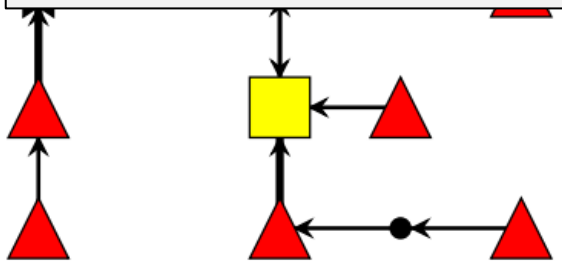


Figure: Single-commodity

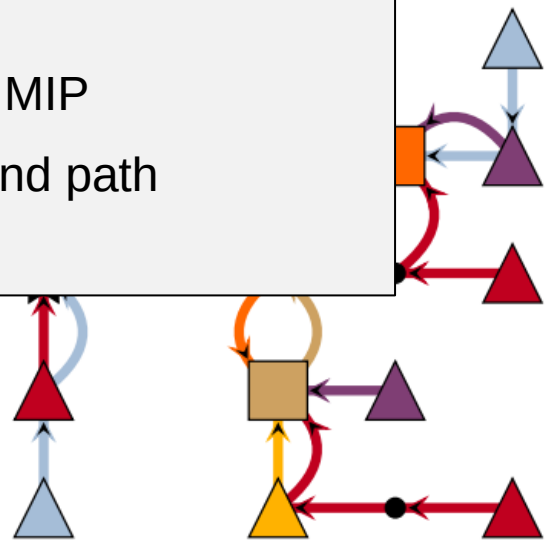
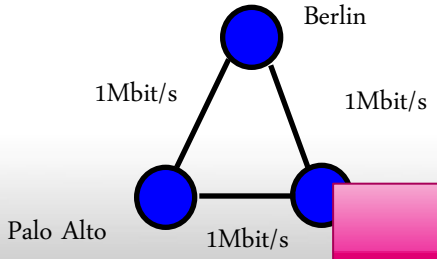


Figure: Multi-commodity

# VNets.
## Use Cases

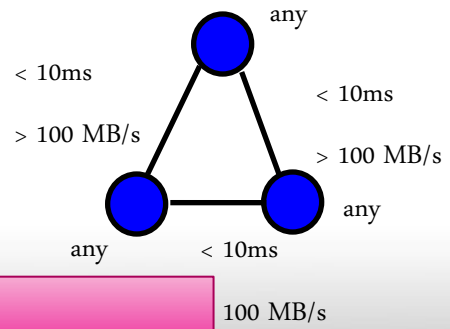
### „VPN++“

Goal: *Fully specified CloudNet mapping constraints (e.g., end-points for a **telco**), but with **QoS guarantees** (e.g., bandwidth) along links*




***„November 22, 1pm-2pm!“***

### Datacenters




***„Guaranteed resources, job deadlines met, no overhead!“***

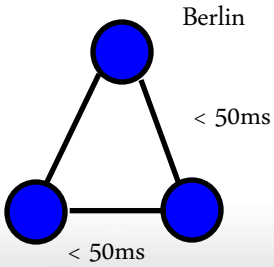
***“Network may delay execution: costly for per hour priced VM!”***

***ctopus system (SIGCOMM 2011)***

**Thank you!**

### Spillover/Out-Sourcing

***Elastic computing***



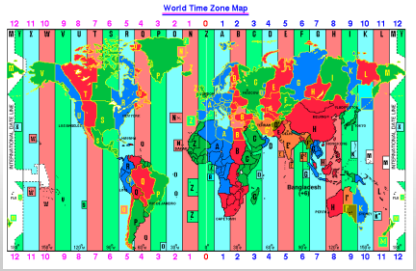

(corporate access network)

***„50 TB storage, 10 Tflops computation!“***

**„any European cloud provider (e.g. due to legal issues?)“**

### Migration / Service Deployment

Goal: *Move with the sun, with the commuters, (QoS) allow for **maintenance**, avoid roaming costs…: e.g., **SAP/game/translator server, small CDN server…***




# Backup.

# Proof Sketch (1): Simplified LP.

$$\min \sum_{e \in E} x_e \cdot c(e) + \sum_{v \in V} x_v \cdot c(v) + \sum_i z_i \cdot d_i \quad s.t.$$

$$\begin{aligned} \text{(Covering Const.)} \forall i \forall \Delta \in \Delta_i \; z_i + \alpha(i, \Delta) &\geq b_i \\ \forall i \forall \Delta \in \Delta_i \; x_e, x_v, z_i &\geq 0 \end{aligned}$$

(I)

$$\max \sum_i b_i \cdot \sum_{\Delta_{ij} \in \Delta_i} f_{ij} \quad s.t.$$

$$\begin{aligned} \text{(Vertex Capacity Const.)} \quad & \forall v \in V \;\; \mathit{flow}(v) \leq c(v) \\ \text{(Edge Capacity Const.)} \quad & \forall e \in E \;\; \mathit{flow}(e) \leq c(e) \\ \text{(Demand Const.)} \quad & \forall i \sum_{\Delta_{ij} \in \Delta_i} f_{ij} \leq d_i \\ & f \geq 0 \end{aligned}$$

(II)

Fig. 1: (I) The Primal linear embedding program. (II) The Dual linear embedding program.

maximize benefit!

realization of i-th request (will be integer, accept fully or not at all)

... while ensuring capacity and no more than demand!

# Proof Sketch (2): Simplified LP.

essentially, exponential load...

$$\min \sum_{e \in E} x_e \cdot c(e) + \sum_{v \in V} x_v \cdot c(v) + \sum_i z_i \cdot d_i \quad s.t.$$

$$\text{(Covering Const.)} \forall i \forall \Delta \in \Delta_i \; z_i + \alpha(i, \Delta) \geq b_i$$

$$\forall i \forall \Delta \in \Delta_i \; x_e, x_v, z_i \geq 0$$

(I)

$$\max \sum_i b_i \cdot \sum_{\Delta_{ij} \in \Delta_i} f_{ij} \quad s.t.$$

$$\text{(Vertex Capacity Const.)} \quad \forall v \in V \;\; \mathit{flow}(v) \leq c(v)$$

$$\text{(Edge Capacity Const.)} \quad \forall e \in E \;\; \mathit{flow}(e) \leq c(e)$$

$$\text{(Demand Const.)} \quad \forall i \sum_{\Delta_{ij} \in \Delta_i} f_{ij} \leq d_i$$

$$f \geq 0$$

(II)

Fig. 1: (I) The Primal linear embedding program. (II) The Dual linear embedding program.

# Proof Sketch (3): Simplified LP.

**Algorithm 1** The ISTP Algorithm.

Input: $G = (V, E)$ (possibly infinite), sequence of requests $\{r_i\}_{i=1}^{\infty}$ where $r_i \triangleq (U_i, c_i, d_i, b_i)$.
Upon arrival of request $r_i$:

1) $j \leftarrow \text{argmin}\{\alpha(i,j) : \Delta_{ij} \in \Delta_i\}$ (find a lightest realization over the terminal set $U_i$ using an oracle).
2) If $\alpha(i,j) < b_i$ then, (accept $r_i$)
   a) $f_{ij} \leftarrow d_i$.
   b) For each $e \in E(\Delta_{ij})$ do

$$x_e \leftarrow x_e \cdot 2^{d_i/c(e)} + \frac{1}{|V(\Delta_{ij})|} \cdot (2^{d_i/c(e)} - 1).$$

   c) For each $v \in V(\Delta_{ij})$ do

$$x_v \leftarrow x_v \cdot 2^{c_i/c(v)} + \frac{d_i/c_i}{|V(\Delta_{ij})|} \cdot (2^{c_i/c(v)} - 1).$$

   d) $z_i \leftarrow b_i - \alpha(i,j)$.
3) Else, (reject $r_i$)
   a) $z_i \leftarrow 0$.

oracle (triangle only)

update primal variables if accepted

# Proof Sketch (4): Simplified LP.

Step (2b) increases the cost $\sum_e x_e \cdot c(e)$ as follows (change $\Delta(x_e) = \sum_e (x_e^t - x_e^{t-1}) \cdot c(e)$):

$$
\begin{aligned}
\Delta(x_e) &\leq \sum_{e \in \Delta} \left[ x_e \cdot (2^{d_i/c(e)} - 1) + \frac{1}{|V(\Delta_{ij})|} \cdot (2^{d_i/c(e)} - 1) \right] \cdot c(e) \\
&= \sum_{e \in \Delta} \left( x_e + \frac{1}{|V(\Delta_{ij})|} \right) \cdot (2^{d_i/c(e)} - 1) \cdot c(e) \\
&\leq c_{\min}(e) \cdot (2^{d_i/c_{\min}(e)} - 1) \sum_{e \in \Delta} \left( x_e + \frac{1}{|V(\Delta_{ij})|} \right) \\
&\leq d_i \cdot (2^1 - 1) \sum_{e \in \Delta} \left( x_e + \frac{1}{|V(\Delta_{ij})|} \right) \\
&\leq d_i \cdot \sum_{e \in \Delta} x_e + d_i \cdot \sum_{e \in \Delta} \frac{1}{|V(\Delta_{ij})|} \\
&\leq d_i \cdot \sum_{e \in \Delta} x_e + d_i \cdot \qquad (1)
\end{aligned}
$$

Step (2c) increases the cost $\sum_v x_v \cdot c(v)$ as follows (change $\Delta(x_v) = \sum_v (x_v^t - x_v^{t-1}) \cdot c(v)$):

$$
\begin{aligned}
\delta(x_v) &\leq \sum_{v \in \Delta} \left[ x_v \cdot (2^{c_i/c(v)} - 1) + \frac{d_i/c_i}{|V(\Delta_{ij})|} \cdot (2^{c_i/c(v)} - 1) \right] \cdot c(v) \\
&= \sum_{v \in \Delta} \left( x_v + \frac{d_i/c_i}{|V(\Delta_{ij})|} \right) \cdot (2^{c_i/c(v)} - 1) \cdot c(v) \\
&\leq c_{\min}(v) \cdot (2^{c_i/c_{\min}(v)} - 1) \sum_{v \in \Delta} \left( x_v + \frac{d_i/c_i}{|V(\Delta_{ij})|} \right) \\
&\leq c_i \cdot (2^1 - 1) \sum_{v \in \Delta} \left( x_v + \frac{d_i/c_i}{|V(\Delta_{ij})|} \right) \\
&\leq c_i \cdot \sum_{v \in \Delta} x_v + c_i \cdot \sum_{v \in \Delta} \frac{d_i/c_i}{|V(\Delta_{ij})|} \\
&\leq c_i \cdot \sum_{v \in \Delta} x_v + d_i \cdot \qquad (2)
\end{aligned}
$$

after each request, primal variables constitute feasible solutions...