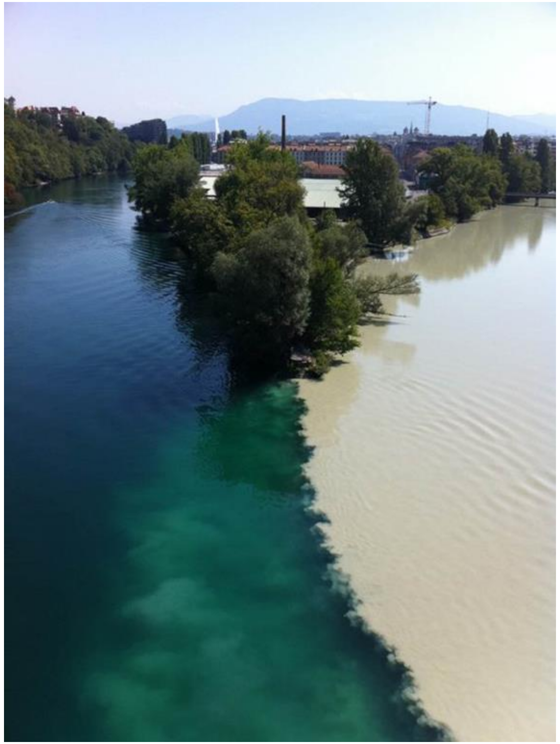
Schedule, Support, Adjust, and Fix: Topology Control Revolutions for Data-Aware Networks

Stefan Schmid (University of Vienna, Austria)

Great Time to be An ALGOSENSORS Researcher!



Rhone and Arve Rivers

Switzerland

Great Time to be An ALGOSENSORS Researcher!

Sensor and wireless networks evolving quickly: in terms of **applications**, **technology** (e.g. ***SDN***) and **scale** (***#devices*** and ***data***).

Rhone and Arve Rivers

Switzerland

Great Time to be An ALGOSENSORS Researcher!

Sensor and wireless networks evolving quickly: in terms of **applications**, **technology** (e.g. ***SDN***) and **scale** (***#devices*** and ***data***).

Rhone and Arve Rivers

Switzerland

Great Time to be An ALGOSENSORS Researcher!

Sensor and wireless networks evolving quickly: in terms of **applications**, **technology** (e.g. ***SDN***) and **scale** (***#devices*** and ***data***).

and Arve Rivers

Switzerland

Actually, Opportunities Have Even More Dimensions!

Passau, Germany

Inn, Donau, Ilz

Actually, Opportunities Have Even More Dimensions!

Example: IoT

- **Converging technologies**
- Traditional fields become "***in***" again: control systems, **automation**

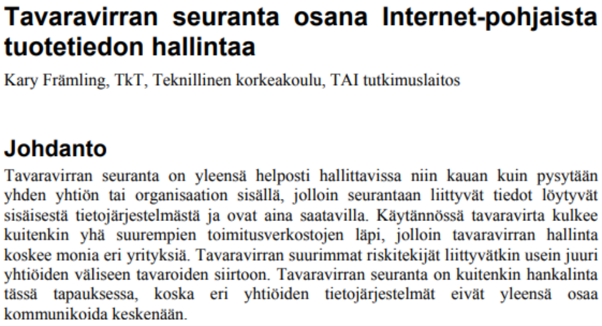
Passau, Germany

Inn, Donau, Ilz

IoT History or: *Finnish for Beginners!*

- ***Network of smart devices*** discussed already **1982**: Coke machine @ CMU connected to the Internet (reports inventory and whether drinks were cold)

The first IoT device



Tavaravirran seuranta osana Internet-pohjaista tuotetiedon hallintaa

Kary Främling, TkT, Teknillinen korkeakoulu, TAI tutkimuslaitos

Johdanto

Tavaravirran seuranta on yleensä helposti hallittavissa niin kauan kuin pysytään yhden yhtiön tai organisaation sisällä, jolloin seurantaan liittyvät tiedot löytyvät sisäisestä tietojärjestelmästä ja ovat aina saatavilla. Käytännössä tavaravirta kulkee kuitenkin yhä suurempien toimitusverkostojen läpi, jolloin tavaravirran hallinta koskee monia eri yrityksiä. Tavaravirran suurimmat riskitekijät liittyvätkin usein juuri yhtiöiden väliseen tavaroiden siirtoon. Tavaravirran seuranta on kuitenkin hankalinta tässä tapauksessa, koska eri yhtiöiden tietojärjestelmät eivät yleensä osaa kommunikoida keskenään.

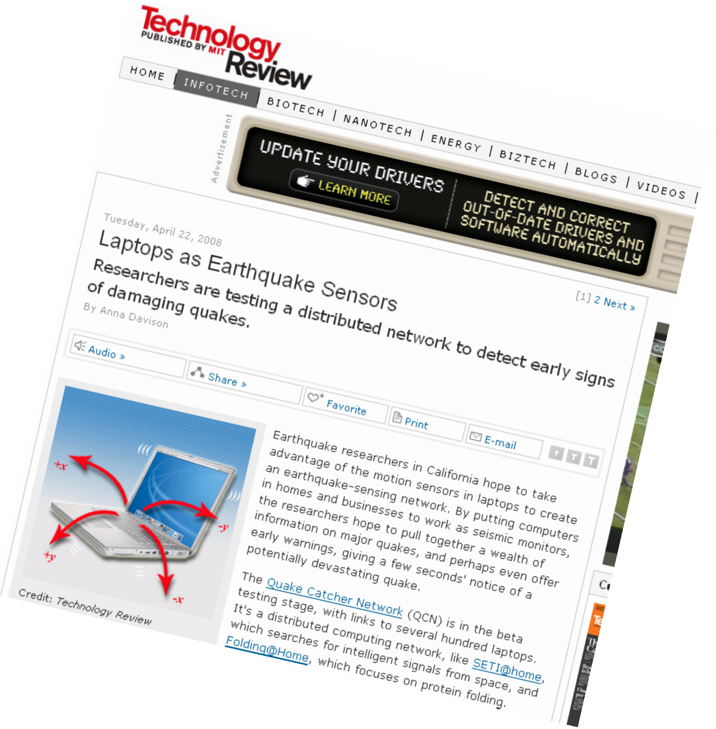
The first IoT paper

- ***First paper*** mentioning the term “IoT” published 2002 in **Finnish** at Helsinki University of Technology

Source: Wikipedia

New Types of Sensor Networks

- Sensor networks based on **laptops**
- E.g., early warning of ***disasters*** such as **earthquakes** (e.g., to stop high-speed trains)
  - Using integrated **accelerometer** (originally to protect harddrive when falling)
  - ***Fill the „gaps“*** between seismometers already in place in California
  - E.g., Apple laptops since 2005



Tuesday, April 22, 2008

Laptops as Earthquake Sensors

Researchers are testing a distributed network to detect early signs of damaging quakes.

By Anna Davison

Credit: *Technology Review*

Earthquake researchers in California hope to take advantage of the motion sensors in laptops to create an earthquake-sensing network. By putting computers in homes and businesses to work as seismic monitors, the researchers hope to pull together a wealth of information on major quakes, and perhaps even offer early warnings, giving a few seconds' notice of a potentially devastating quake.

The Quake Catcher Network (QCN) is in the beta testing stage, with links to several hundred laptops. It's a distributed computing network, like SETI@home, which searches for intelligent signals from space, and Folding@Home, which focuses on protein folding.

New Types of Applications: Smart Devices Move…

- E.g., smart devices/things move with their owners: (social) **mobile network**

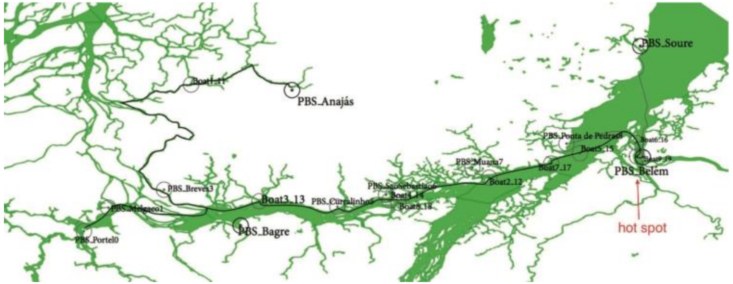
- May have **intermittent** connectivity and must hence be **delay-tolerant**
    - E.g., student (“data mule”) commuting between hotspots

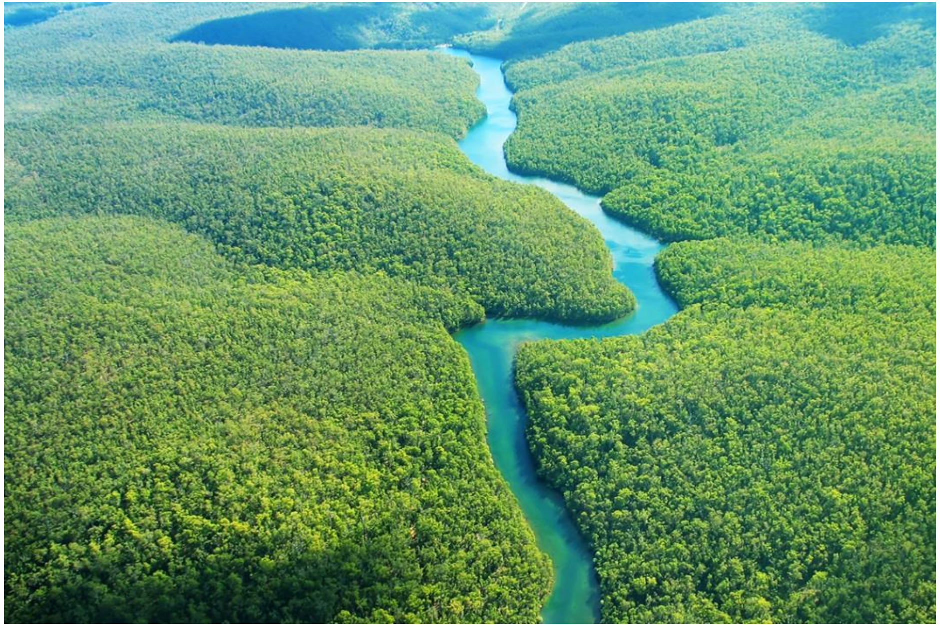
*MULE = Mobile Ubiquitous LAN Extension

Case Study: “Data Mules” in Amazon Riverine

Enjoy: last river in this presentation!

- By ***Richa et al.***, with Brazilian collaborators
- Challenge: providing **health care** to the remote communities in the Brazilian Amazon
- Constraint: Lack of modern **communication infrastructure** in these communities
- Solution: **delay-tolerant network**
  - ***Local nurses*** perform routine clinical examinations, such as **ultrasounds** on pregnant women
  - **Records sent** to the doctors in Belem (city) for evaluation
  - Idea: use of regularly **scheduled boats** as data mules





Source: Liu et al. Robust data mule networks with remote healthcare applications in the Amazon region. HealthCom 2015: 546-551

… Fly …

- E.g., 1000+ high-tech **drones** at Winter olympics

… Collect Lots of Data While Flying …: The “Internet of Aircraft Things”

- Geared Turbo Fan (GTF) **engines** fitted with **5000 sensors** that generate **10 GB** of data per second
    - I.e., twin-engine aircraft with ***12 h flight*** time: **>800 TB** of data
- Usage, e.g. **AI** for prediction of engine demands to ***adjust thrust levels***

Source: aviationweek.com

Hey, Robots!

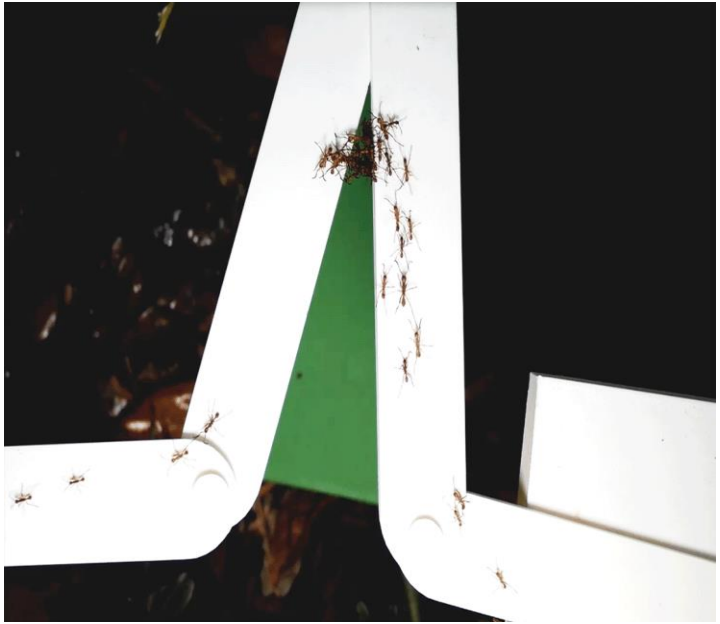
Last Week’s Dagstuhl Seminar on Programmable Matter and Swarm Robotics

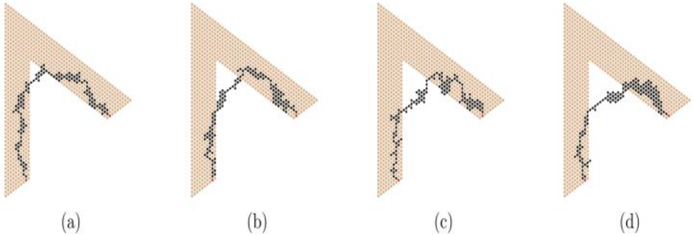
- Emergence of large number **simple and small** robots that, when combined, can perform **complex** tasks
- Often inspired by **nature**: ***What can we learn from natural swarms?***

Dagstuhl Seminar “**Programmable Matter**”

Example: Smart Natural Swarms

- Army ants (*Eciton*) can solve ***non-trivial optimization*** problems, e.g., **build "ant-bridges"** as ***shortcut*** along supply chain
- Bridge location ***depends on angle*** of gap
- **Tradeoff**: ***longer bridges*** make the total path ***shorter***, but need to ***sacrifice more workers***





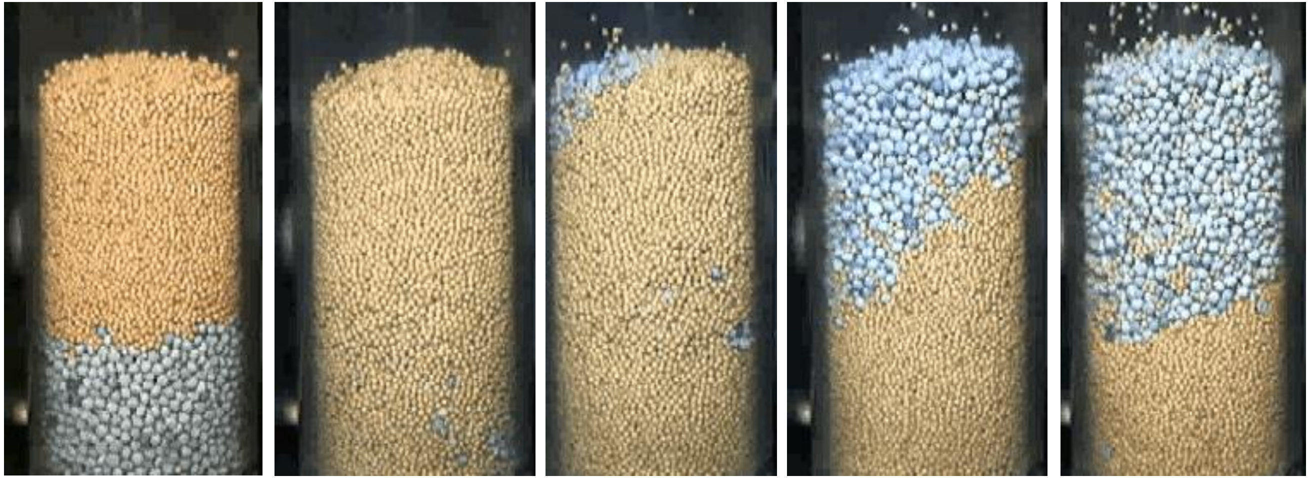
RLPKCG 2015: "Army ants dynamically adjust living bridges…"

Dagstuhl Seminar "Programmable Matter"
Thanks to Andrea Richa, ASU

Example: Exploit Brazil Nut Effect

- **Task:** separate different robots

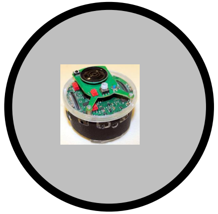
- **Inspiration:** exploit nuts effect and gravity: shaking / random move of nuts: ***big nuts*** up, ***small nuts*** down



From: University of Amsterdam

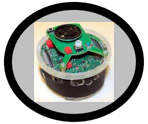
E-puck

- **Solution:** robots with **distance sensors**:



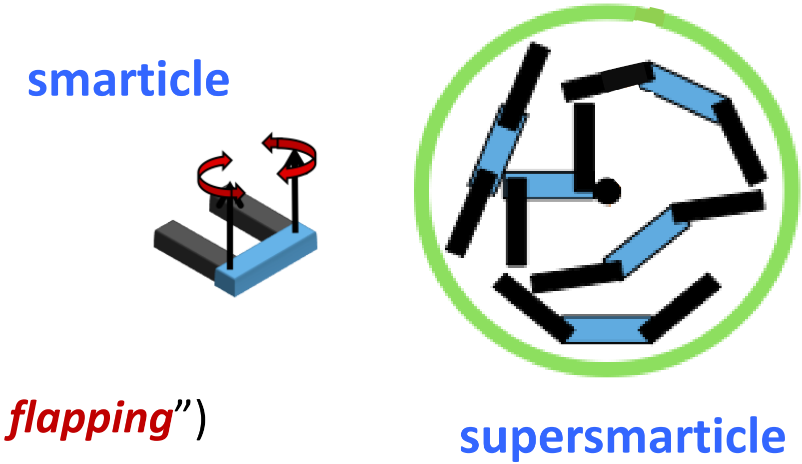
Big nut: large ***collision radius***

Small nut: small collision radius



Locomotion by *Combining* Very Simple Robots

*A **ring-robot** made of robots*



- Very simple things: **smarticle** (alone: “random ***flapping***”)
  - One smarticle: ***no locomotion***
- But when interaction with ***multiple*** smarticles confined in a ***ring***, with ***one inactive*** smarticle:
  - **Brownian motion w/ drift** (toward **inactive** smarticle)

Discovered ***by accident***: one smarticle died!

[Cannon, Daymude, Goldman, Li, Randall, Richa, Savoie, SWARM’17]

Further Applications

Reconfigurable robots ("transformers")

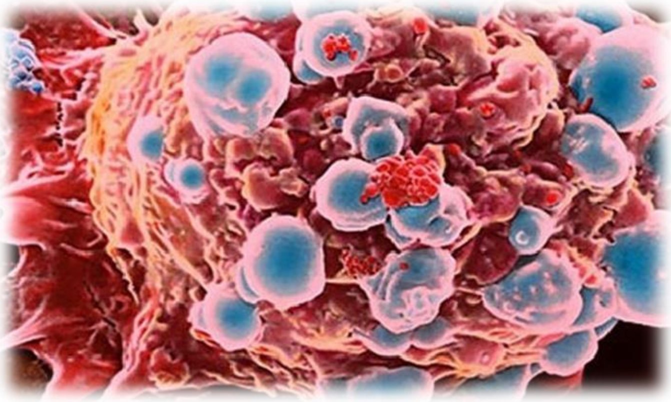
Flora Robotica: e.g., robots to grow houses

"Helpful" robots ☺

Dagstuhl Seminar "Programmable Matter"
© Roderich Gross, Natural Robotics Lab
© Julien Bourgeois, FEMTO-ST
© Heiko Hamann, Service Robotics

“IoT trend” also for robots: Many becoming smaller in size but larger scale

Kilobots, Harvard



Monitor health and assist
in **surgeries**

We still lack ***models*** (first attempts such as ***Amoeba*** model, name *depricated*)!

Dagstuhl Seminar “Programmable Matter”
Thanks to Andrea Richa, ASU

Common Theme: *Bigger*

In terms of **size**:

E.g., 8.4 billion IoT devices in 2017, 30 billion devices expected by 2020

But also in terms of **data**:

„Data generated by ***aerospace industry*** alone could be in the order of the ***consumer Internet***.“ AviationWeek.com

New challenges, new solutions required!

“Large scale and big data”:
Babyphone Attack (Fall 2016)

- First big IoT device **attack**
- Attackers exploited household devices: IP-cameras, printers, **babyphones**, tv recorders, ...
- DoS attack of more than ***500 Gbps***!
- Twitter, Netflix, Spotify, … unreachable for several hours

Security / #CyberSecurity

SEP 25, 2016 @ 10:00 AM 41,011 VIEWS

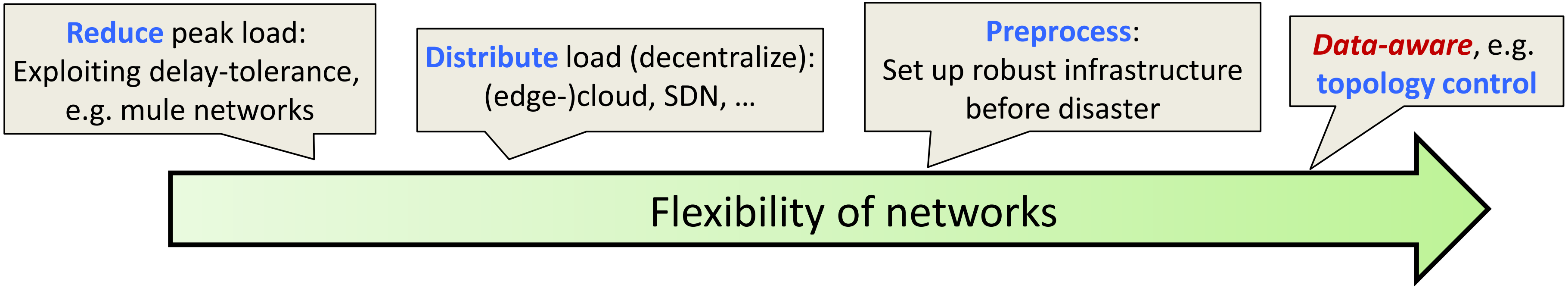
How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen

Thomas Fox-Brewster, FORBES STAFF

I cover crime, privacy and security in digital and physical forms. FULL BIO

“Cyber-attack from the babyphone” – Spiegel, 2016

Dealing With These Challenges: Exploiting Algorithmic Flexibilities



Reduce peak load: Exploiting delay-tolerance, e.g. mule networks

Distribute load (decentralize): (edge-)cloud, SDN, …

Preprocess: Set up robust infrastructure before disaster

Data-aware, e.g. **topology control**

Flexibility of networks

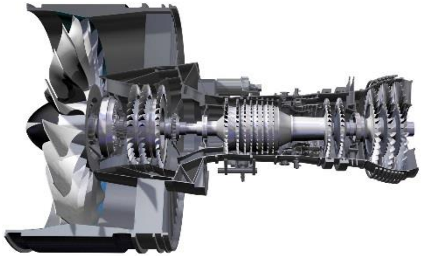
In general: make algorithms „aware“ of network flexibilities and specific properties of the workload!

Emerging applications and large-scale sensor networks processing big data require ***new models and algorithms***!

Some (early) examples.

General Remark:
CS = “The Science of the Machine”

- Technological **enablers** are there, but emerging machines ***hardly understood*** today!
- ***Models***? Practical constraints? Objectives? Etc.
- Algorithmic ***opportunities***?



“data mule”

Credits: Marcos K. Aguilera

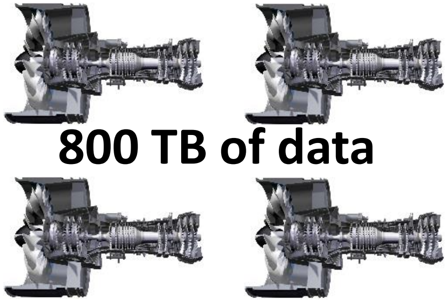
PODC 2018

Opportunity: Choose Level of Decentralization

Centralized vs Decentralized: Example “Edge Cloud / Distributed Cloud”

- ***Big scale*** and big data: **central cloud** is not always the best solution
- Motivation for ***applying scale out*** to the datacener: **distributed cloud**
- Lower latency, less bandwidth
- E.g. airplane example

Many more levels of edges: e.g., routers/switches



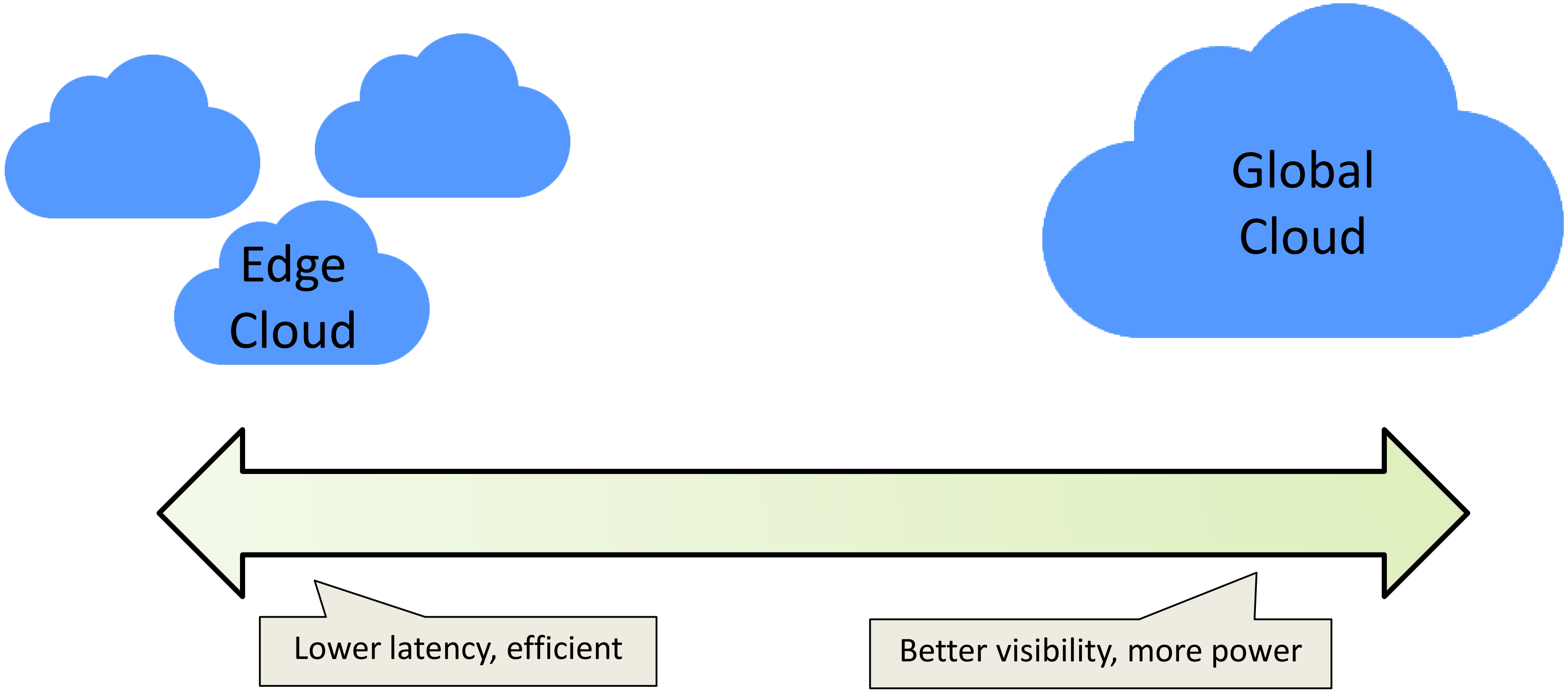
800 TB of data

low latency
high bandwidth

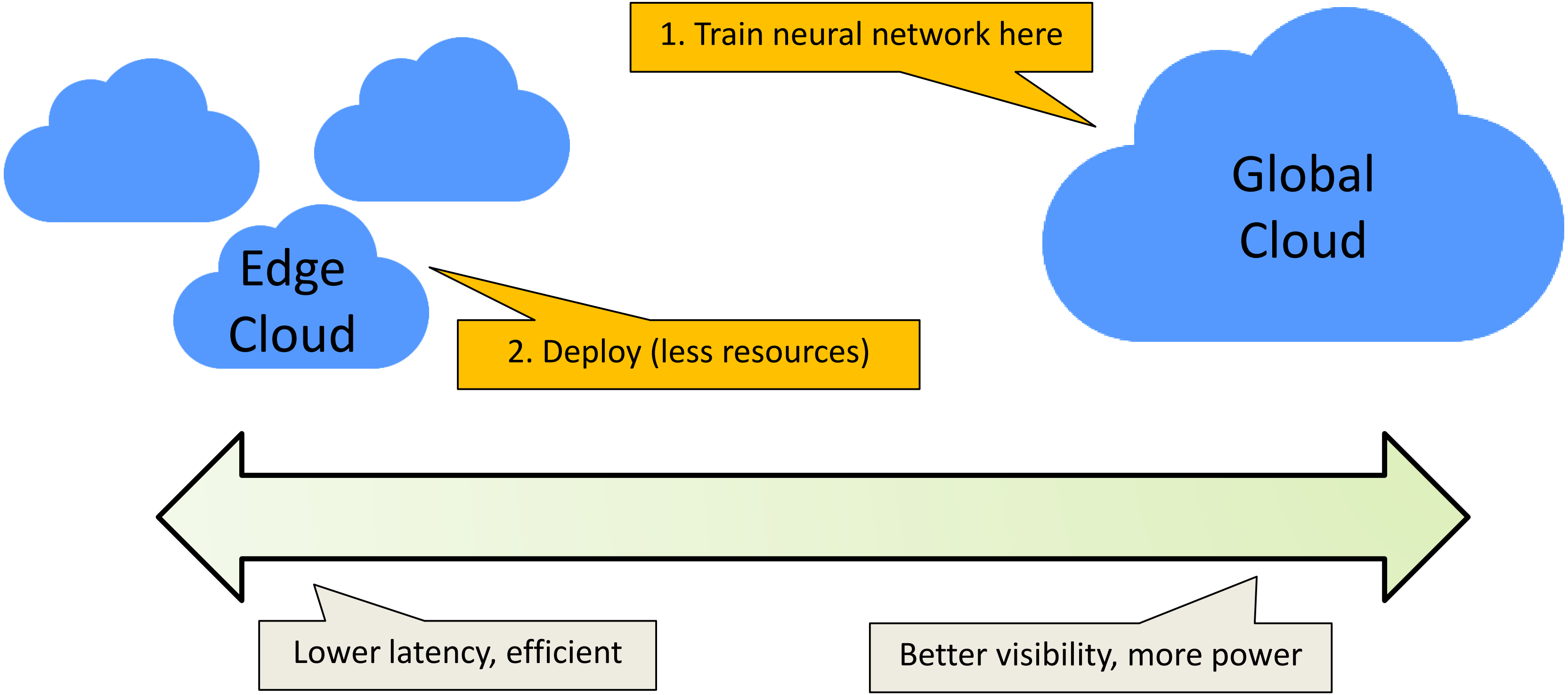
Cloud

high latency
low bandwidth

Centralized vs Decentralized: A Tradeoff



Centralized vs Decentralized: A Tradeoff



1. Train neural network here

Global Cloud

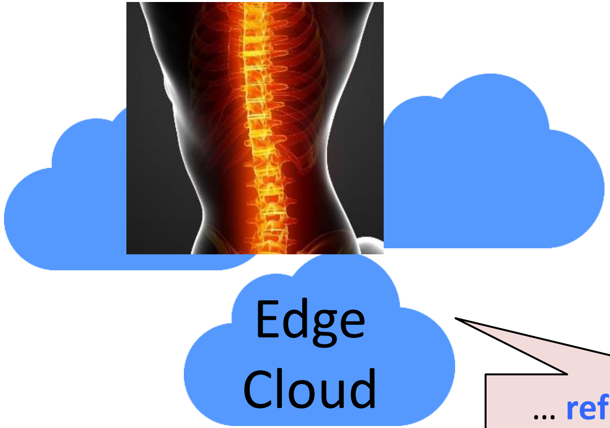
Edge Cloud

2. Deploy (less resources)

Lower latency, efficient

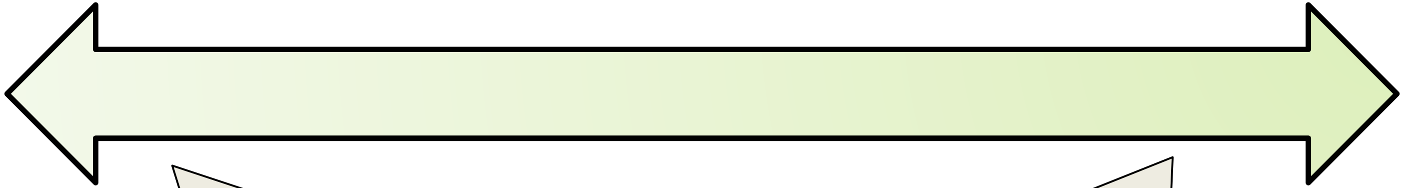
Better visibility, more power

Centralized vs Decentralized: A Tradeoff



Learning“with the brain”…

… **reflexes** go to the spine.



Lower latency, efficient

Better visibility, more power

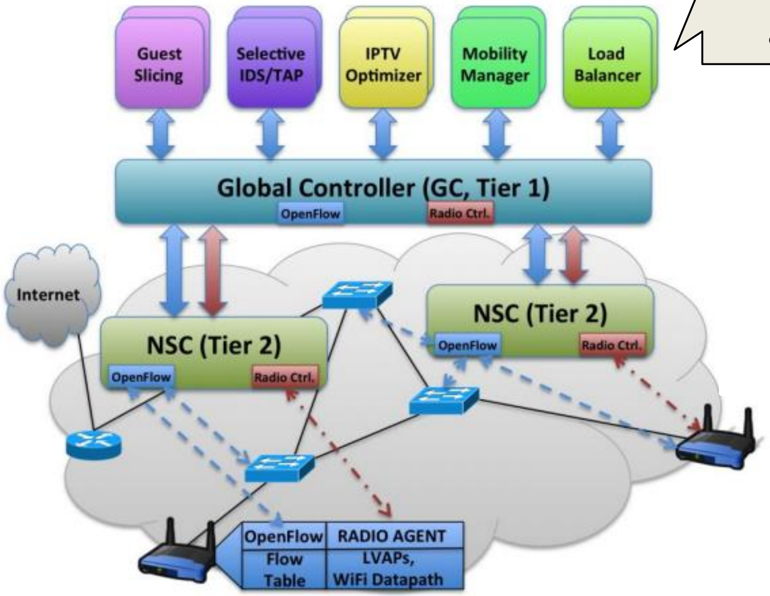
Centralized vs Decentralized: Example “Software-Defined Networks (SDN)”

- Interesting new technology for ***Algosensors***: (Wireless) networks become programmable („ **software-defined**“)

- Easy to deploy your own algorithm for: routing, **rate/power control**, interference/ **mobility** management, load-balancing, etc. :
  - Opportunity for research & innovation: „the ***linux of networking***“

Interesting dimension: distributed control plane

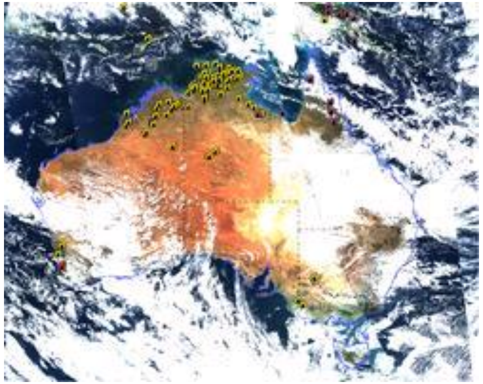
- **Global controller** for coarse-grained control
  - Services that require global visibility (e.g. ***spanning tree*** or shortest path)
- **Near-sighted controller** for fine grained control
  - ***Latency-critical*** transmission control ***or load-intensive*** tasks

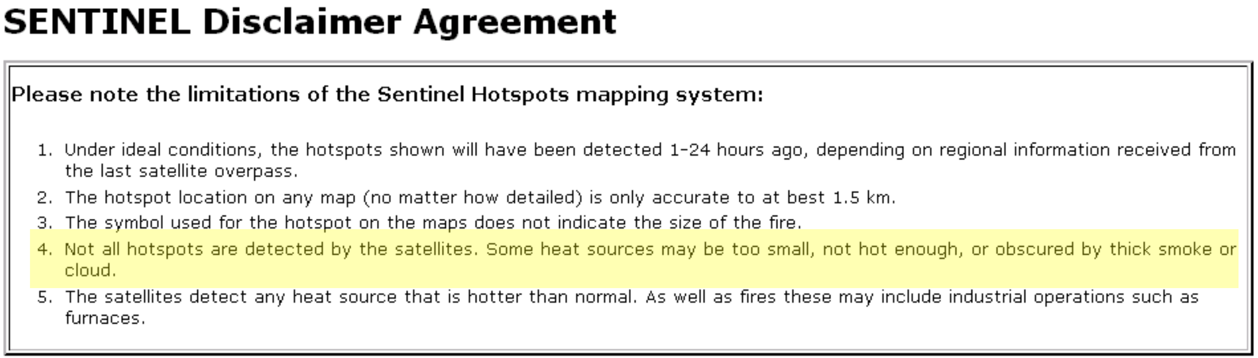


Programmable: Your algorithm goes here!

Going back to Distaster Detection“Bushfire Monitoring”: Centralized vs Decentralized?

- Example: SENTINEL Australian **bushfire monitoring** system
- Centralized, based on **satellites**
- However, satellites may ***miss certain heat*** sources, e.g., if there is ***smoke***!
- **Distributed** sensor nodes (in addition) can be a good alternative





SENTINEL Disclaimer Agreement

Please note the limitations of the Sentinel Hotspots mapping system:

1. Under ideal conditions, the hotspots shown will have been detected 1-24 hours ago, depending on regional information received from the last satellite overpass.
2. The hotspot location on any map (no matter how detailed) is only accurate to at best 1.5 km.
3. The symbol used for the hotspot on the maps does not indicate the size of the fire.
4. Not all hotspots are detected by the satellites. Some heat sources may be too small, not hot enough, or obscured by thick smoke or cloud.
5. The satellites detect any heat source that is hotter than normal. As well as fires these may include industrial operations such as furnaces.

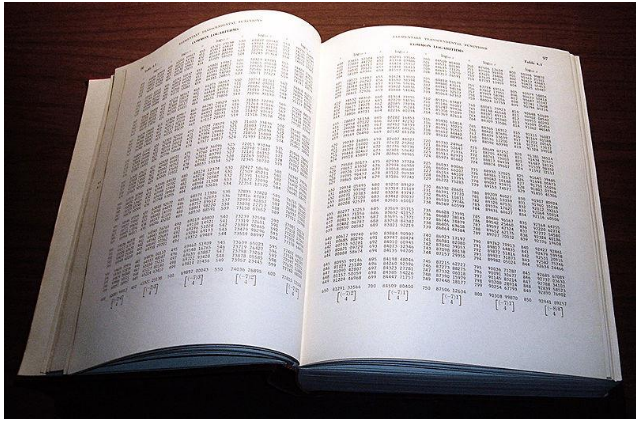
Research Challenges

So what can be computed ***locally***, what should be ***global*** comes in many new flavors:

- E.g., in distributed / edge cloud: Trend of moving away from ***client-server***: How does „**client-edge-server**“ change consistency, fault-tolerance etc. compare to „user-cloud/server“? New ***distributed computing challenges***!
- E.g., in software-defined wireless networks
- Etc.

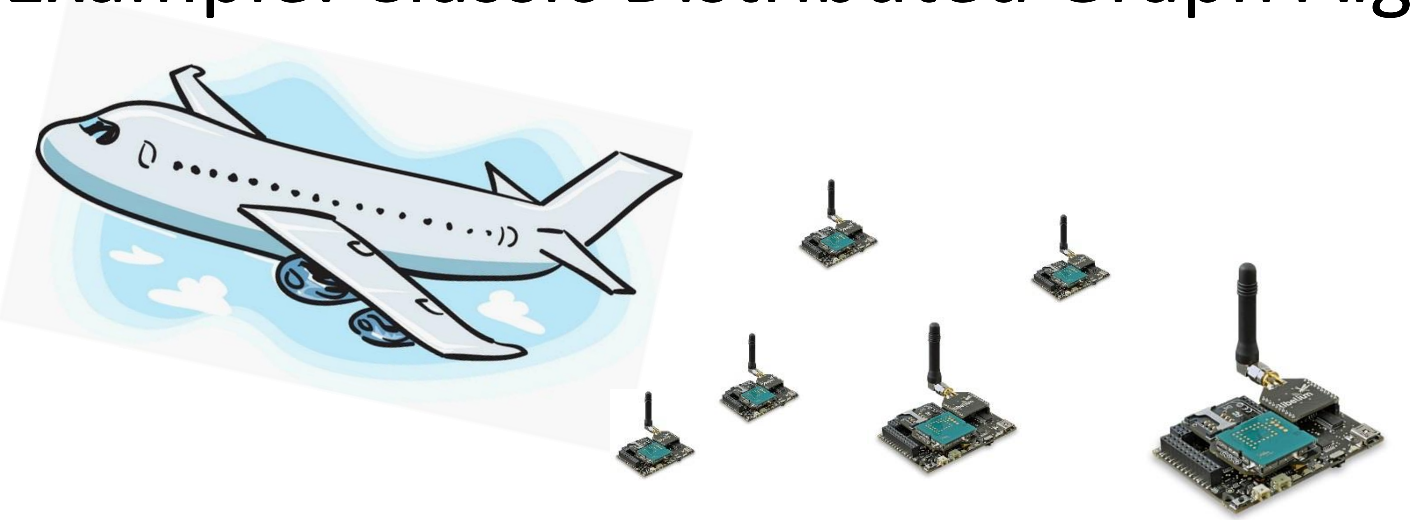
The **clue**: Unlike classic distributed graph algorithms, there is a ***choice***! Hybrid distributed-centralized networks

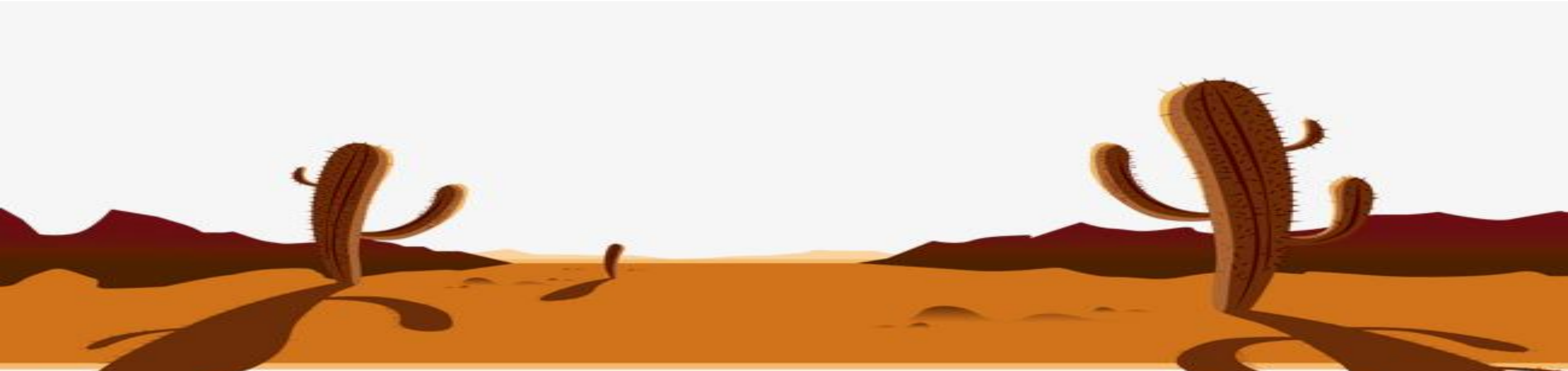
Opportunity: Precomputation



Precomputed logarithms
(20th century book)

Example: Classic Distributed Graph Algorithms





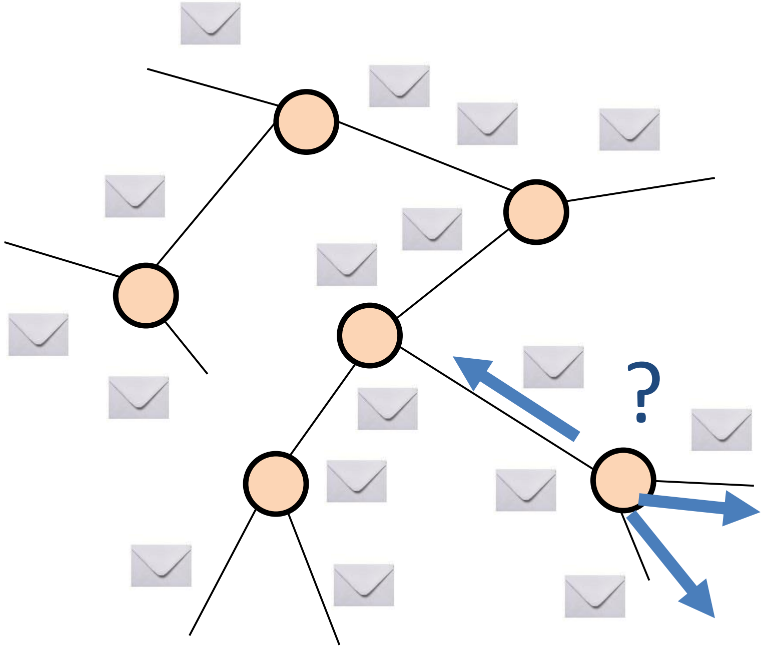
The Power of Precomputation?

- Example: Distributed graph algorithms

- Traditional model (LOCAL and CONGEST):
  - Start **from scratch**
  - Each node first needs to explore ist neighborhood, break **symmetries**, etc.

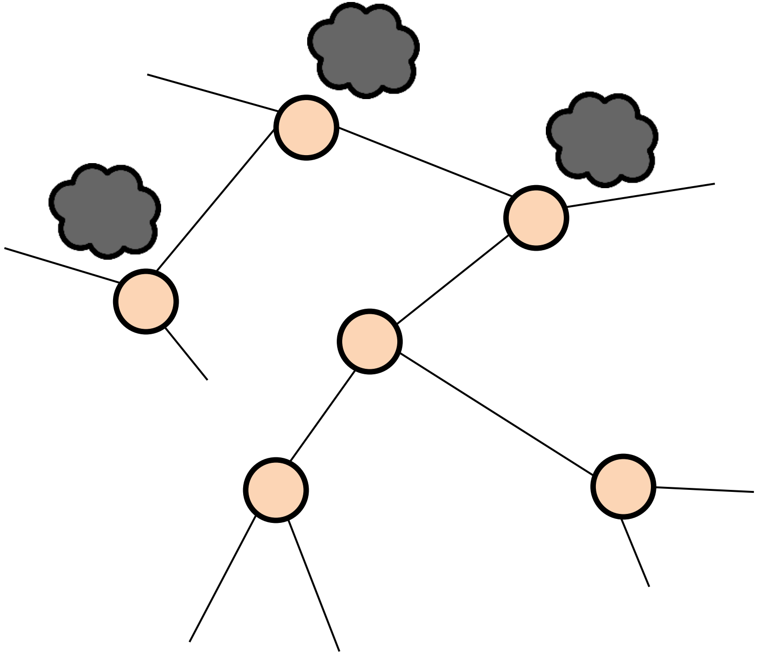
Sometimes, some **pre-computation** is possible: E.g., (rough) idea how network looks like (***node locations*** or links «***before failures***»)!
  - Just don’t know, e.g., **failures** or **demand**

- So:
  - Which information is ***useful to precompute*** so that later distributed algorithms are faster?
  - How to ***exploit*** it in algorithms?

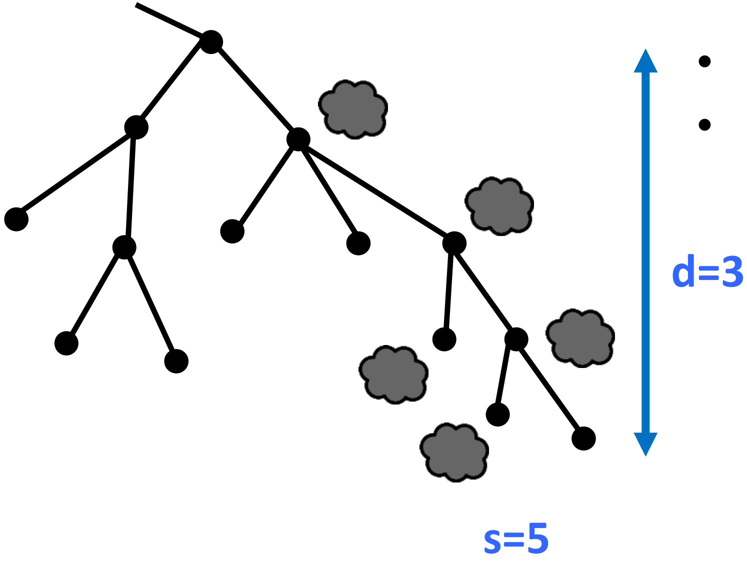


Going Back to Bushfire: Estimate the Disaster Size

- A simple **model**: ***bush fire*** breaks out, smoke detected by local nodes
- Goal: efficiently detect ***size of disaster***, i.e., of connected «**event component**»
  - at least 1 node should know
- Ideally, don't waste energy for ***small events***: algorithm should be **output sensitive**. In case of „small disasters", only a small number of messages is transmitted
- Different model flavors:
  - ***On-duty:*** non-affected nodes can help too
  - ***Off-duty:*** they cannot help



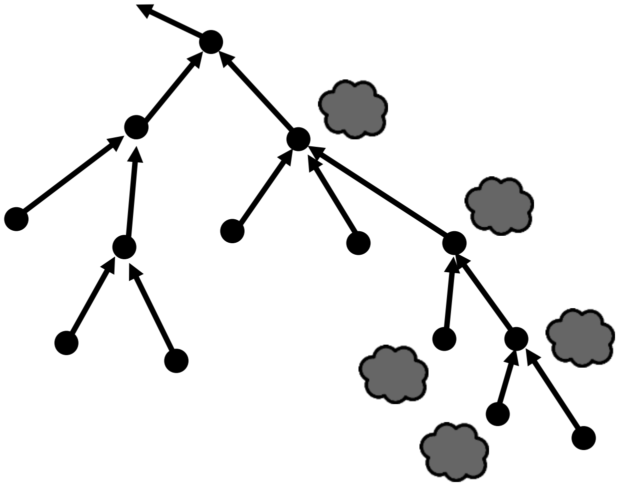
Example: Disaster Estimation on Trees



- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity s, time complexity d

How to achieve this?

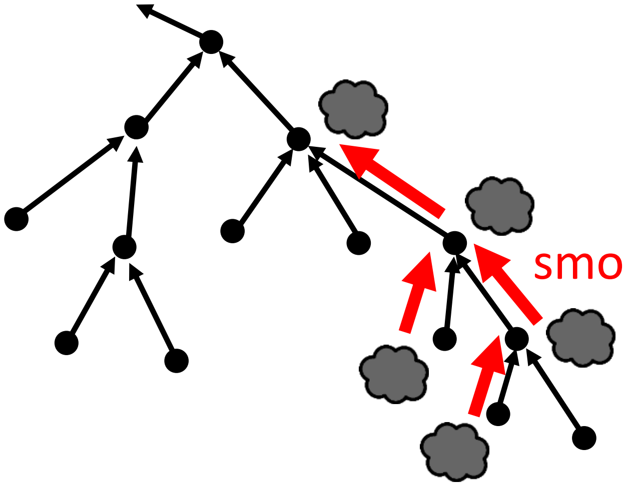
Example: Disaster Estimation on Trees



- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Preprocess: make tree **rooted** and **directed**!

Example: Disaster Estimation on Trees

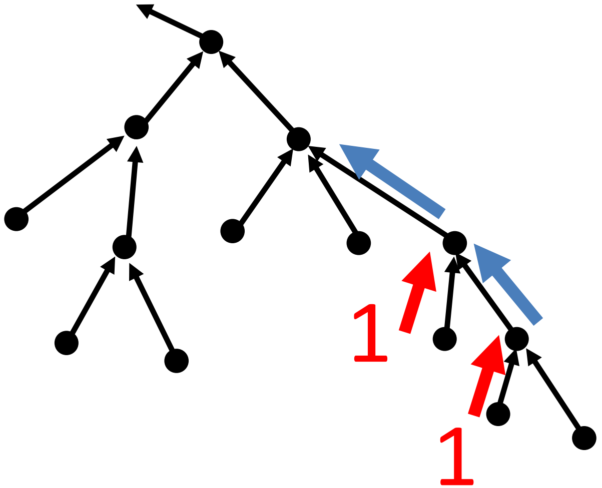


- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Round 1

1. each node v **immediately informs** its parent in case of an event «sensed»

Example: Disaster Estimation on Trees

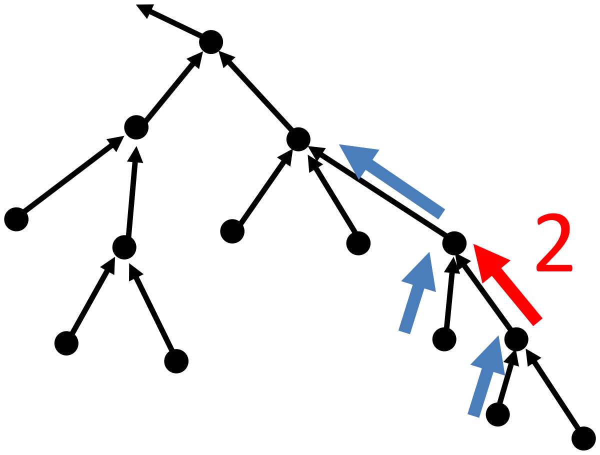


- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Round 2

1. each node v immediately informs its parent in case of an event «sensed»
2. wait until all **event-children** counted the total number of event nodes in their subtrees

Example: Disaster Estimation on Trees

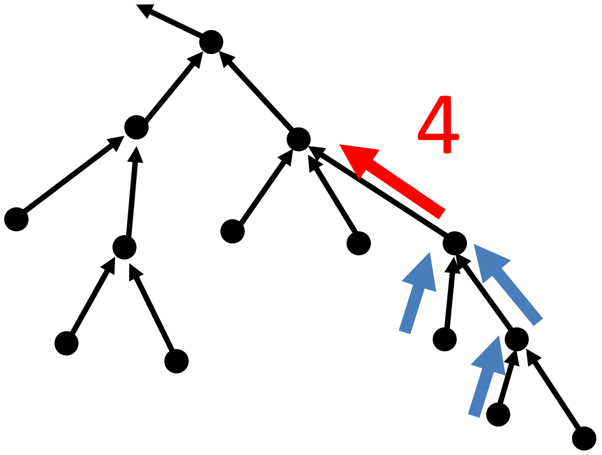


- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Round 3

1. each node v immediately informs its parent in case of an event «sensed»
2. wait until all **event-children** counted the total number of event nodes in their subtrees
3. **propagate** up

Example: Disaster Estimation on Trees

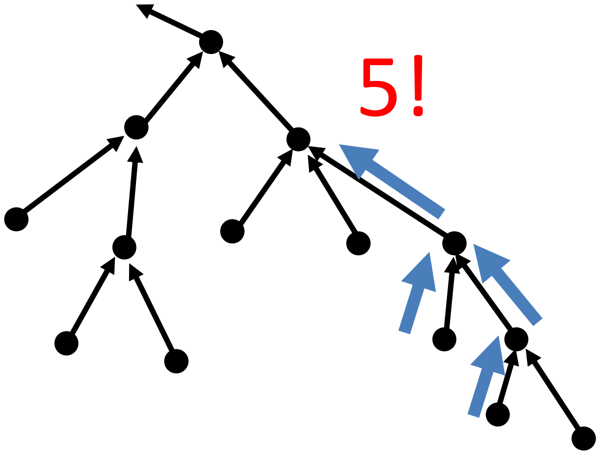


- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Round 4

1. each node v immediately informs its parent in case of an event «sensed»
2. wait until all **event-children** counted the total number of event nodes in their subtrees
3. **propagate** up

Example: Disaster Estimation on Trees



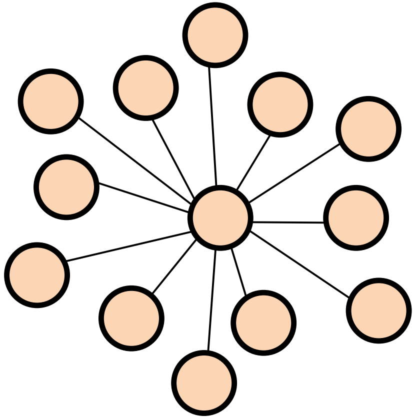
- **Assume:** disaster of ***size s*** of ***diameter d***
- **Ideally:** message complexity **s**, time complexity **d**

Round 4

1. each node v immediately informs its parent in case of an event «sensed»
2. wait until all **event-children** counted the total number of event nodes in their subtrees
3. **propagate** up

Preprocessing For Neighborhood Discovery

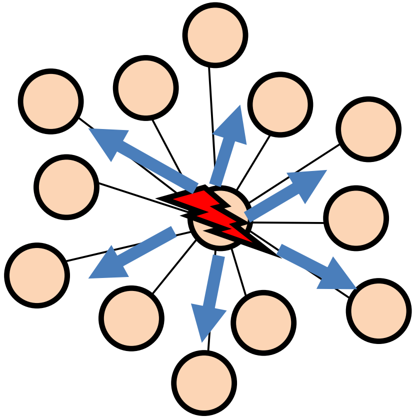
- How to do this on ***general graphs***?
- Even more **basic problem**: How to efficiently find out which of my neighbors also sensed an event? **Neighborhood discovery!** Goal again: «output-sensitive»
- Idea: «Just inform all neighbors!»



Preprocessing For Neighborhood Discovery

- How to do this on ***general graphs***?
- Even more **basic problem**: How to efficiently find out which of my neighbors also sensed an event? **Neighborhood discovery!** Goal: «output-sensitive»!
- Idea: «Just inform all neighbors!»

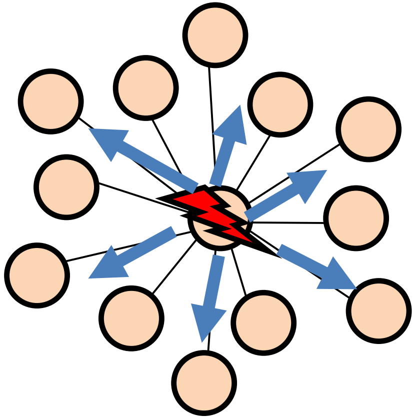
Problem:
disaster size 1, but cost n



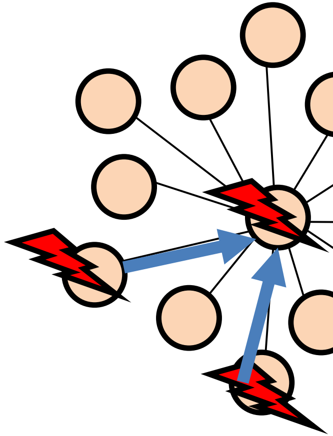
Preprocessing For Neighborhood Discovery

- How to do this on ***general graphs***?
- Even more **basic problem**: How to efficiently find out which of my neighbors also sensed an event? **Neighborhood discovery!** Goal: «output-sensitive»!
- Idea: «Just inform all neighbors!»

Problem:
disaster size 1, but cost n



- Another idea: «Low-degree nodes inform ***high-degree neighbors***!»



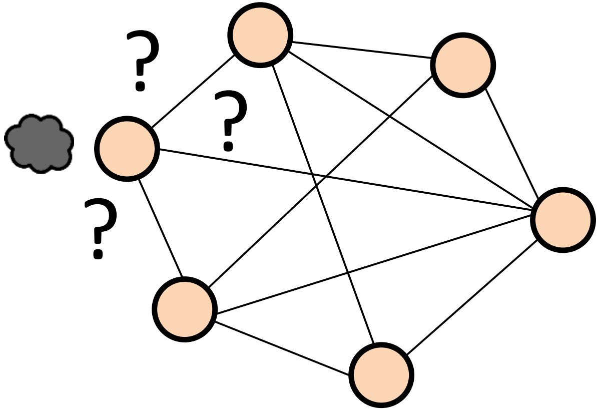
Efficient here!
(Note: on-duty only)

Preprocessing For Neighborhood Discovery

But what about symmetric graphs like **clique**?

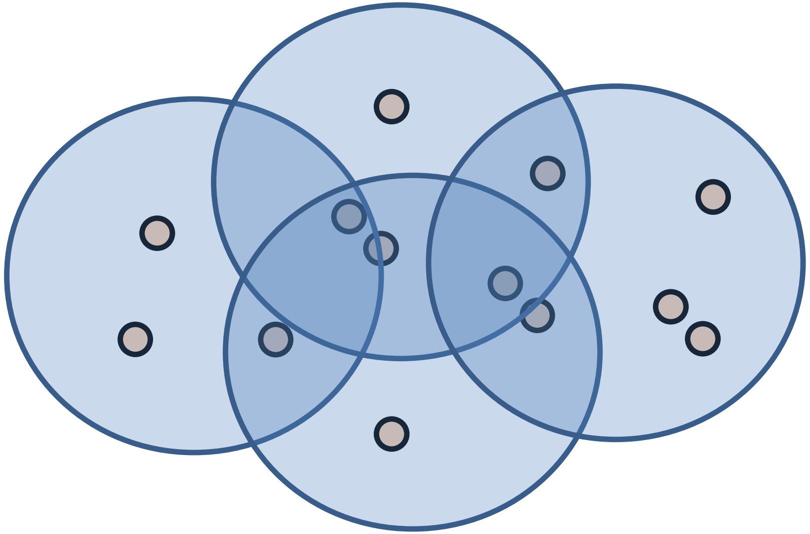
Again **neighborhood discovery** only: how to know which neighbors sensed the event with ***O(s) messages*** in total only?

Can we avoid cost n for small components?



Useful Preprocessing For Neighborhood Discovery: Neighborhood Cover

Yes we still can ***leverage preprocessing***: graph **decompositions**! (On-duty only.)



E.g., pre-process sparse (k,t)- **neighborhood cover**, clustering ensures that:

- Each **t-neighborhood** included entirely in ***at least one cluster***
- Diameter of cluster at most **O(kt)**
- Sparse: node part of at most **$kn^{1/k}$** clusters

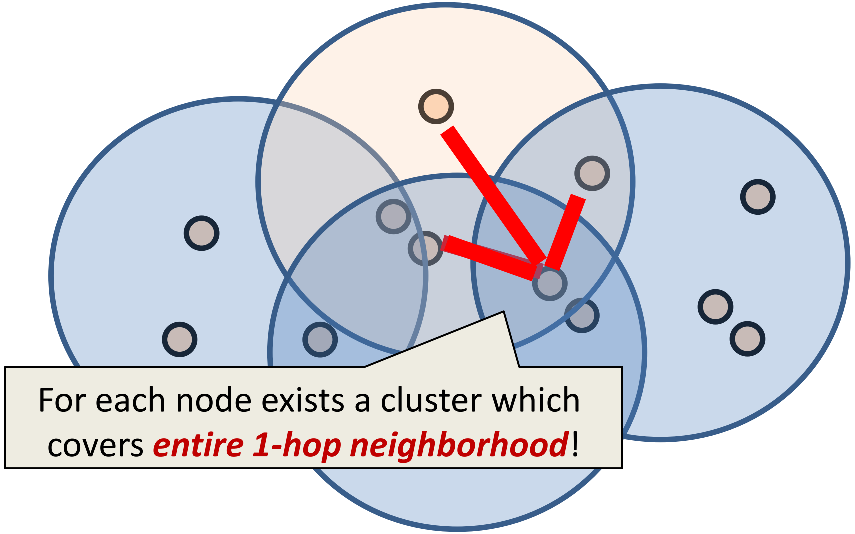
Idea:

- Preprocess neighborhood cover **k=log n**, **t=1**
- Assign one node per cluster («**cluster head**») collects «***who sensed event***» information
- Since **low diameter**: nodes can send to cluster head in ***log n hops***
- Since **sparse**: Nodes need to send to at most ***log n cluster heads***

Time O(log n), Message complexity O(polylog n)

Useful Preprocessing For Neighborhood Discovery: Neighborhood Cover

Yes we still can ***leverage preprocessing***: graph **decompositions**! (On-duty only.)



For each node exists a cluster which covers ***entire 1-hop neighborhood***!

E.g., pre-process sparse (k,t)- **neighborhood cover**, clustering ensures that:

- Each **t-neighborhood** included entirely in ***at least one cluster***
- Diameter of cluster at most **O(kt)**
- Sparse: node part of at most **$kn^{1/k}$** clusters

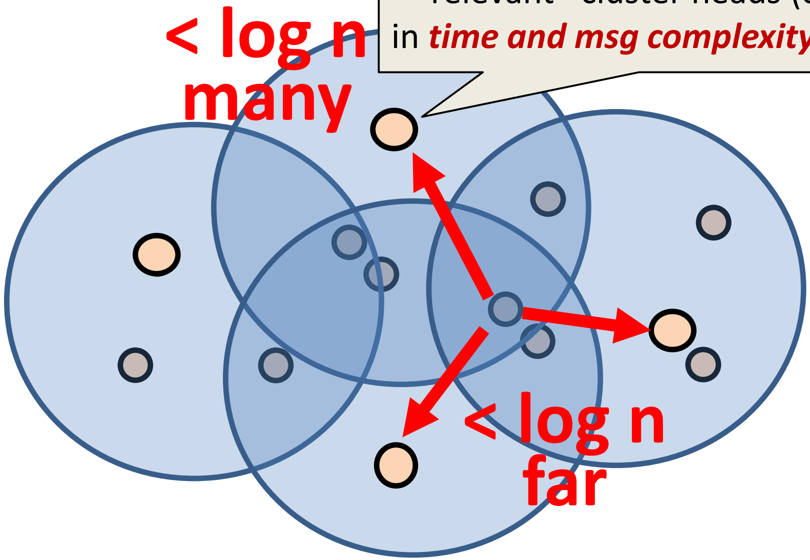
Idea:

- Preprocess neighborhood cover **k=log n**, **t=1**
- Assign one node per cluster («**cluster head**») collects «***who sensed event***» information
- Since **low diameter**: nodes can send to cluster head in ***log n hops***
- Since **sparse**: Nodes need to send to at most ***log n cluster heads***

Time O(log n), Message complexity O(polylog n)

Useful Preprocessing For Neighborhood Discovery: Neighborhood Cover

Yes we still can ***leverage preprocessing***: graph **decompositio**



Each node *v* ***only needs to inform*** “relevant” cluster heads (covering *v*) in ***time and msg complexity polylog(n)***.

E.g., pre-process sparse (k,t)- **neighborhood cover**, clustering ensures that:

- Each **t-neighborhood** included entirely in ***at least one cluster***
- Diameter of cluster at most **O(kt)**
- Sparse: node part of at most **$kn^{1/k}$** clusters

Idea:

- Preprocess neighborhood cover **k=log n**, **t=1**
- Assign one node per cluster («**cluster head**») collects «***who sensed event***» information
- Since **low diameter**: nodes can send to cluster head in ***log n hops***
- Since **sparse**: Nodes need to send to at most ***log n cluster heads***

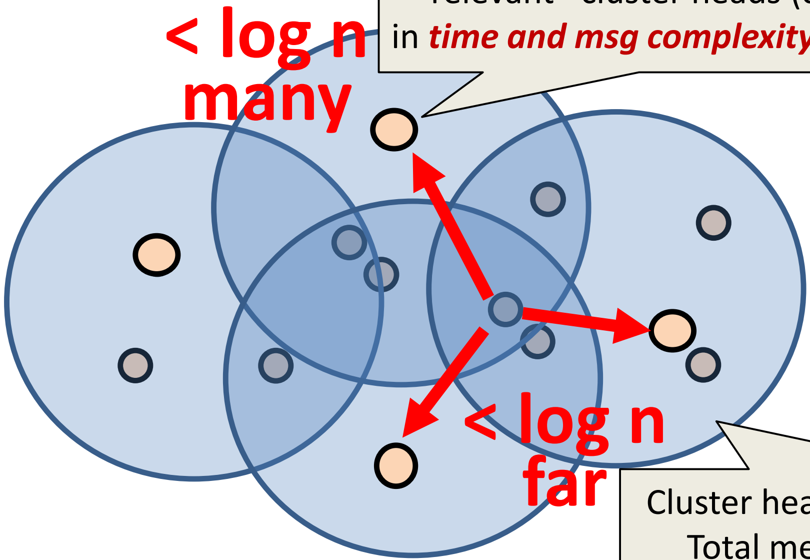
Time O(log n), Message complexity O(polylog n)

Useful Preprocessing For Neighborhood Discovery: Neighborhood Cover

Yes we still can ***leverage preprocessing***: graph **decompositio**

E.g., pre-process sparse (k,t)- **neighborhood cover**, clustering ensures that:

- Each **t-neighborhood** included entirely in ***at least one cluster***
- Diameter of cluster at most **O(kt)**
- Sparse: node part of at most **$kn^{1/k}$** clusters



Each node *v* ***only needs to inform*** “relevant” cluster heads (covering *v*) in ***time and msg complexity polylog(n)***.

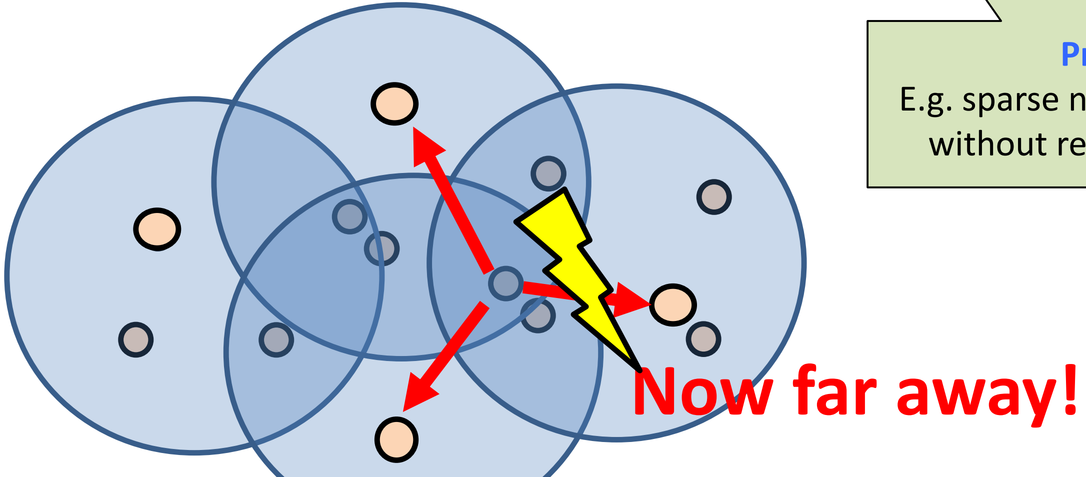
Idea:

- Preprocess neighborhood cover **k=log n**, **t=1**
- Assign one node per cluster («**cluster head**») collects «***who sensed event***» information
- Since **low diameter**: nodes can send to cluster head in ***log n hops***
- Since **sparse**: Nodes need to send to at most ***log***

plexity O(polylog n)

Cluster heads can ***communicate neighborhood***.
Total message complexity in ***O(s polylogn)***
(for neighborhood discovery alone).

But what if we cannot use "event nodes" (e.g., due to smoke/heat)?! Off-duty model!



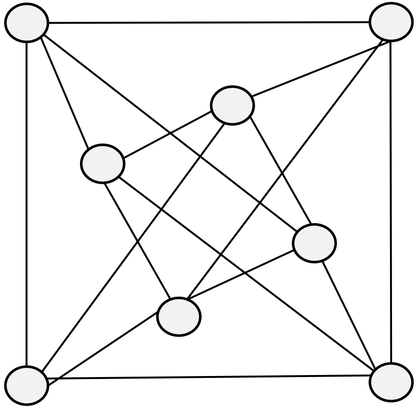
Preprocessing useful at all?
E.g. sparse neighborhood cover ***loses properties***:
without relay, cluster head may be ***far away***!

Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** α

Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_\alpha\}$: α forests **covering** the entire original network

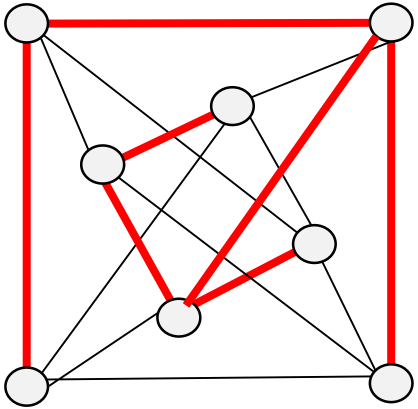


Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** ɑ

Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_\alpha\}$: ɑ forests **covering** the entire original network



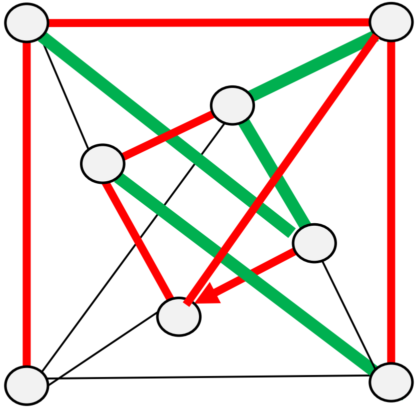
Forest F_1 (a tree)

Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** α

Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_\alpha\}$: α forests **covering** the entire original network



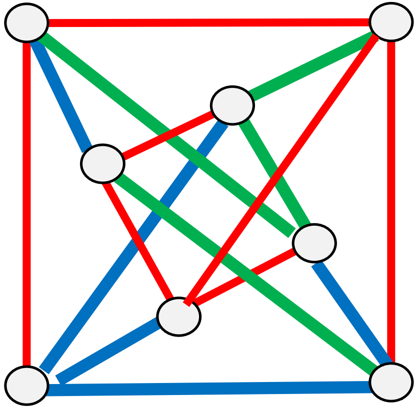
Forest F_2

Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** ɑ

Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_ɑ\}$: ɑ forests **covering** the entire original network



Forest F_3

Edge partition!

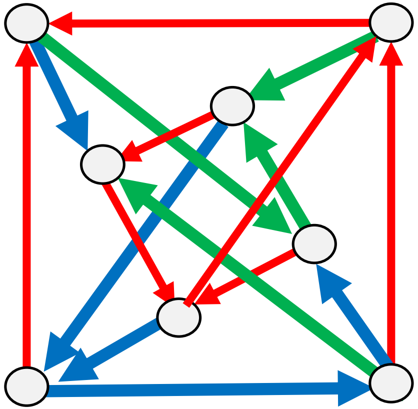
Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** α

Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\dots,F_\alpha\}$: α forests **covering** the entire original network

Solution to "**neighborhood problem**": Preprocess forests by making them ***rooted and directed***.



Forest F_3

Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** α

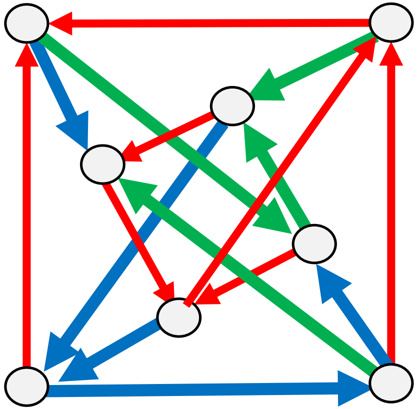
Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_\alpha\}$: α forests **covering** the entire original network

Solution to "**neighborhood problem**": ***Preprocess*** forests by making them ***rooted and directed***.

- Let P_v be set of all parents of a node in these forests: $|P_v| \leq \alpha$

The clue: degree may be high, but number of parents not!



Forest F_3

Precomputation in Off-Duty Model: Covering Forests

- Consider graph of **arboricity** ɑ

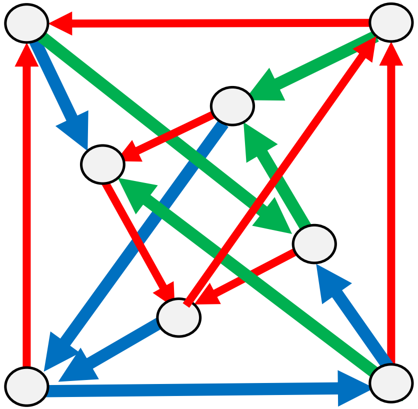
Minimum number of forests into which graph ***edges can be partitioned***.

- ***Pre-compute*** rooted and directed **forests** $\{F_1,\ldots,F_ɑ\}$: ɑ forests **covering** the entire original network

Solution to "**neighborhood problem**": ***Preprocess*** forests by making them ***rooted and directed***.

- Let P_v be set of all parents of a node in these forests: $|P_v| \leq a$

The clue: degree may be high, but number of parents not!



Forest F_3

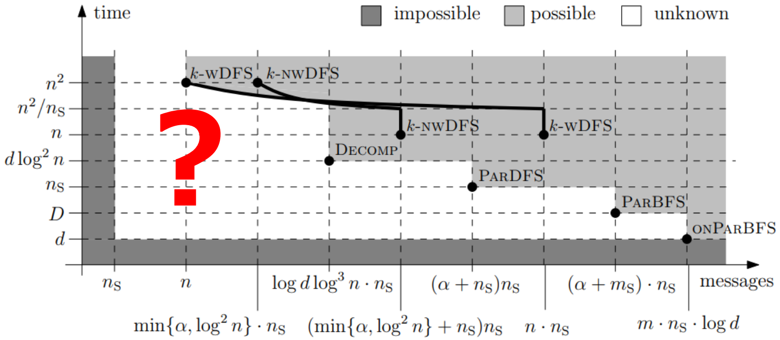
At runtime:

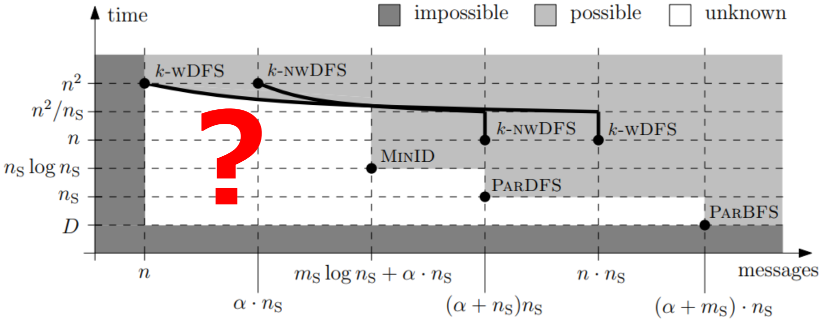
1. Each "event node" *v* sends a ***hello message*** to all its ***neighbors in*** P_v (at most $|P_v| \leq ɑ$ many)
2. Those "event nodes" that receive *hello* messages reply in the second round (at most $|P_v| \leq ɑ$ many)

Since it is a **cover**: each event node either receives a ***hello*** message ***or a reply*** from all neighbors that are event nodes, and thus may effectively ***learn their neighborhood***.

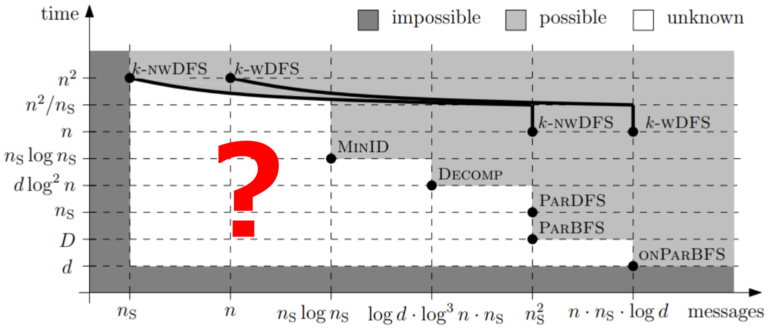
Many Open Problems

- For distributed disaster size detection alone: On-duty, Off-duty, randomized





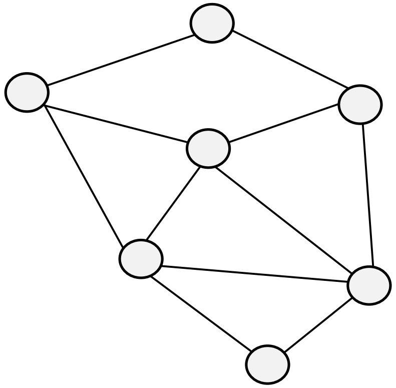
- … but many other problems!



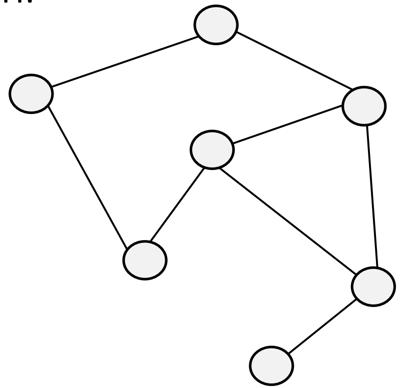
Another Use of Preprocessing: Preparing for Link Failures and "Fast Fixing" (aka. SUPPORTED Model)

- What can we precompute to quickly fix a solution (e.g., coloring, DS, MIS, …) under link failures?
- Problem has two phases and there are two graphs:
  - **Support graph H**: on which one can do ***precomputation***
  - **Input graph G ⊆ H**: on which solution should be computed ***fast***
- Motivation: **Fast fixing**
  - After failures (induce subgraph), want to fix solution quickly

Support graph H:



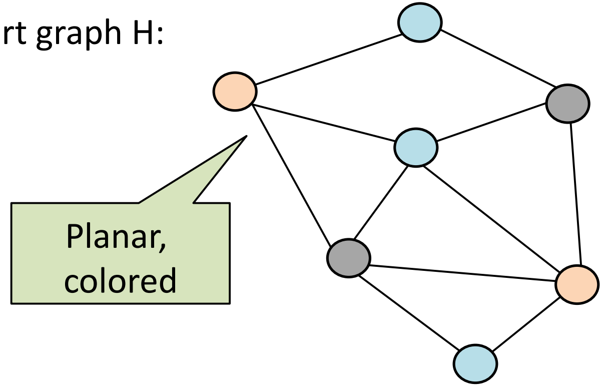
Input graph G ⊆ H:



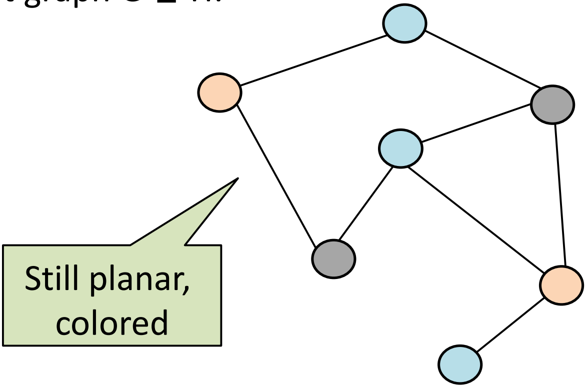
Idea: Exploit Properties That Are Preserved!

- As input graph is subgraph, some **properties remain**:
  - E.g., if it was ***sparse/planar/bounded-genus***… it remains
- Consequence: Legal coloring stays legal coloring
  - Can **precompute** it!
- Case study: Czygrinow et al. algorithm to compute (1+Ɛ)-approx. **MIS** in **non-constant time** in planar graphs
  - First computes pseudo-forest in O(1) time
  - Then performs 3-coloring: allows to find "heavy-stars" of constant diameter

Support graph H:



Input graph G ⊆ H:

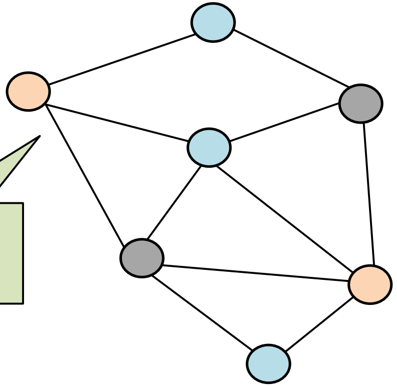


SUPPORTED Model: Some Observations

- As input graph is subgraph, some **properties remain**:
  - E.g., if it was ***sparse/planar/bounded-genus***… it remains
- Consequence: Legal coloring stays legal coloring
  - Can **precompute** it!
- Case study: Czygrinow et al. algorithm to compute (1+Ɛ)-approx. **MIS** in **non-constant time** in planar graphs
  - First computes pseudo-forest in O(1) time
  - Then performs 3-coloring: allows to find “heavy-stars” of constant diameter

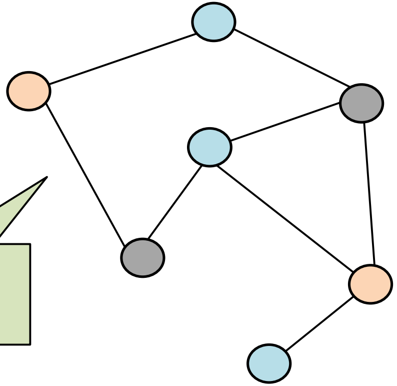
Only non-constant time part. But planar graph 4-colorable (***precomputed***), and can reduce from 4 to 3 colors in **constant time**!

Support graph H:



Planar, colored

Input graph G ⊆ H:



Still planar, colored

Distributed Disaster Detection and SUPPORT: Many Open Questions

- Some ***positive results*** for SUPPORT, e.g.,
  - See above: MIS, MaxMatching, MDS can be computed in constant time in **bounded-genus** graphs in SUPPORTED.
  - Many other examples: **Dominating Set** algorithm by Friedman and Kogan consists of two phases: symmetry breaking (distance-2 coloring) and optimization (greedy). In supported model, both phases in O(1).

- Some **lower bounds** can be generalized: (maybe suprising) ***limits*** on what can be achieved with precomputation

- Everything else pretty much ***open***

Further Reading

- Online Aggregation of the Forwarding Information Base: Accounting for Locality and Churn
  Marcin Bienkowski, Nadi Sarrar, Stefan Schmid, and Steve Uhlig.
  IEEE/ACM Transactions on Networking (**TON**), 2018.
- Exploiting Locality in Distributed SDN Control
  Stefan Schmid and Jukka Suomela.
  ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**), Hong Kong, China, August 2013.

Opportunity: Topology Control for Data-Intensive Networks?

Traditional Networks

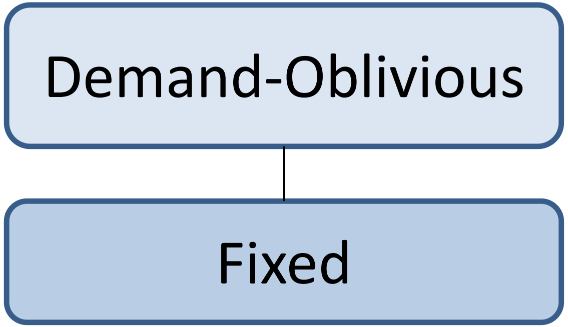
- Usually optimized for the "worst-case" (**all-to-all** communication)
- Typical criteria:
  - Classic network design: small ***degree***, small ***diameter***, high ***mincut***
  - Topology control: short routes, contains ***min energy path***
- Lower bounds and hard **trade-offs**, e.g., degree vs diameter

Traditional Networks

- Usually optimized for the “worst-case” (**all-to-all** communication)

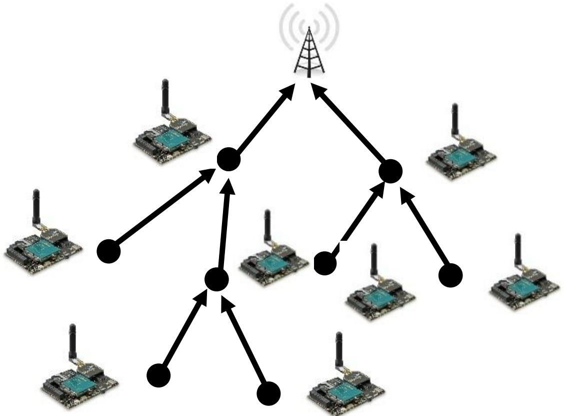
- Typical criteria:
    - Classic network design: small ***degree***, small ***diameter***, high ***mincut***
    - Topology control: short routes, contains ***min energy path***

- Lower bounds and hard **trade-offs**, e.g., degree vs diameter



Data-Intensive Networks

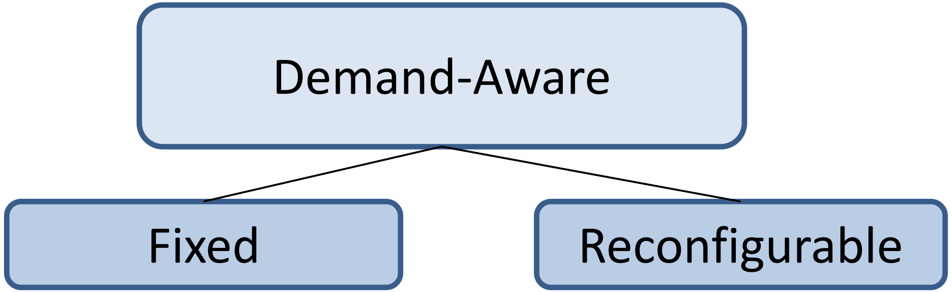
- Recall: not only size of networks grows but also ***amount of data***
- At the same time, traffic often has ***specific patterns*** and is far from all-to-all
  - E.g., all-to-one: **sink node** collects results
  - Also in **datacenter**: ***sparse***



Can we make networks more data/demand-aware?

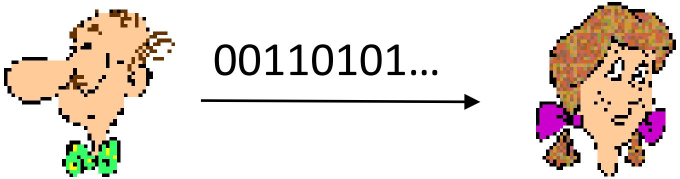
Demand-Aware Networks: 2 Flavors

- ***DAN***: Demand-Aware Network
  - Statically optimized **toward the demand**
- ***SAN***: Self-Adjusting Network
  - **Dynamically optimized toward** the (time-varying) demand

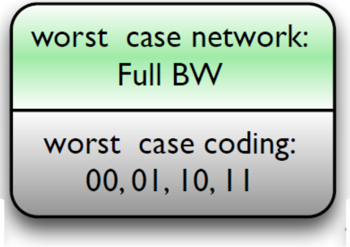


An Analogy to Coding

‚Coming to ALGOSENSORS?‘



structure: static / future demand: unknown

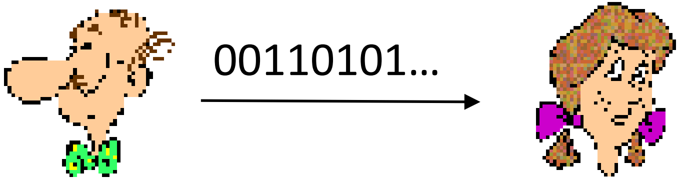


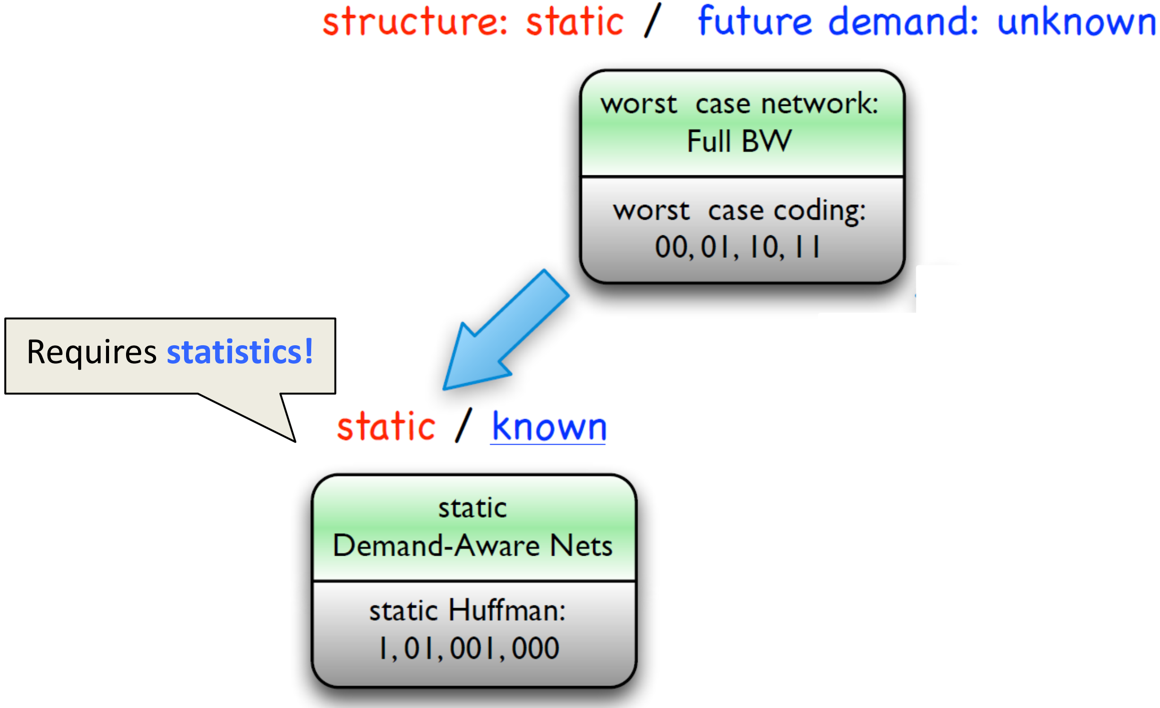
worst case network:
Full BW

worst case coding:
00, 01, 10, 11

An Analogy to Coding

‚Coming to ALGOSENSORS?‘





structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

Requires **statistics!**

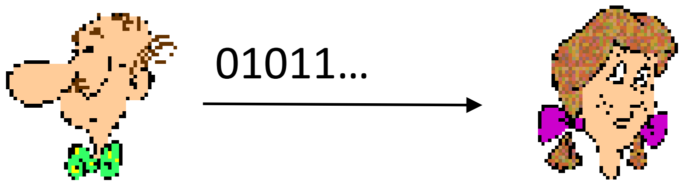
static / known

static
Demand-Aware Nets

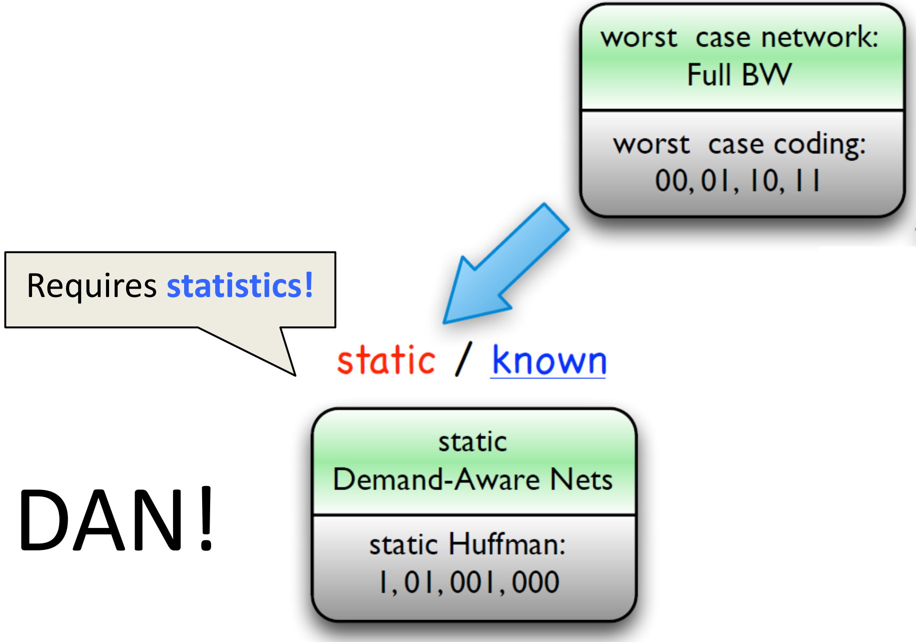
static Huffman:
1, 01, 001, 000

An Analogy to Coding

‚Coming to ALGOSENSORS?‘



structure: static / future demand: unknown



worst case network:
Full BW

worst case coding:
00, 01, 10, 11

Requires **statistics!**

static / known

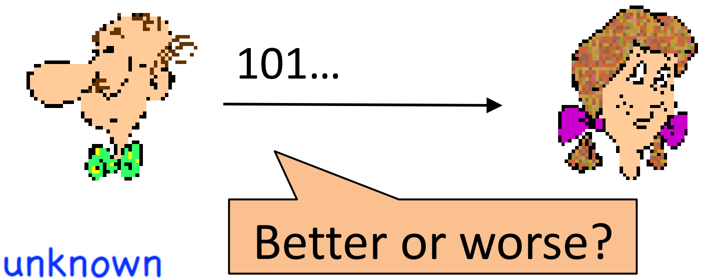
static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

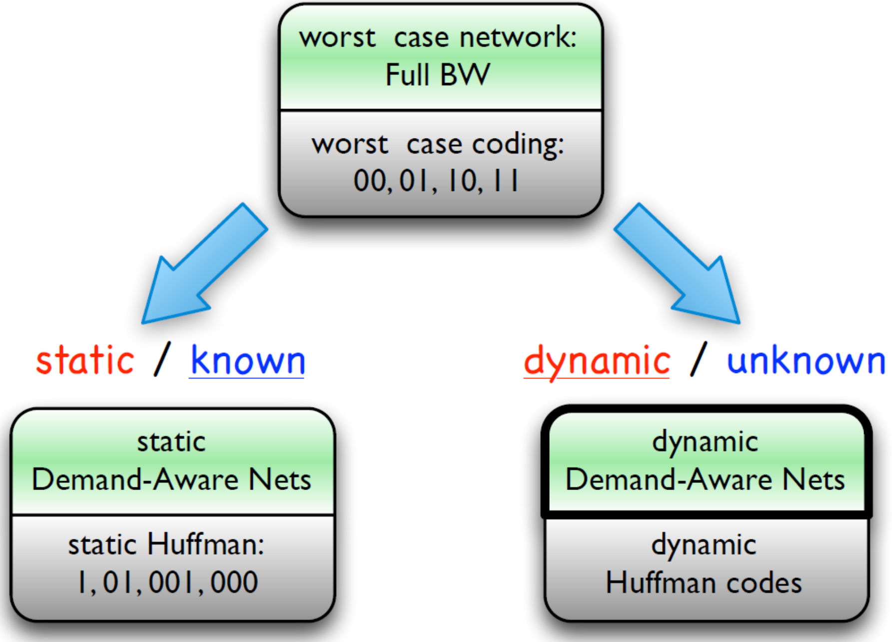
DAN!

An Analogy to Coding

‚Coming to ALGOSENSORS?‘



structure: static / future demand: unknown



worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

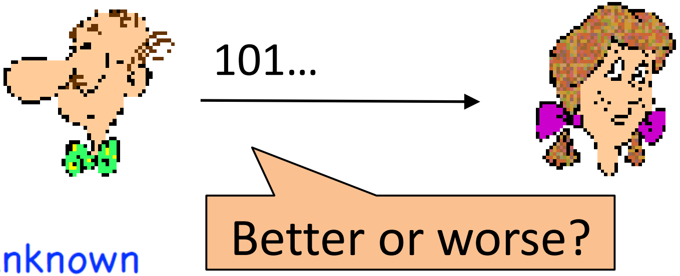
dynamic / unknown

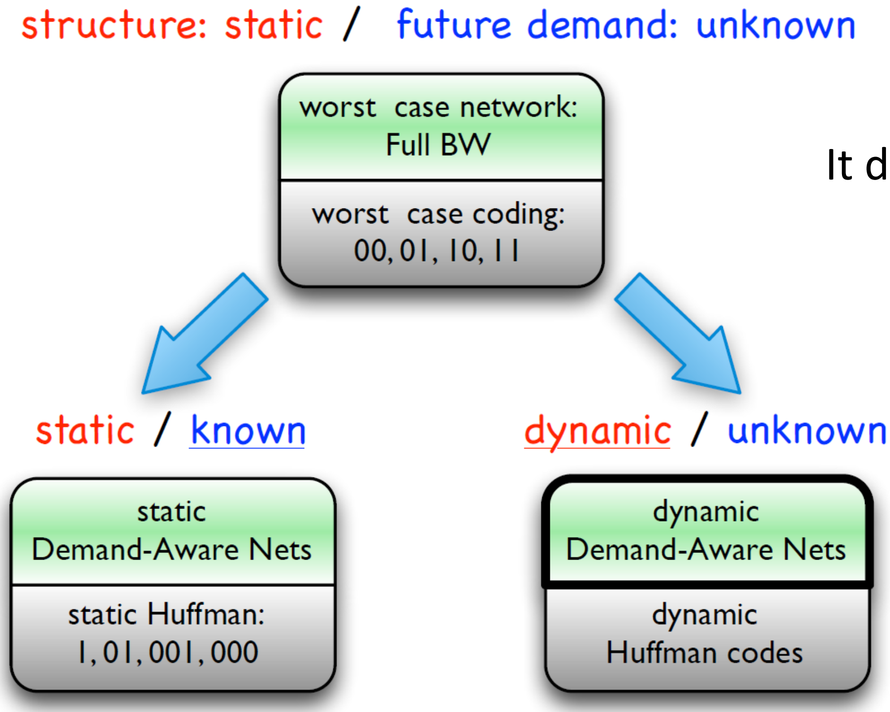
dynamic
Demand-Aware Nets

dynamic
Huffman codes

An Analogy to Coding

‚Coming to ALGOSENSORS?‘





structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

dynamic / unknown

dynamic
Demand-Aware Nets

dynamic
Huffman codes

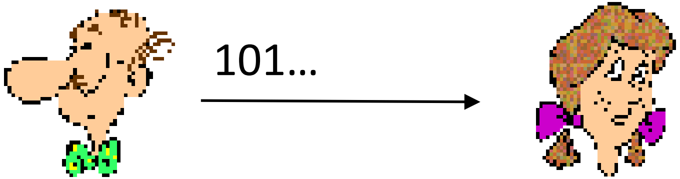
It depends:

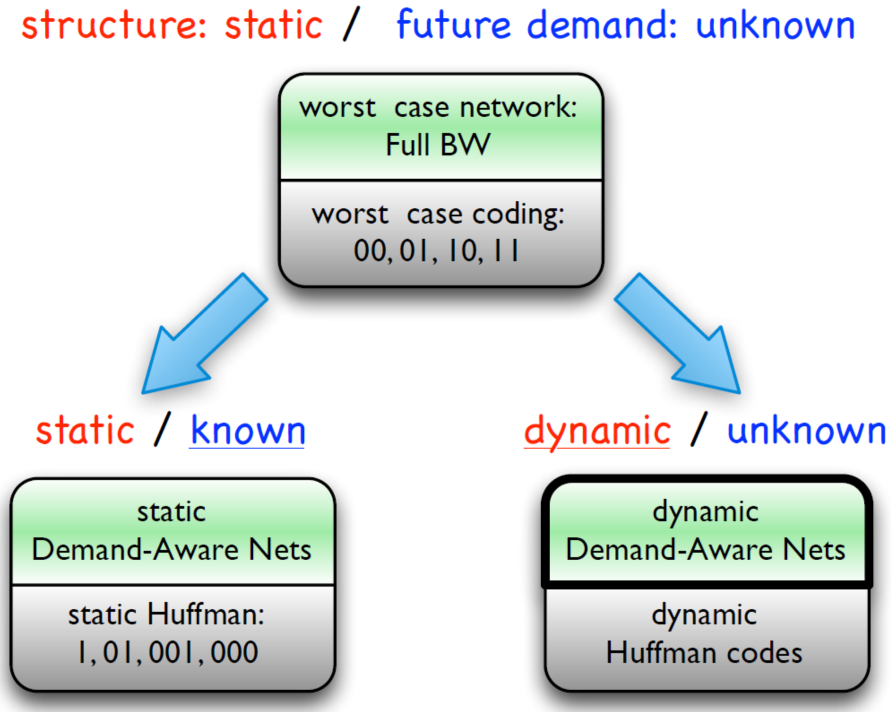
👍 Can exploit ***temporal locality***!

But: No statistics: online!

An Analogy to Coding

‚Coming to ALGOSENSORS?‘





structure: static / future demand: unknown

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

static / known

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

dynamic / unknown

dynamic
Demand-Aware Nets

dynamic
Huffman codes

DAN!

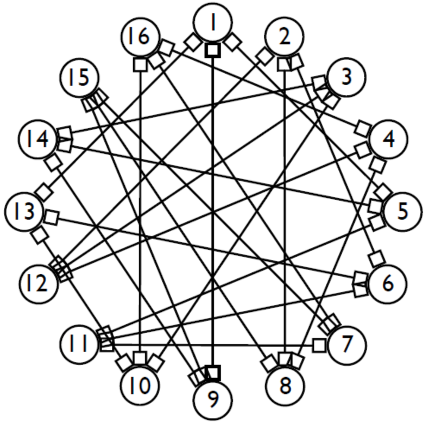
Can exploit *spatial locality*!

SAN!

Additionally exploit *temporal locality*!

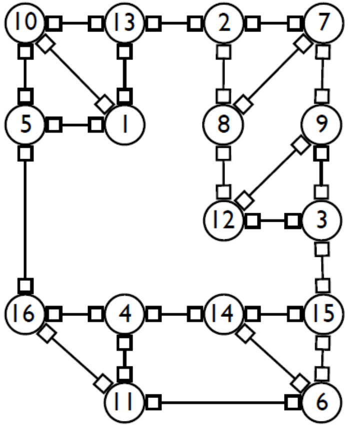
Spectrum of Flexible Networks

Oblivious



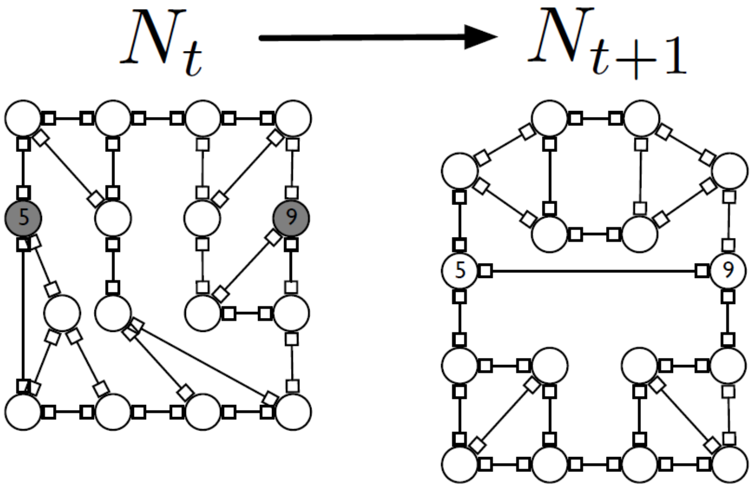
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**: Route lengths ***depend on demand***

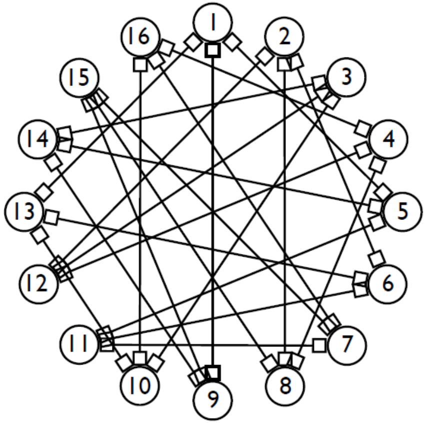
SAN



Exploit **temporal locality** as well

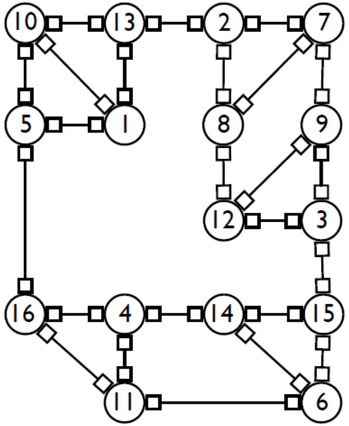
Spectrum of Flexible Networks

Oblivious



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

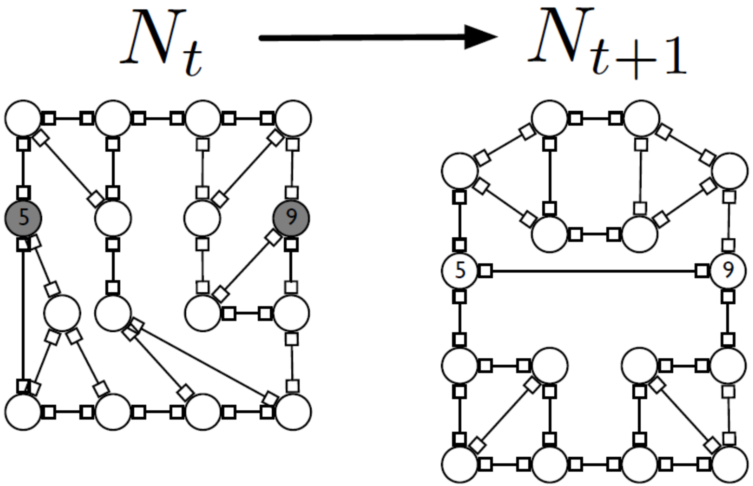
DAN



Exploit **spatial locality**: Route lengths ***depend on demand***

Good metric?

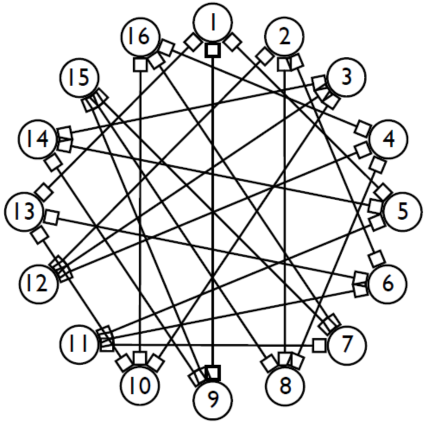
SAN



Exploit **temporal locality** as well

Spectrum of Flexible Networks

Oblivious

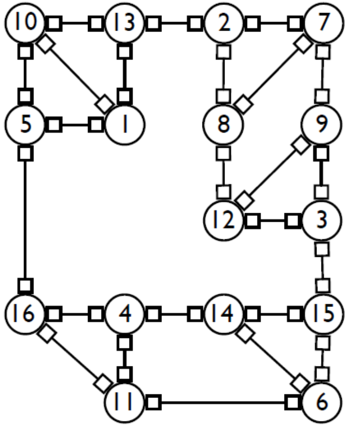


Const degree

(e.g., **expander**):

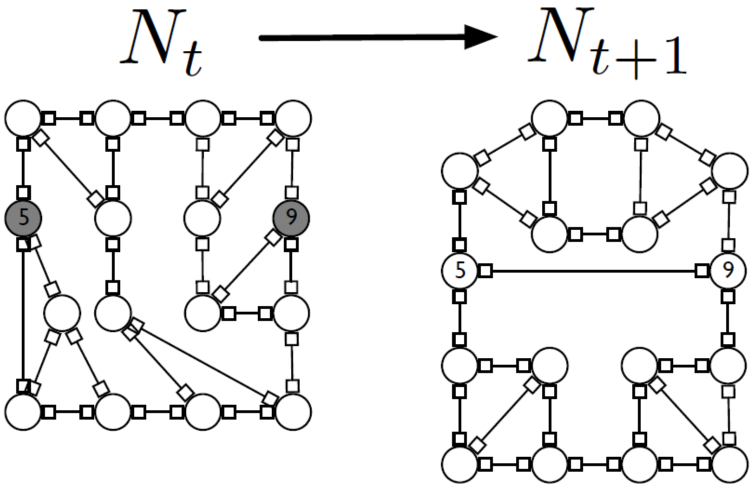
route lengths ***O(log n)***

DAN



Exploit **spatial locality**: Route lengths ***depend on demand***

SAN



Exploit **temporal locality** as well

Good metric?

e.g., ***entropy***!

Example of A DAN Design Problem

Input: Workload

Destinations

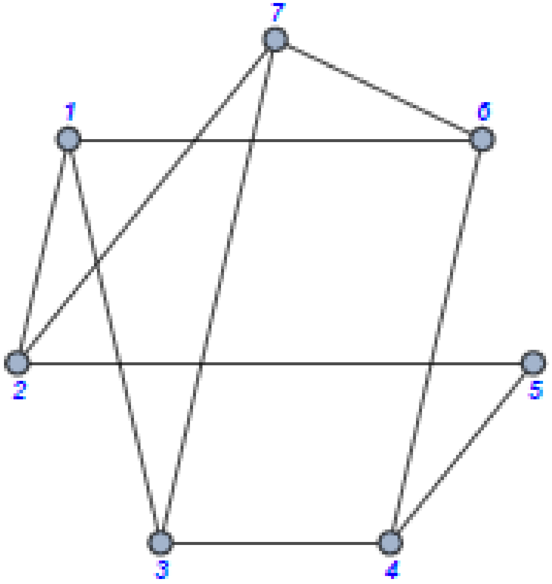
Sources

$$
\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0 \\
\hline
\end{array}
$$

Demand matrix: joint distribution

design

Output: DAN



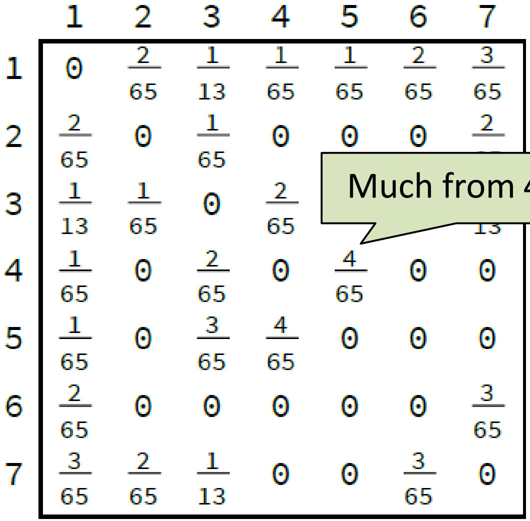
… of constant degree (**scalability**)

Example of A DAN Design Problem

Input: Workload

Destinations

Sources

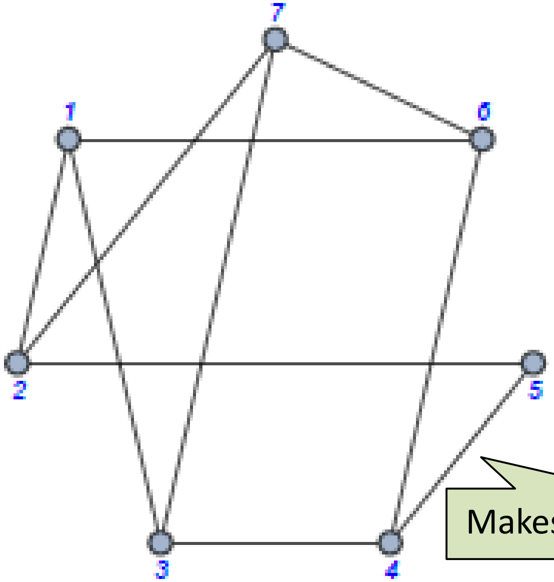


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	2
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$			13
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Much from 4 to 5.

design

Output: DAN



Makes sense to add link!

Demand matrix: joint distribution

… of constant degree (**scalability**)

Example of A DAN Design Problem

Input: Workload

Destinations

1 communicates to many.

Sources

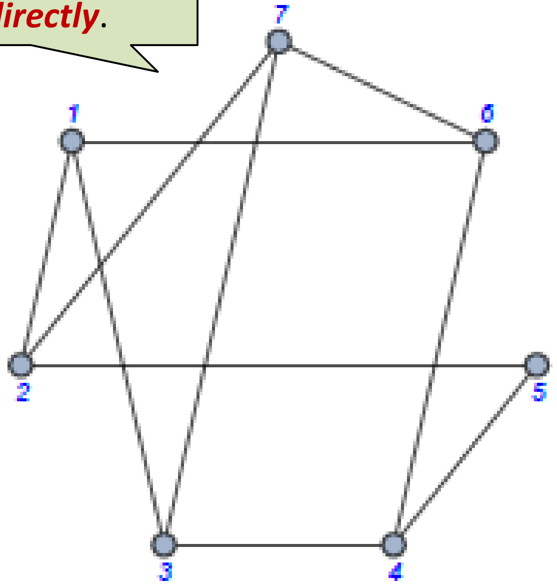
$$\begin{array}{c|ccccccc|}
 & & & & & & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0 \\
\hline
\end{array}$$

Demand matrix: joint distribution

design

Output: DAN

Bounded degree: route to 7 ***indirectly***.



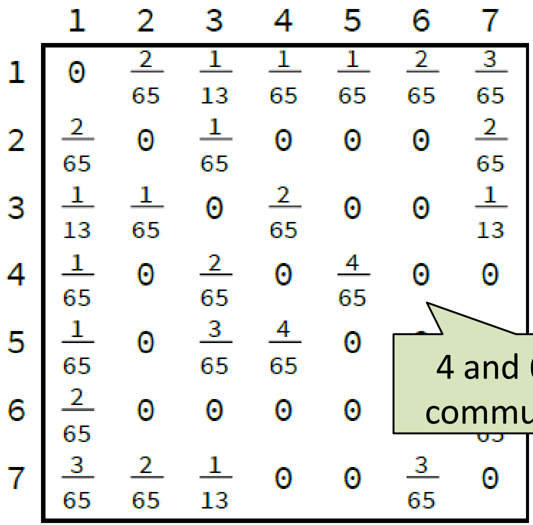
… of constant degree (**scalability**)

Example of A DAN Design Problem

Input: Workload

Destinations

Sources

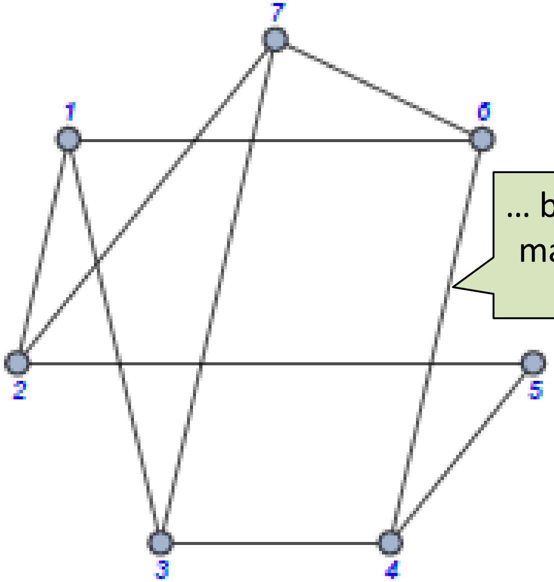


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	[illegible]	[illegible]
6	$\frac{2}{65}$	0	0	0	0	[illegible]	[illegible]
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

4 and 6 don’t communicate…

design

Output: DAN



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

Demand matrix: joint distribution

… of constant degree (**scalability**)

More Formally: DAN Design Problem

Input:

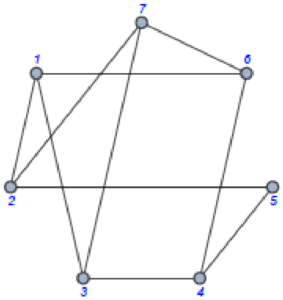
$\mathcal{D}[p(i,j)]$: joint distribution, **Δ**

$$X \quad \overset{Y}{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\ 2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\ 3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\ 4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\ 5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\ 6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\ 7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0 \end{array}}$$

$\Rightarrow$

Output:

N: DAN



Bounded degree Δ=3

Expected Path Length (EPL): Basic measure of efficiency

Path length ***on DAN***.

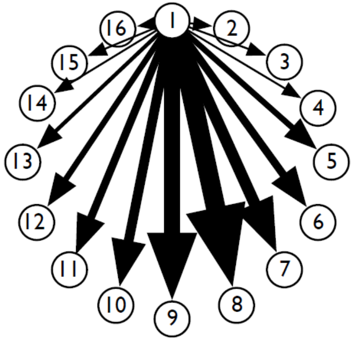
$$\mathrm{EPL}(\mathcal{D},N) = E_{\mathcal{D}}[d_N(\cdot,\cdot)] = \sum_{(u,v)\in\mathcal{D}} p(u,v)\cdot d_N(u,v)$$

Frequency

Good Metrics for DANs?

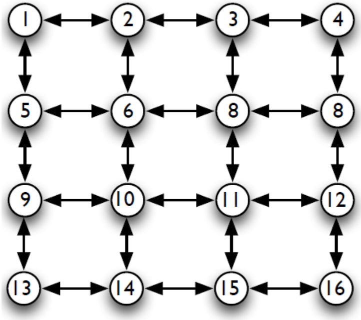
- Clique communication (**all-to-all**) is hard: nothing to exploit
- But what about such traffic patterns:

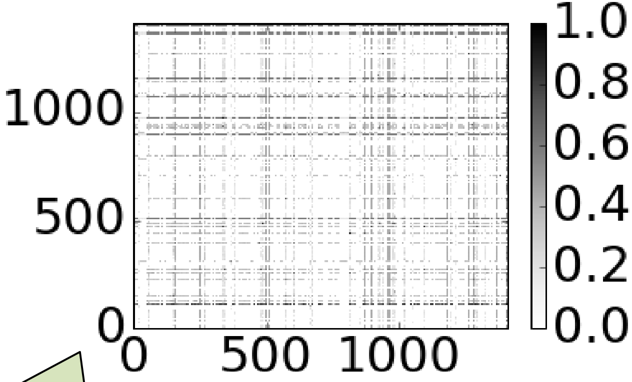
Skewed!



or

Structure!



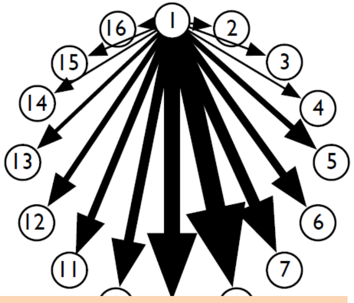


Spatial locality!

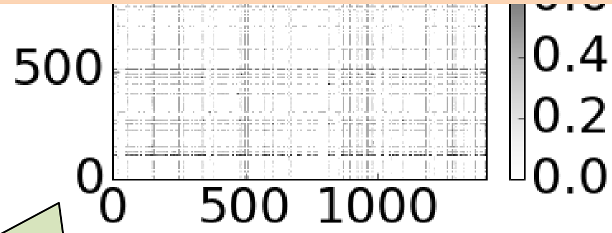
Good Metrics for DANs?

- Clique communication (**all-to-all**) is hard: nothing to exploit
- But what about such traffic patterns:

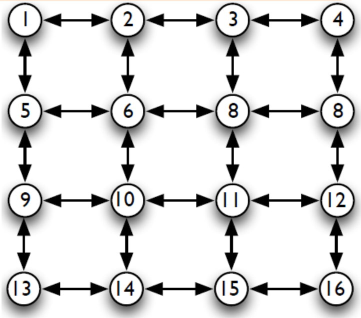
Skewed!



These traffic matrices have low **entropy**: allows for excellent demand-aware networks!



Spatial locality!



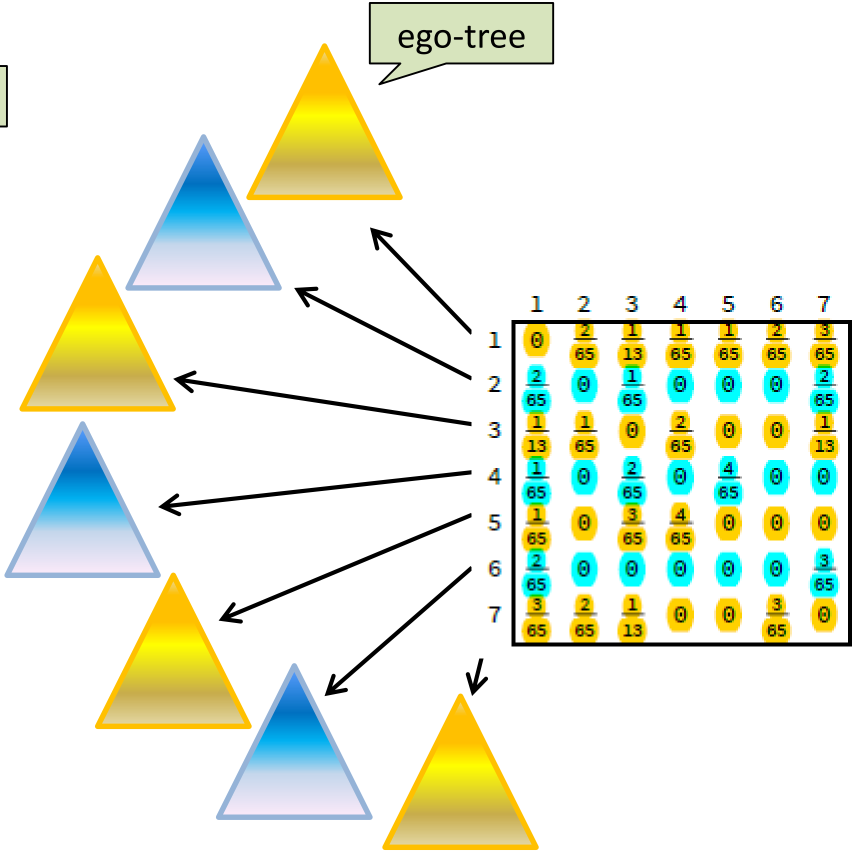
Structure!

Indeed: Entropy is a Lower Bound (Sources)

sources

destinations

- **Proof idea** (EPL=$\Omega(H_\Delta(Y|X))$):
- Consider ***union*** of all **ego-trees**
- Violates ***degree restriction*** but valid lower bound

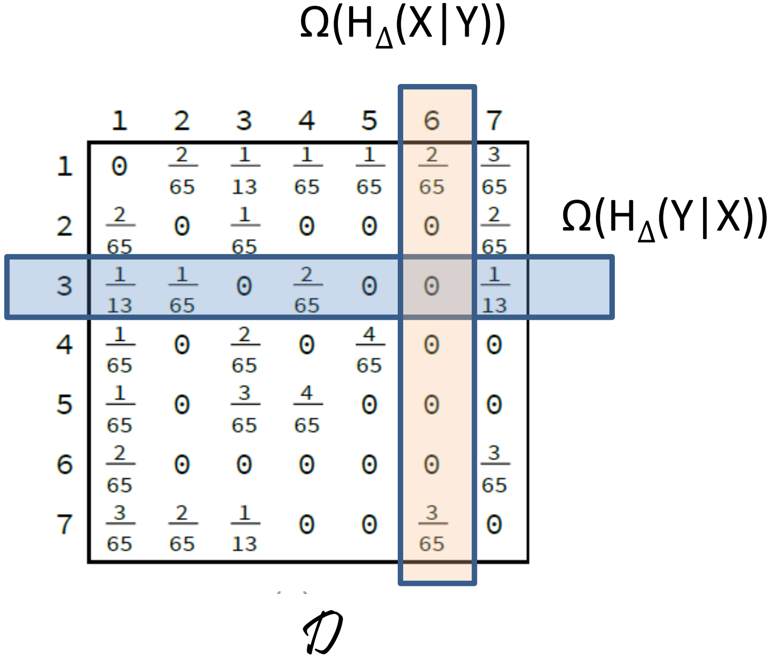


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Lower Bound: Sources + Destinations

Do this in **both dimensions**:

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



$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & \frac{2}{65} & \frac{1}{13} & \frac{1}{65} & \frac{1}{65} & \frac{2}{65} & \frac{3}{65} \\
2 & \frac{2}{65} & 0 & \frac{1}{65} & 0 & 0 & 0 & \frac{2}{65} \\
3 & \frac{1}{13} & \frac{1}{65} & 0 & \frac{2}{65} & 0 & 0 & \frac{1}{13} \\
4 & \frac{1}{65} & 0 & \frac{2}{65} & 0 & \frac{4}{65} & 0 & 0 \\
5 & \frac{1}{65} & 0 & \frac{3}{65} & \frac{4}{65} & 0 & 0 & 0 \\
6 & \frac{2}{65} & 0 & 0 & 0 & 0 & 0 & \frac{3}{65} \\
7 & \frac{3}{65} & \frac{2}{65} & \frac{1}{13} & 0 & 0 & \frac{3}{65} & 0 \\
\hline
\end{array}$$

Problem Related To…:

- Sparse, distance-preserving (low-distortion) **spanners**

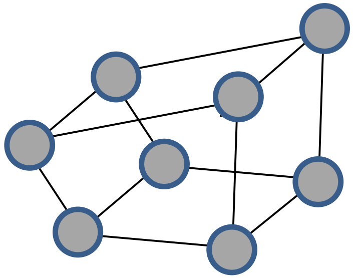
- But:
  - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
  - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
  - We require ***constant degree***

- Nevertheless: we can sometimes ***leverage the connection*** to spanners!

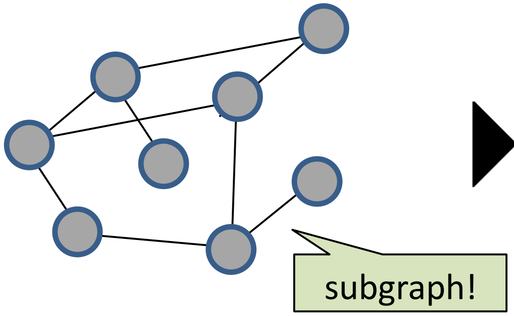
Leveraging The Connection to Spanners

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

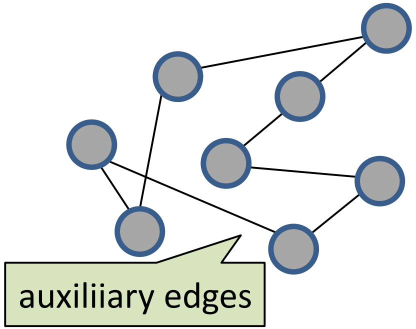
r-regular *and uniform* **request***:*



Sparse, ***irregular*** ***(constant)*** **spanner***:*



Constant *degree* ***optimal*** **DAN** (EPL at most ***log r***):



Leveraging The Conn rs

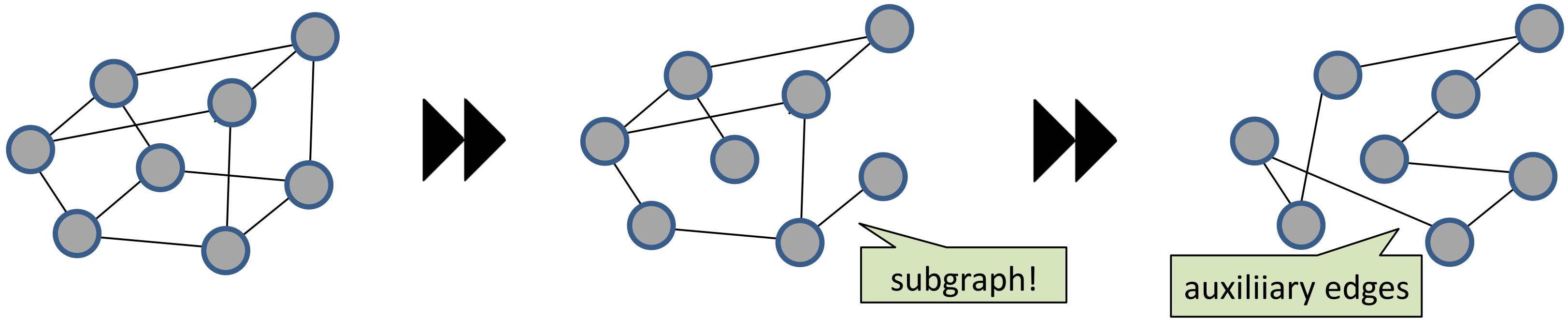
By taking the union of "ego-trees" and balance degrees.

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

r-regular *and uniform* **request***:*

Sparse, ***irregular (constant)*** **spanner***:*

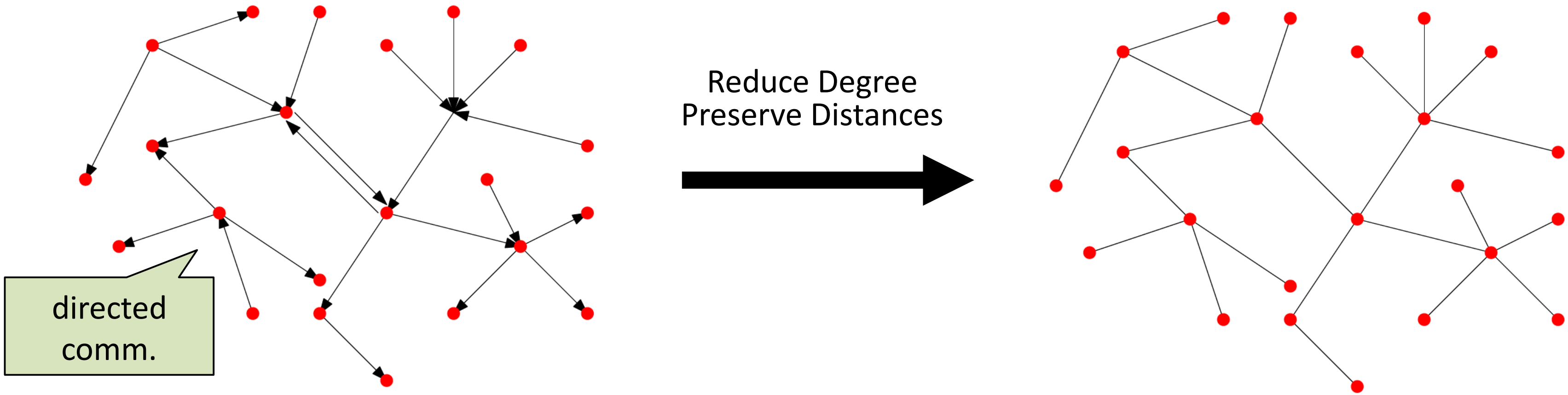
Constant *degree* ***optimal*** **DAN** (EPL at most ***log r***):



Proof Intuition: How to Balance Degrees

Example: Tree Distributions

- Basic idea to get from irregular spanner to constant-degree tree of low distortion:

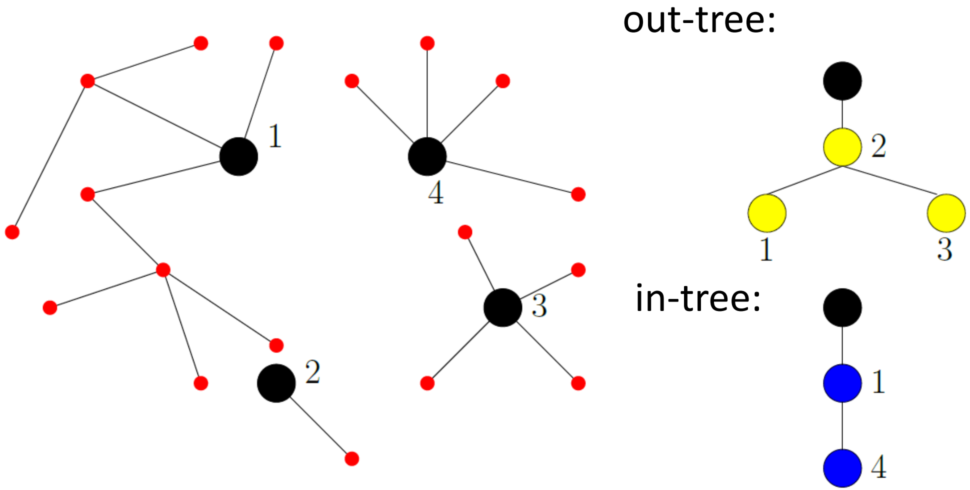


directed comm.

Reduce Degree
Preserve Distances

Proof Idea: Construct Huffman Trees

- Make tree **rooted** and directed: gives **parent-child** relationship
- Arrange the outgoing edges (to children) of each node in a **binary (Huffman) tree**
- Repeat for the incoming edges: make another **another binary (Huffman) tree** with incoming edges from children
- Analysis
  - Can ***appear in at most 4 trees***: in&out own tree and in&out tree of parent (parent-child helps to avoid many "children trees")
  - ***Degree*** at most 4*3=12
  - Huffman trees maintain ***distortion***: proportional to ***conditional entropy***



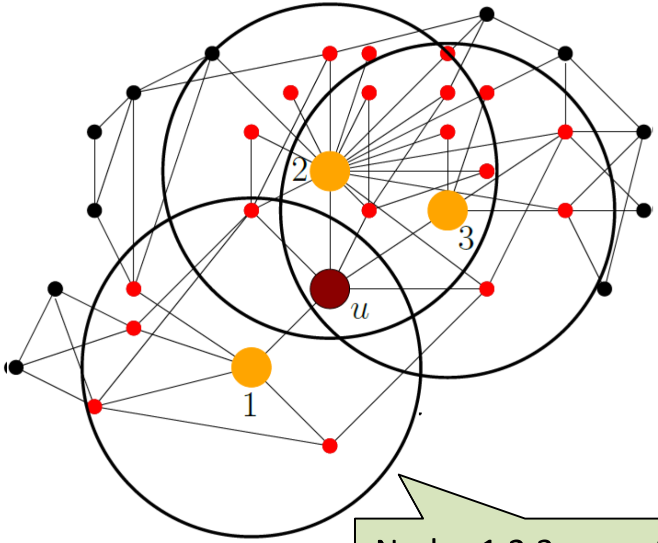
Example: Constant-Sparse Spanner for Demands of Locally-Bounded Doubling Dimension

- **LDD**: $G_{\mathcal{D}}$ has a **Locally-bounded Doubling Dimension** (LDD) iff all 2-hop neighbors are covered by 1-hop neighbors of just λ nodes
  - Note: care only about ***2-neighborhood***

We only consider 2 hops!

- Formally, $B(u, 2) \subseteq \bigcup_{i=1}^{\lambda} B(v_i, 1)$

- Note: LDD graph can still be of ***high degree***!



Nodes 1,2,3 cover 2-hop neighborhood of u.

DAN for Locally-Bounded Doubling Dimension

Lemma: There exists a sparse 9-(subgraph)spanner for LDD.

This ***implies optimal DAN***: still focus on regular and uniform!

Def. (ε-net): A subset V' of V is a **ε-net** for a graph G = (V,E) if

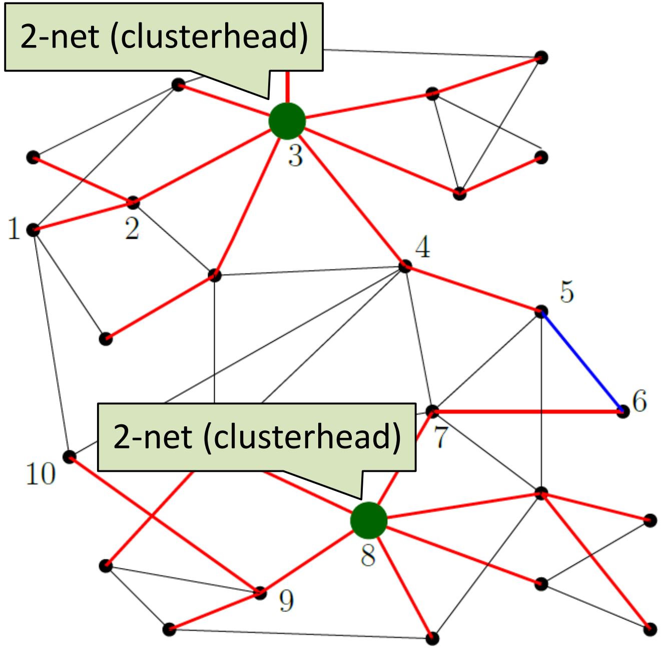
- V' sufficiently "independent": for every u, v ∈ V', $d_G(u, v) > \varepsilon$
- "dominating" V: for each w ∈ V , ∃ at least one u ∈ V' such that, $d_G(u,w) \leq \varepsilon$

9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

Easy: Select nodes into 2-net **one-by-one** in decreasing (remaining) degrees, **remove 2-neighborhood**. Iterate.

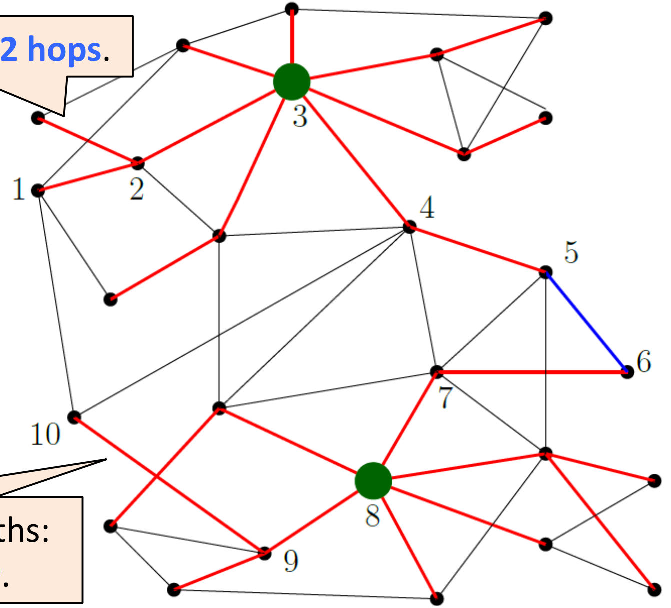


9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Assign nodes to one of the closest 2-net nodes



Assign: at most **2 hops**.

Union of these shortest paths: a **forest**. Add to **spanner**.

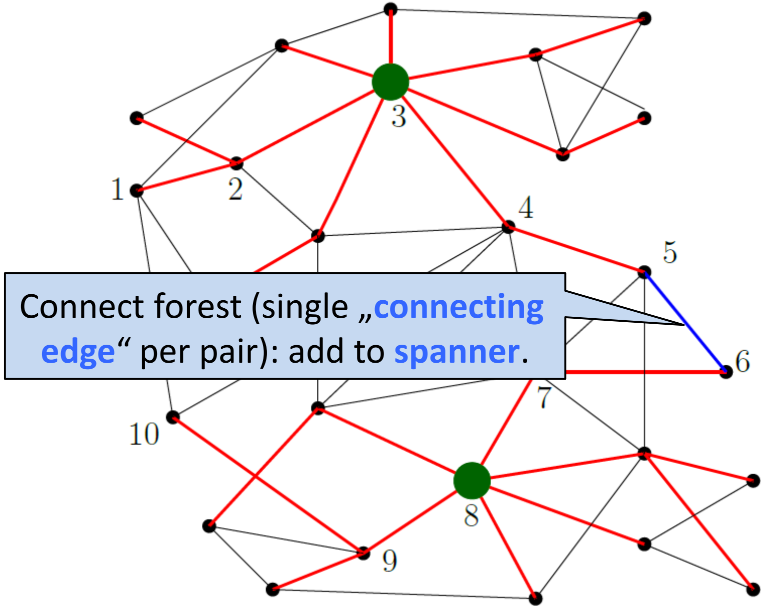
9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Assign nodes to one of the closest 2-net nodes

3. Join two clusters if there are edges in between



Connect forest (single „**connecting edge**“ per pair): add to **spanner**.

9-Spanner for LDD (= optimal DAN)

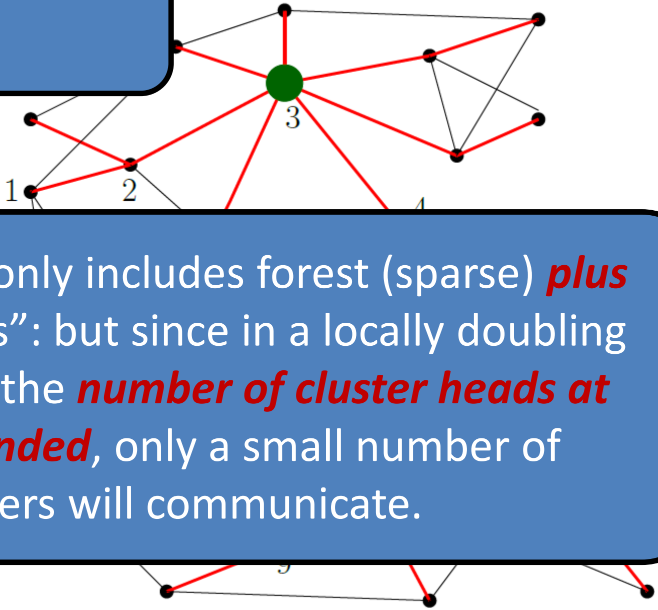
Distortion 9: Detour via clusterheads and bridge: from u to v via u,ch(u),x,y,ch(v),v

1. Find a 2-net

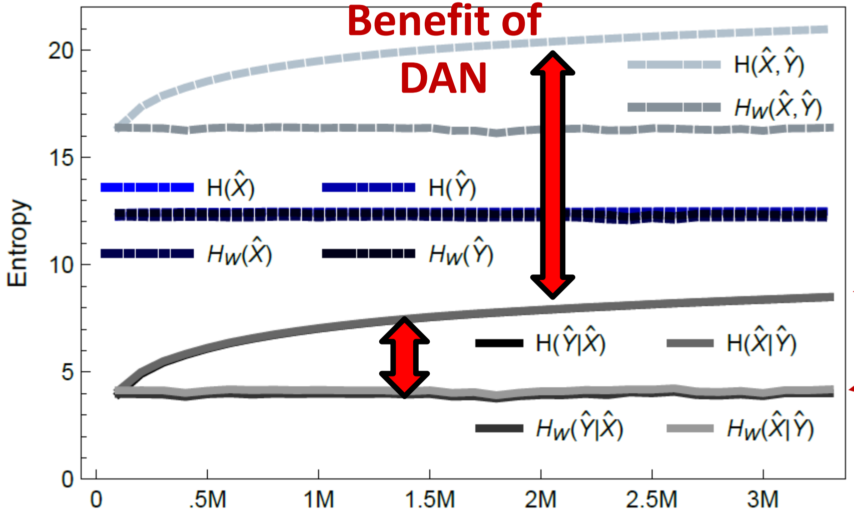
2. Assign nodes to one of the closest 2-net node

3. Join two clusters edges in between

Sparse: Spanner only includes forest (sparse) ***plus*** "connecting edges": but since in a locally doubling dimension graph the ***number of cluster heads at distance 5 is bounded***, only a small number of neighboring clusters will communicate.



Flexibility: How Often to Reconfigure?



Benefit of DAN

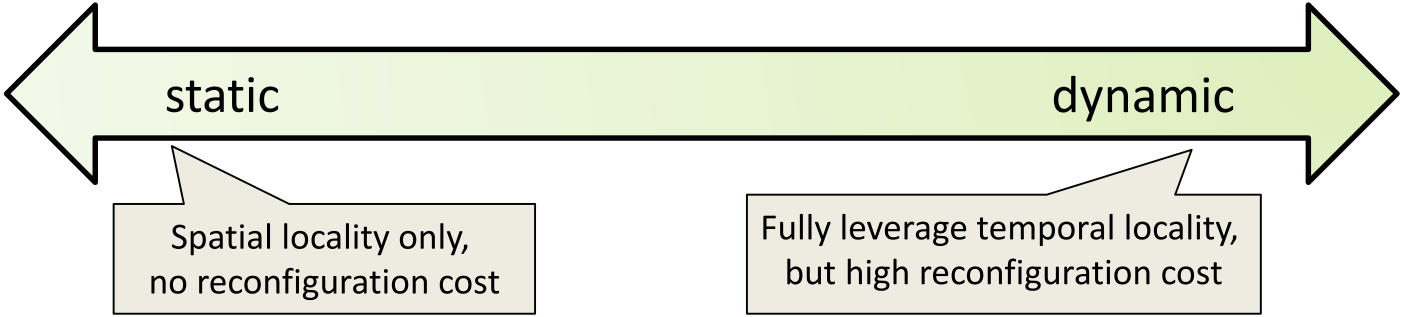
Benefit of SAN

(if we change every 100k requests)

Oblivious design: proportional to ***joint entropy***

DAN: proportional to ***conditional entropy***

SAN: proportional to ***conditional entropy in time windows W***



static

dynamic

Spatial locality only,
no reconfiguration cost

Fully leverage temporal locality,
but high reconfiguration cost

What About Distributed Algos?

- What about **geometric** demand-aware graphs to connect robots in a plane?
- What about **distributed** algorithms for self-adjusting networks?
- Can we additionally provide **self-stabilizing** properties?
- ***A (general?) technique:*** each node repeatedly computes "correct graph" **on neighbors** only: seems to be powerful but open question…

Example: Delaunay Graph

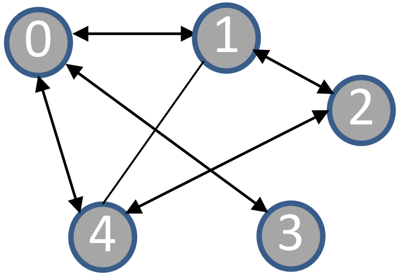
Slightly simplified:

Distributed Algo

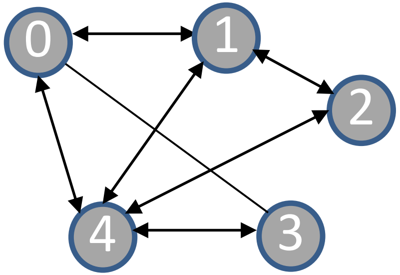
1. Compute local Delaunay
2. Goto 1.

Converges to global Delaunay graph!

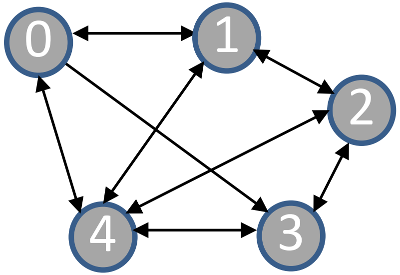
(0)



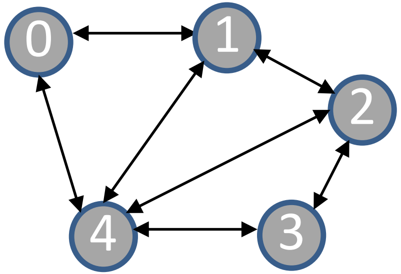
(1)



(2)



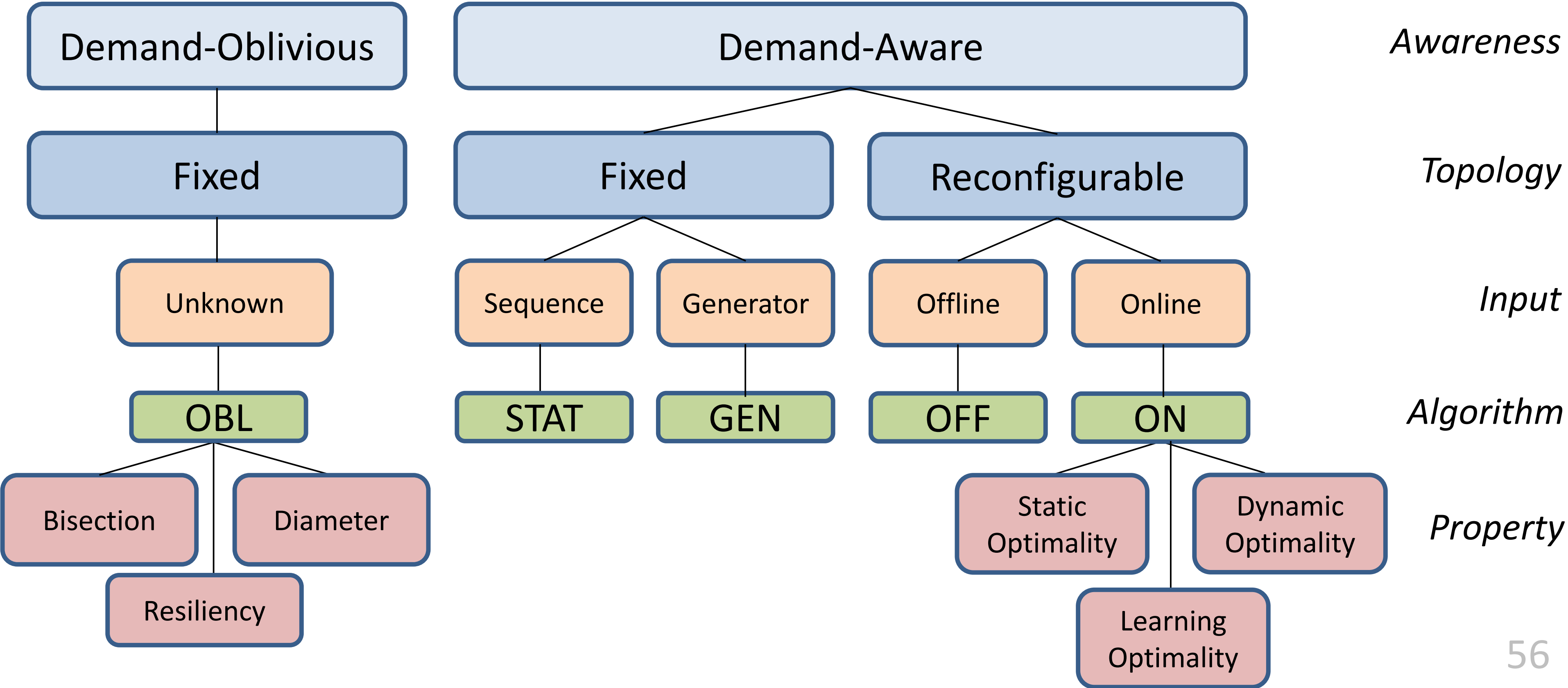
(3)



How to generalize to DANs?

Taxonomy

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **ArXiv** 2018.



Further Reading

- Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks
  Chen Avin and Stefan Schmid.
  ArXiv Technical Report, July 2018.

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

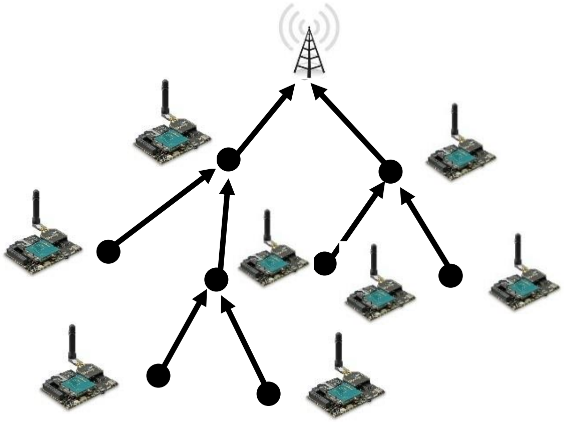
- SplayNet: Towards Locally Self-Adjusting Networks
  Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
  IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.

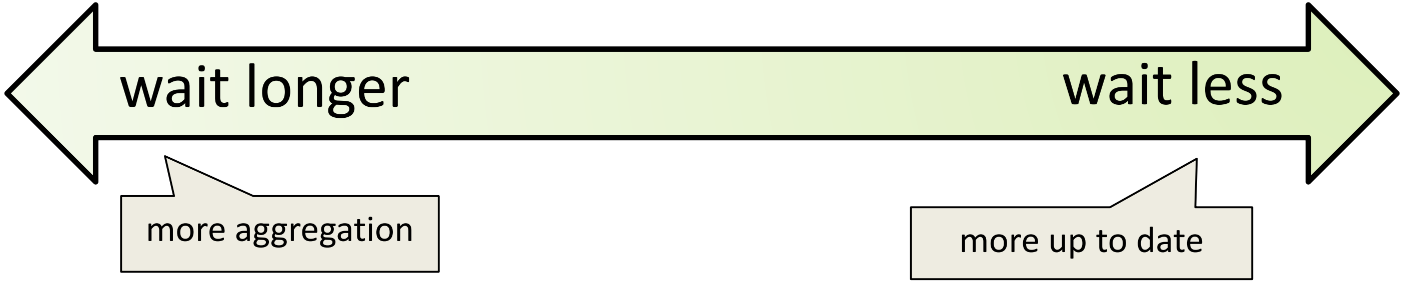
- Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
  Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
  ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.

Opportunity: Scheduling

Many Networks Are Delay-Tolerant: Introduces Flexibility "Wait or Proceed"?

e.g., online **aggregation**:





wait longer

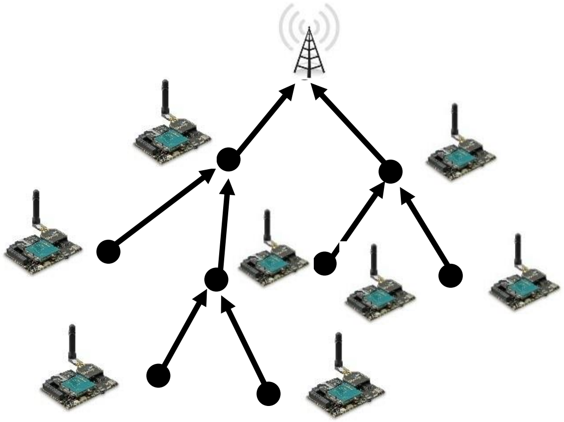
wait less

more aggregation

more up to date

Many Networks Are Delay-Tolerant: Introduces Flexibility “Wait or Proceed”?

e.g., online **aggregation**:



Little known today about ***scheduling in delay tolerant networks***.

wait longer

wait less

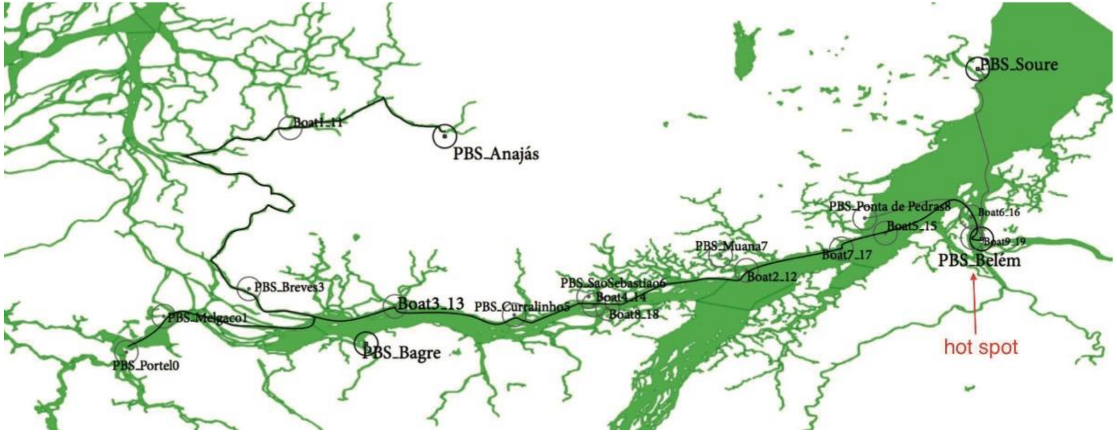
more aggregation

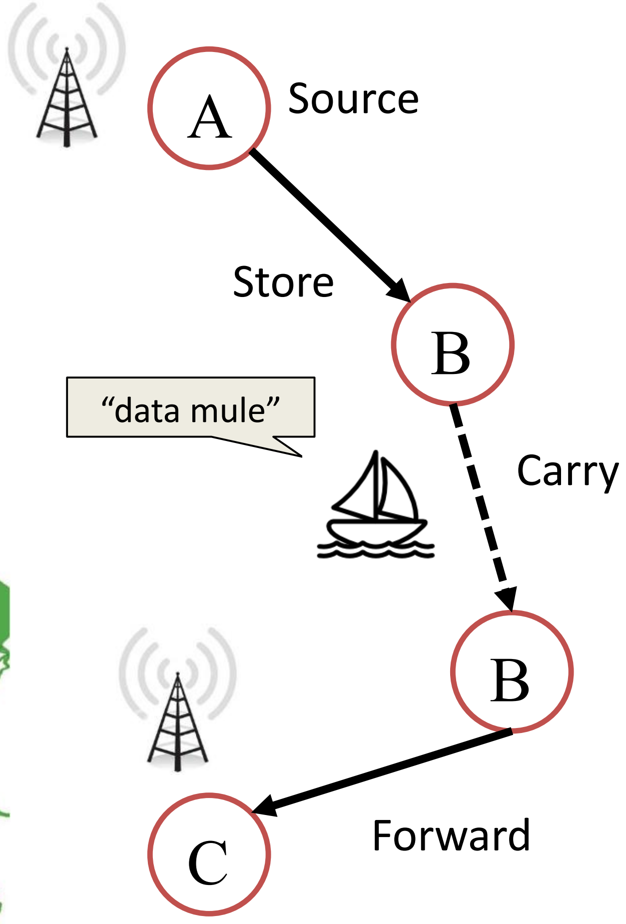
more up to date

Scheduling in DTNs: Back to Amazon Delta!

- Recall: **mobile smart devices** with limited opportunities for transfer

- E.g., **Amazon** delta riverine:





Source

Store

"data mule"

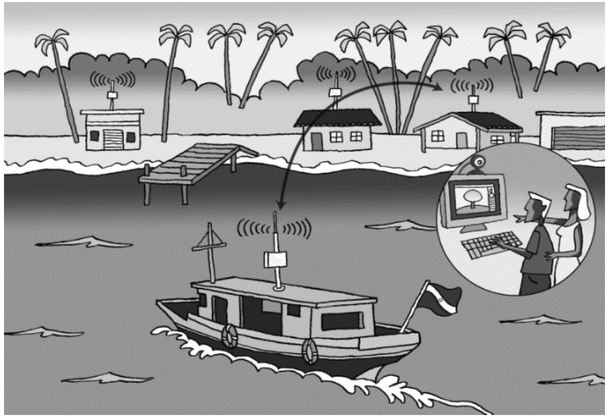
Carry

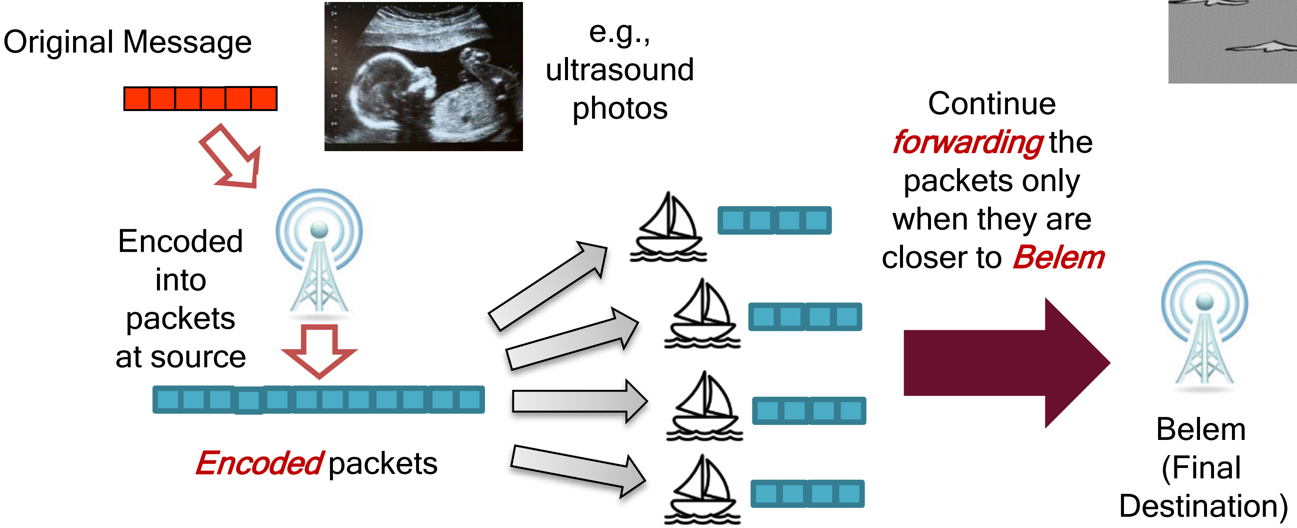
Forward

Destination

A Simple Model: Time-Schedule Networks

- Assume: boats have a fixed time schedule (**Time Scheduled Networks**)
- Idea: transmit packets to nearby boats (according to schedule):





Original Message

e.g., ultrasound photos

Encoded into packets at source

Encoded packets

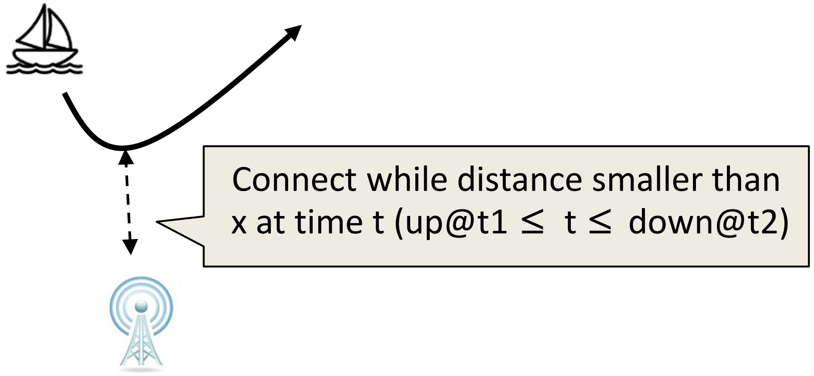
Continue *forwarding* the packets only when they are closer to *Belem*

Belem (Final Destination)

Transmitted to boats that stop by source

Goal (e.g.): ***maximize throughput over DTN***

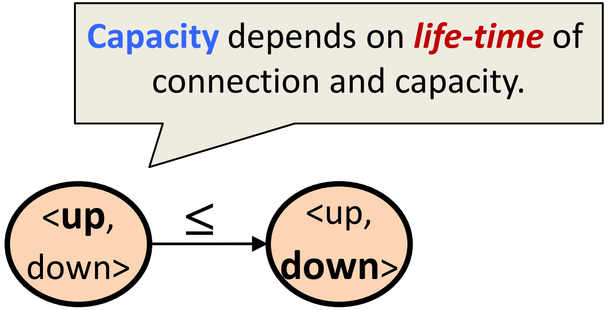
Model can be transformed…



Connect while distance smaller than x at time t (up@t1 ≤ t ≤ down@t2)

- *m* **moving nodes** : has schedule P_i
- *n* **stationary nodes**
- *k* **commodities** between (possibly different sources and destinations)

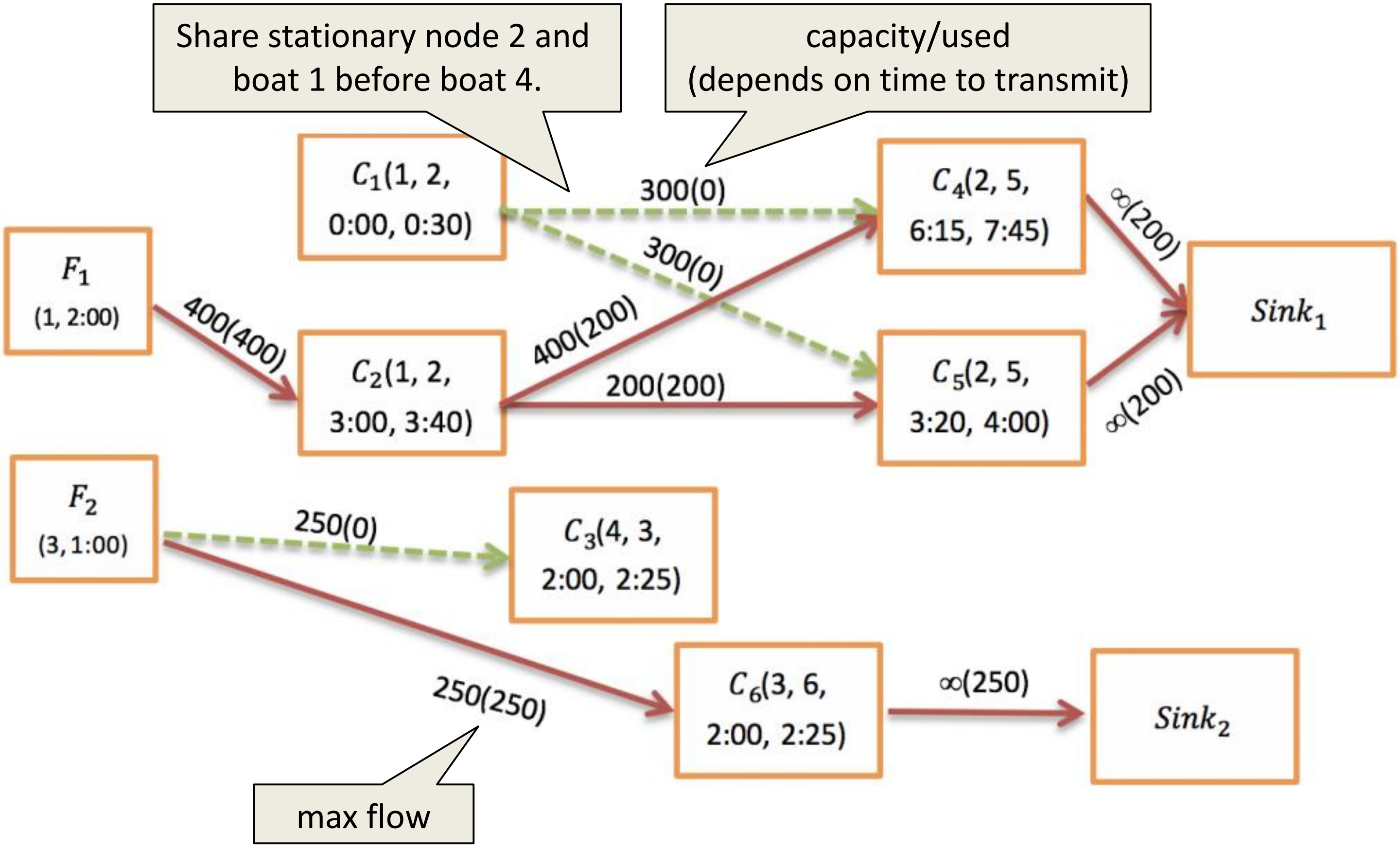
… into a connection graph and MCF problem:



Capacity depends on ***life-time*** of connection and capacity.

- Vertices:
  - moving and stationary nodes **plus** commodity sources **plus** connection nodes
- Directed edges:
  - From connection node C_x to connection node C_y if they ***share a common object and*** $Up_x \leq Down_y$

Connection Graph Example



Share stationary node 2 and boat 1 before boat 4.

capacity/used
(depends on time to transmit)

max flow

All-or-Nothing (Splittable) Multicommodity Flow (ANF)

- Observation:

valid **multicommodity** flow in the connection graph

⟺

feasible routing **schedule** in the DTN network

- Maximum throughput corresponds to All-or-Nothing (Splittable) Multicommodity Flow (**ADF**)

Relaxed version of Maximum Edge Disjoint Path (MEDP) problem: ***fractional***.
Find max subset of commodities that can be simultaneously routed.

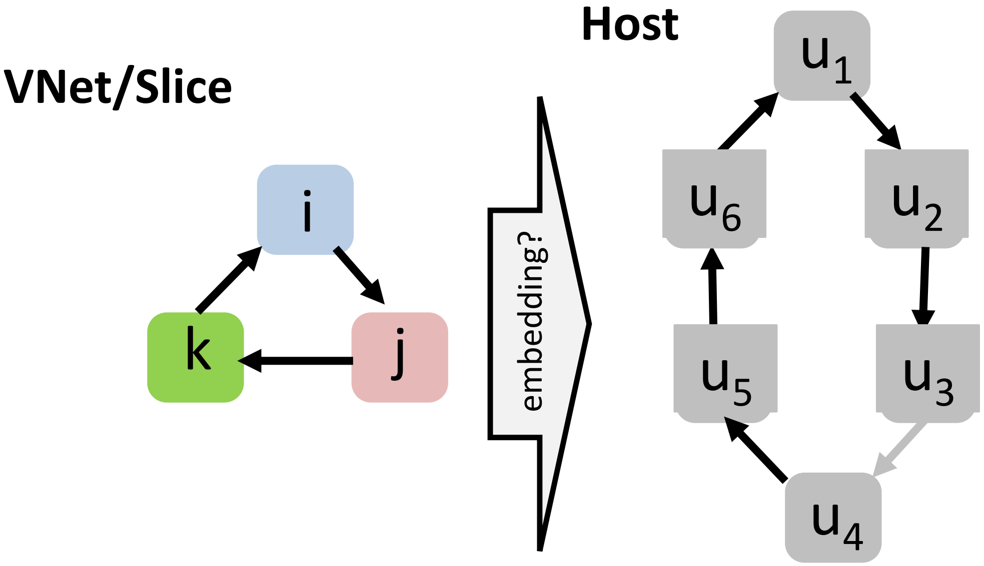
ANF: Still not well understood!

- Problem **NP-hard**
- Goal (α, β)- approximation: α factor from optimum and capacity violation at most factor β
- ***Challenge 1:*** **Randomized rounding** with low augmentation: so far $\boldsymbol{\alpha = O(1)}$, $\boldsymbol{\beta} = \sqrt{n}$

Liu, Richa, Rost, Schmid, 2018.

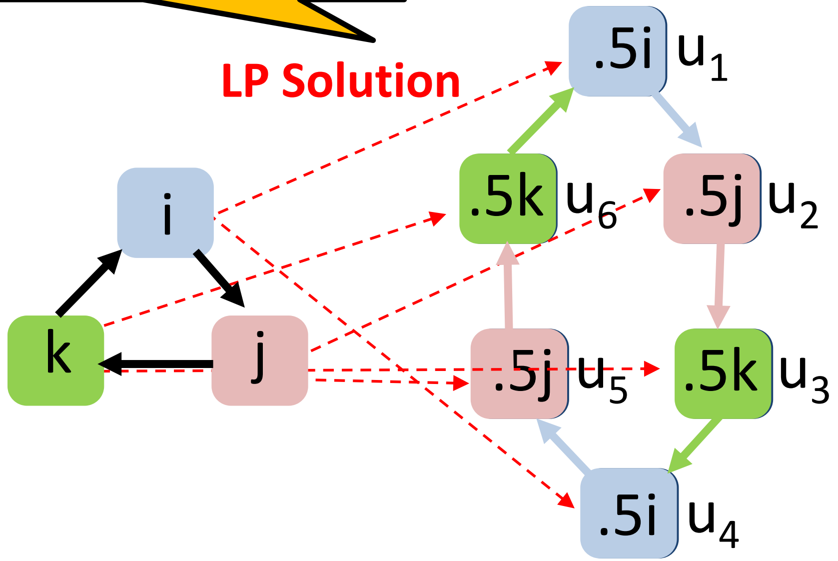
- Can we do better?!

Challenge 2 - Decomposable ILP Formulations: Randomized Rounding based on MCF Can Fail!



Challenge 2 - Decomposable ILP Formulations: unding based on MCF Can Fail!

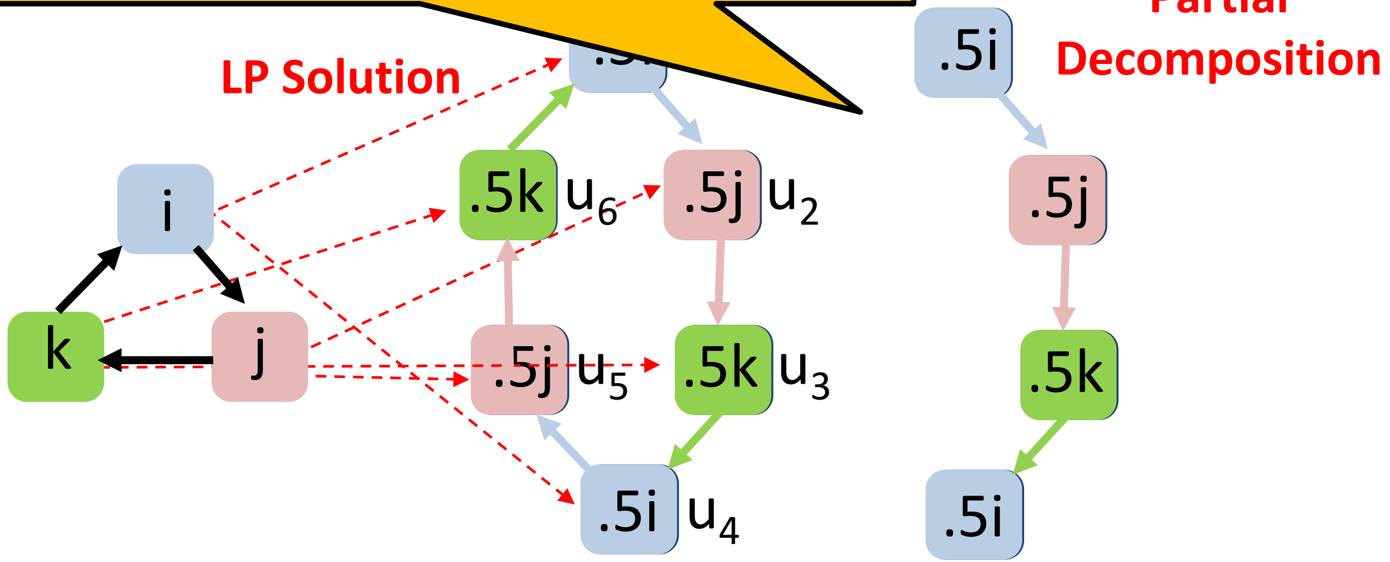
Valid LP solution: virtual node mappings sum to 1 and each virtual node connects to its neighboring node **with half a unit of flow**…



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

Formulations:
MCF Can Fail!

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**.



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

Further Reading

- Robust data mule networks with remote healthcare applications in the Amazon region: A fountain code approach Mengxue Liu, Thienne Johnson, Rachit Agarwal, Alon Efrat, Andrea Richa, Mauro Margalho Coutinho. **HealthCom** 2015.

- Charting the Complexity Landscape of Virtual Network Embeddings
Matthias Rost and Stefan Schmid. **IFIP Networking**, Zurich, Switzerland, May 2018.

Conclusion

- Much work ahead: sensor and wireless networks become **larger** and carry more **data**
- An opportunity to make networks **data-/demand-aware**?
- Or to exploit **flexibilities** to precompute, to schedule, to find new tradeoffs between distributed and centralized?
- Sometimes boils down to classic problems (e.g., DTN scheduling): a great time to reconsider!
- Technology is evolving quickly (e.g., drone-to-drone communication): what are the right **models**?

Thank you! Question?

Further Reading

Online Aggregation of the Forwarding Information Base: Accounting for Locality and Churn
Marcin Bienkowski, Nadi Sarrar, Stefan Schmid, and Steve Uhlig.
IEEE/ACM Transactions on Networking (**TON**), 2018.
Exploiting Locality in Distributed SDN Control
Stefan Schmid and Jukka Suomela.
ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**), Hong Kong, China, August 2013.
Tight Bounds for Delay-Sensitive Aggregation
Yvonne Anne Oswald, Stefan Schmid, and Roger Wattenhofer.
27th Annual ACM Symposium on Principles of Distributed Computing (**PODC**), Toronto, Canada, August 2008.
Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks
Chen Avin and Stefan Schmid.
ArXiv Technical Report, July 2018.
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.
SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.
Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.
Charting the Complexity Landscape of Virtual Network Embeddings
Matthias Rost and Stefan Schmid. **IFIP Networking**, Zurich, Switzerland, May 2018.