Imperial College London

Distributed Software Engineering (DSE) Group
Department of Computing

Building Large-scale Distributed Systems with Network Coordinates

Peter Pietzuch
prp@doc.ic.ac.uk

Distributed Software Engineering (DSE) Group
Department of Computing
Imperial College London

Joint work with
Jonathan Ledlie[1] and Margo Seltzer
Division of Engineering and Applied Science
Harvard University

University College London – EE Department – May 2008

[1]Many slides courtesy of Jonathan

New Applications ➔ New Demands

New Internet-scale distributed applications

- Internet TV (e.g. BBC iPlayer, Zattoo, Joost, ...)
- Streaming video (e.g. YouTube, Netflix, iTunes, ...)
- Distributed multi-player games (e.g. WoW, CS, ...)
- Peer-assisted file distribution (e.g. BitTorrent, Vudu, ...)

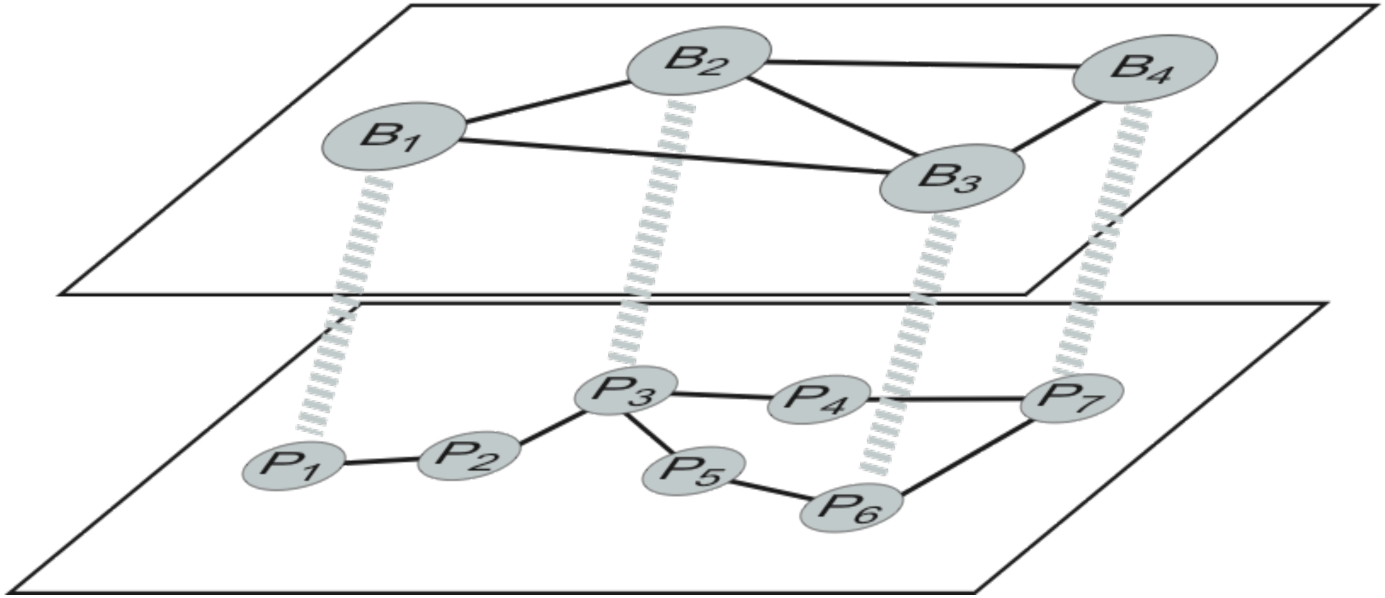
New demands

- Peer-to-peer connections; not client/server model
- Scalability: Millions of concurrent users
- Intolerant to high latency/jitter: VoIP, FP shooter games
- Intolerant to poor bandwidth: file downloads, media streaming

Locality in Overlay Networks

Overlay
Network
(e.g. P2P app)

Physical
Network
(e.g. Internet)



Insight: Exploit flexibility in choice of overlay neighbours

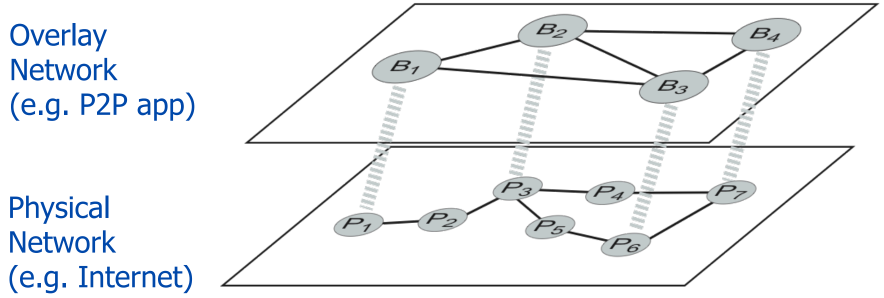
Why does Locality matter?

- Lower latency
- Lower network utilisation
  - Process data close to data sources and discard locally
  - "Don't send data to Australia and back."
- Better reliability
  - Data traverses fewer network links and routers
- Higher bandwidth
  - Inverse correlation between latency and available bandwidth
- Lower cost
  - Choose peers from same autonomous system

Why is Locality-awareness hard?

Locality metrics

- Different applications/ nodes require different metrics



Measurement overhead

- Underlying network is opaque
- Burden of taking measurements
  - Per measurement overhead
  - All pairs measurements in topology $O(n^2)$
- Dissemination of measurement results

Network Coordinates (NCs) to the Rescue

Embed inter-node **latency** measurements into metric space

- Measure only (small) subset of network
- Establish coordinates for nodes

Purpose

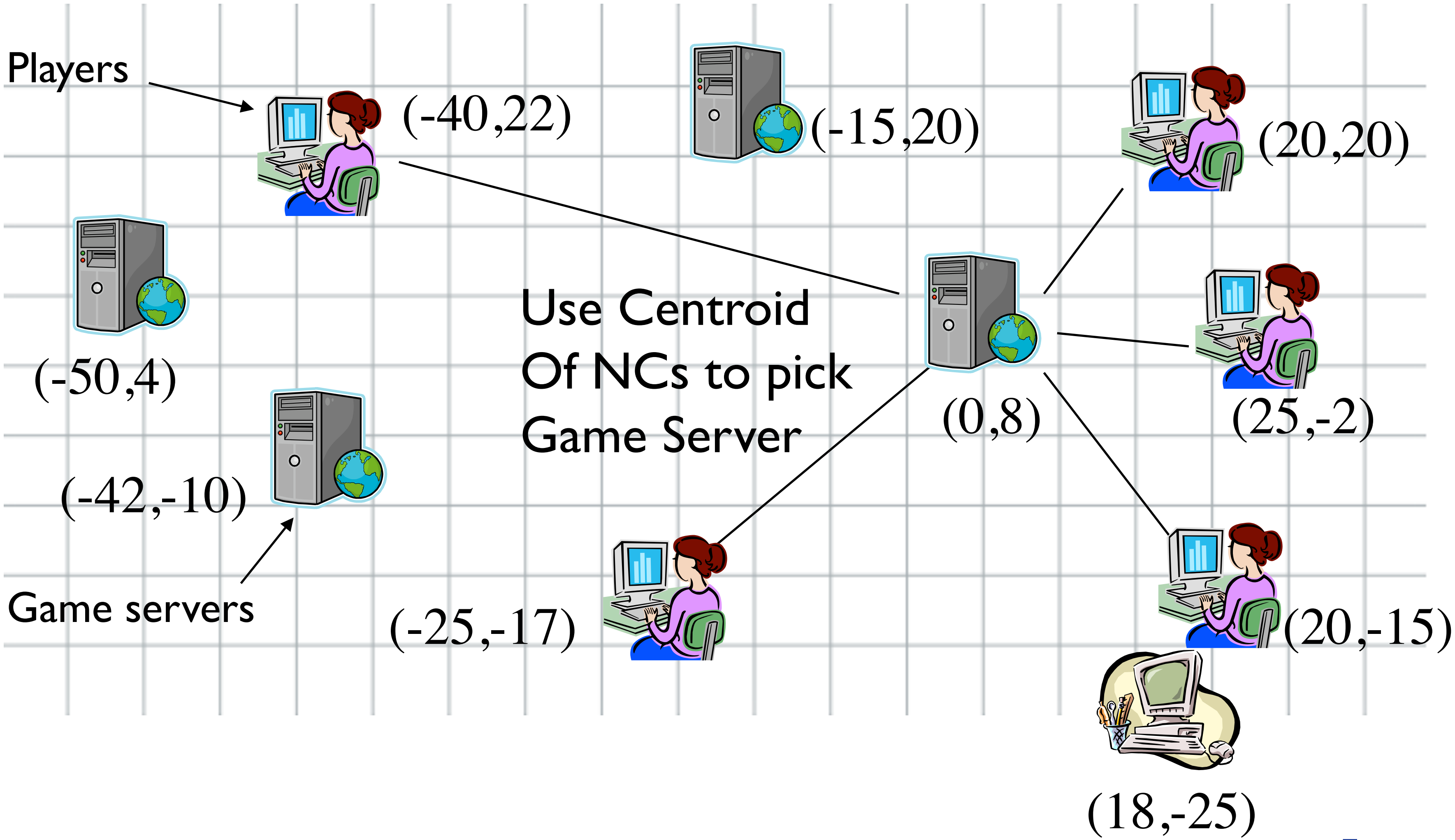
- Predict missing measurements

Works with low dimensional space

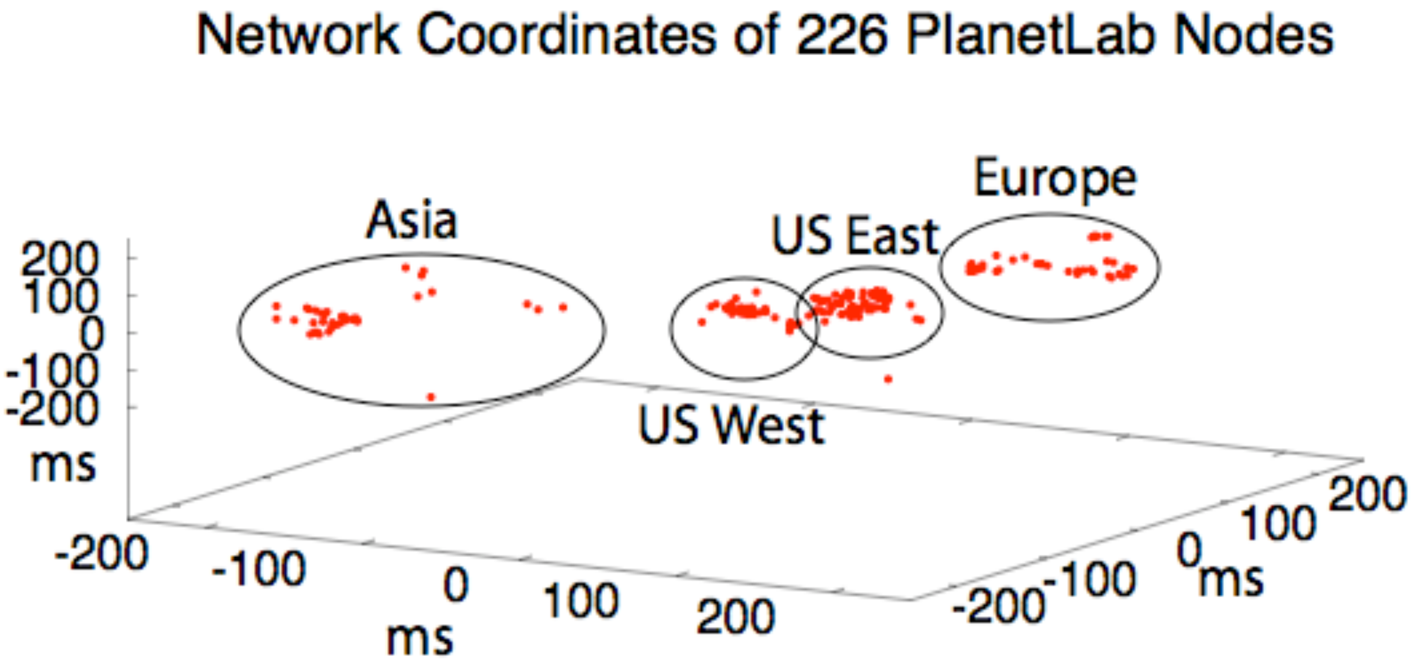
- 2-5 dimensions in practice

NCs Simplify Distributed Systems Problems

Pick game server with lowest mean latency



Network Coordinates on PlanetLab



Points represent locations of PlanetLab nodes in 3D relative coordinate space

Overview

Introduction

- Network Coordinates
- Decentralised NC computation: Vivaldi

Practical NCs: Accuracy and Stability

- Challenges and Solutions

Applications of NCs

- Routing overlays
- Placement of stream operators
- Locality-awareness in Bittorrent

Open Questions and Conclusions

How are NCs calculated?

Landmark-based algorithms

(e.g. GNP [CMU], Lighthouses [Cambridge], PIC [MSR], ...)

- Each node measures latency to set of landmark nodes
- Use landmark nodes to calculate own coordinate

Simulation-based algorithms

(e.g. **Vivaldi** [MIT], Big Bang [Tel Aviv], ...)

- Each node measures latency to random other nodes
- Model embedding as physical system
    - Network of springs, particles in force field, ...

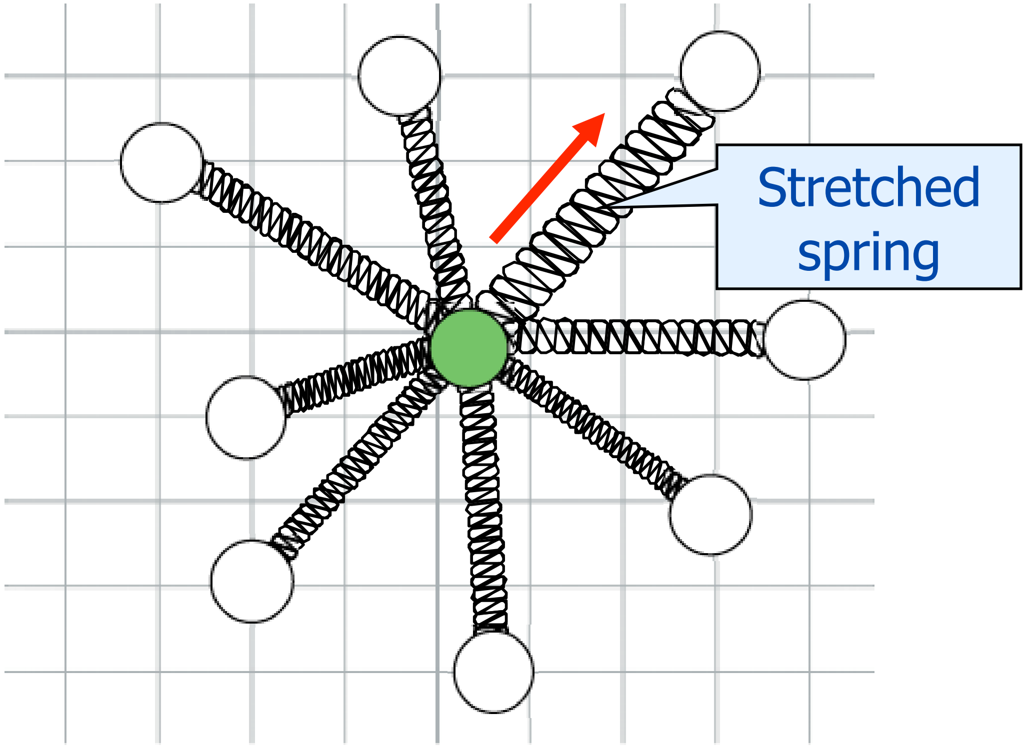
Vivaldi Algorithm

Concept: Springs connect all nodes

Vivaldi [Cox03, Dabek04]

$O(n^2)$ springs

Rest length of spring (a,b) = lat(a,b)

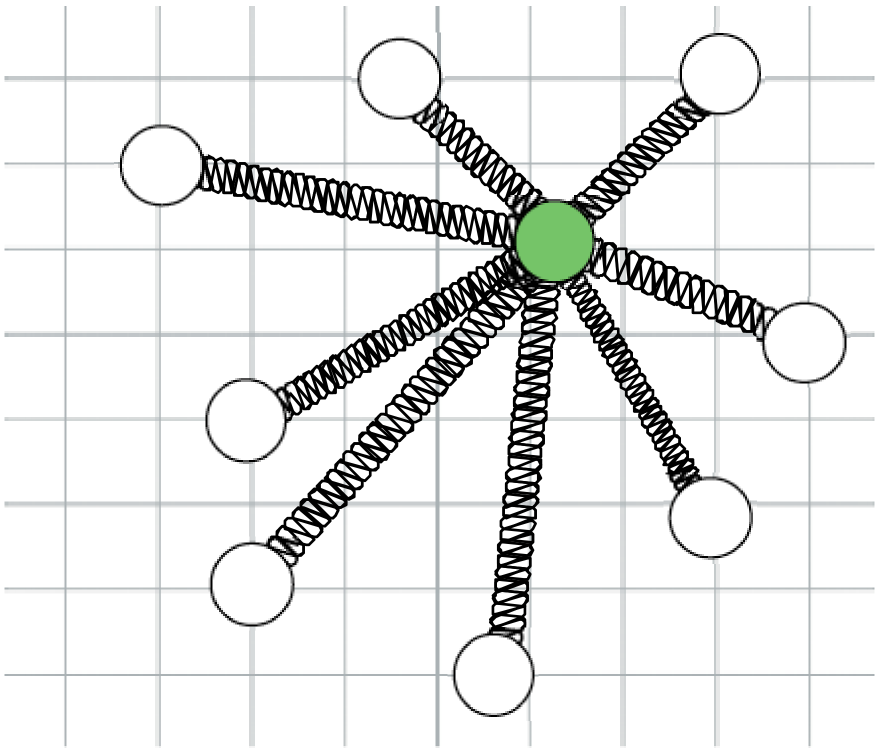


Vivaldi: Adjustment

Concept: Springs connect all nodes

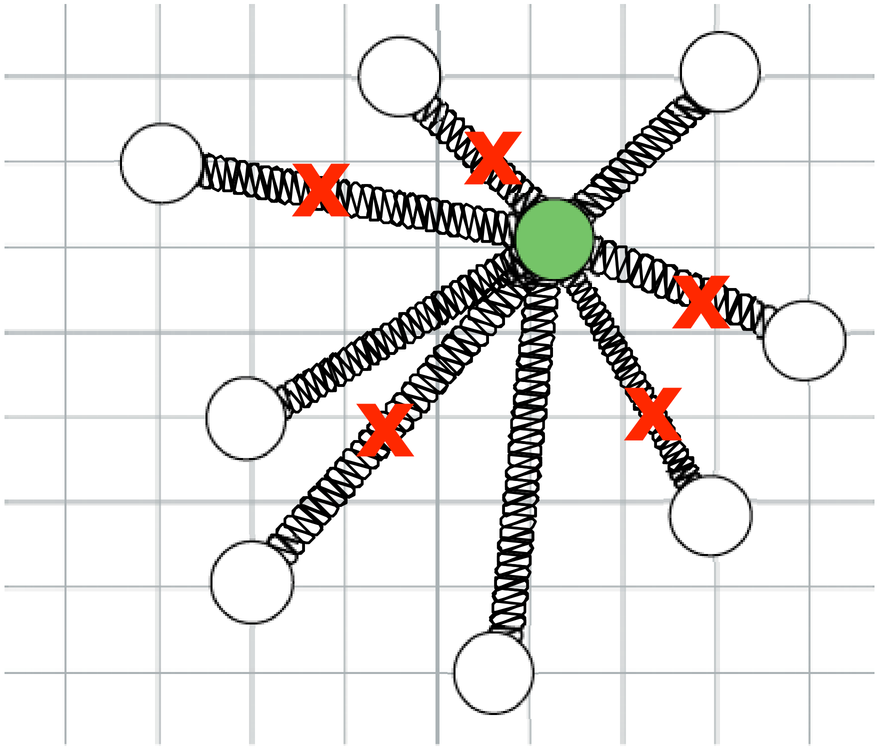
Nodes adjust coords
- Simulate spring forces

Move to "low energy" state
- Abstract position mirrors physical latency



Vivaldi: Made Feasible

In practice: Use handful of springs



Measurement complexity becomes O[log(n)]

Vivaldi in Detail

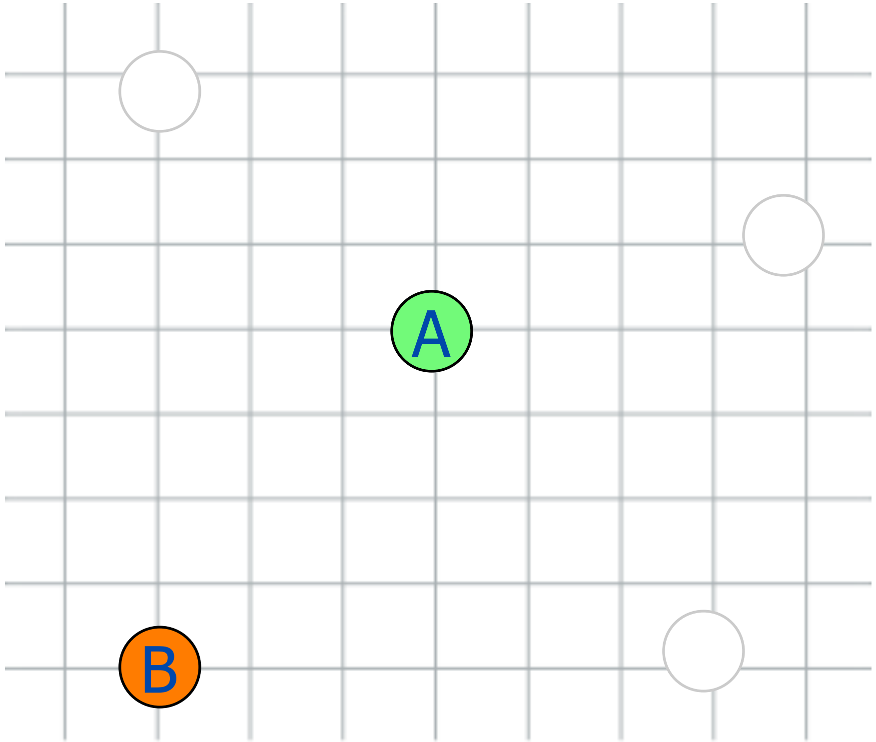
Incremental refinement: minimise global prediction error

Continuous Loop:

- Measure to a few nodes
- Determine coordinate
  - Low-dim space

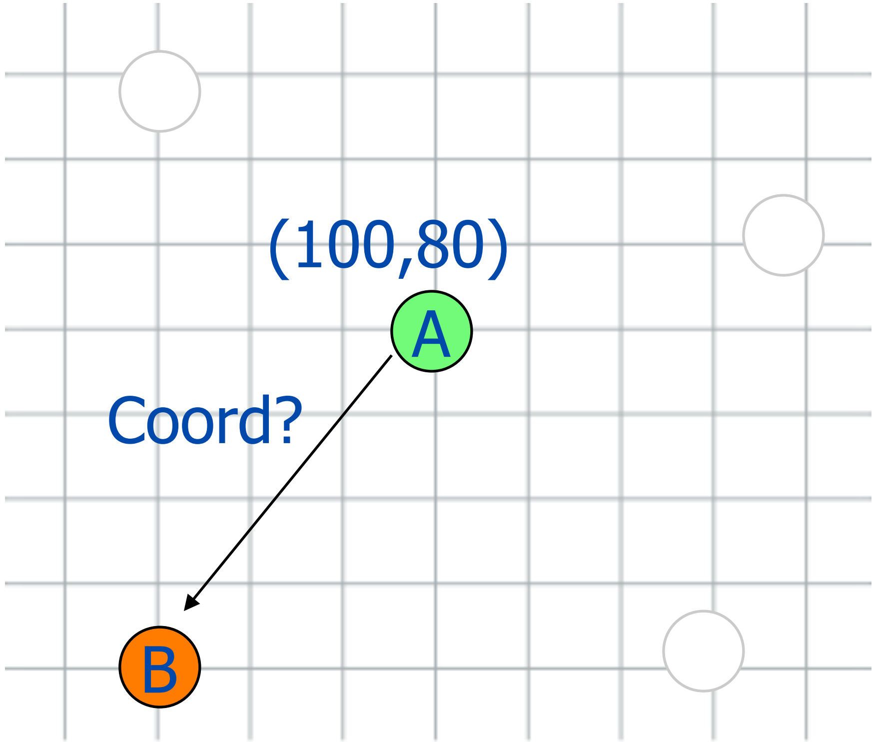
Result:

- Predict latencies to rest of network



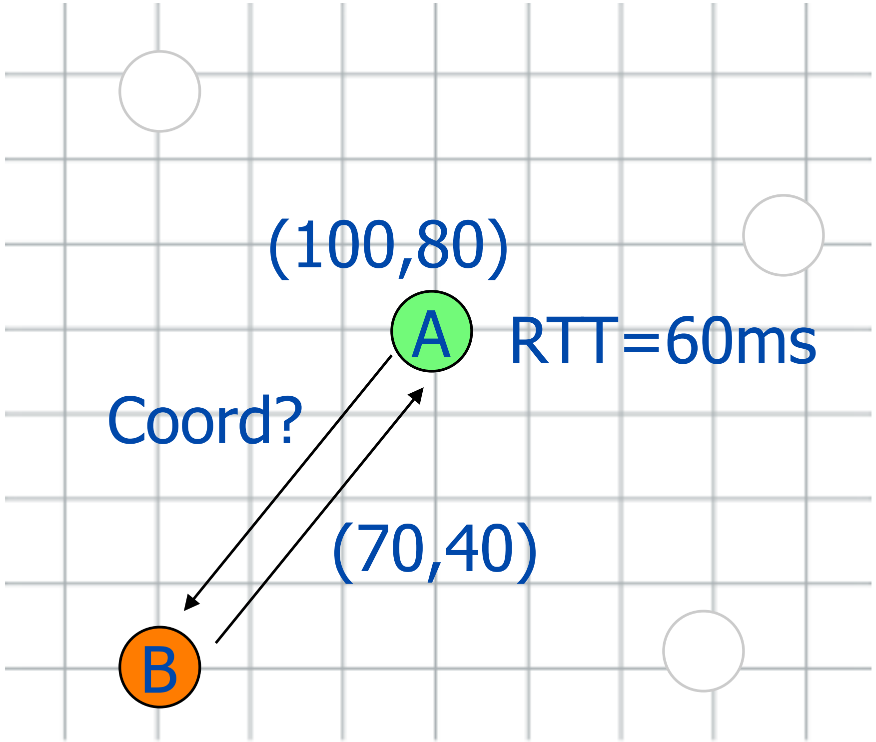
Vivaldi: Measurement

1. A measures latency to B.



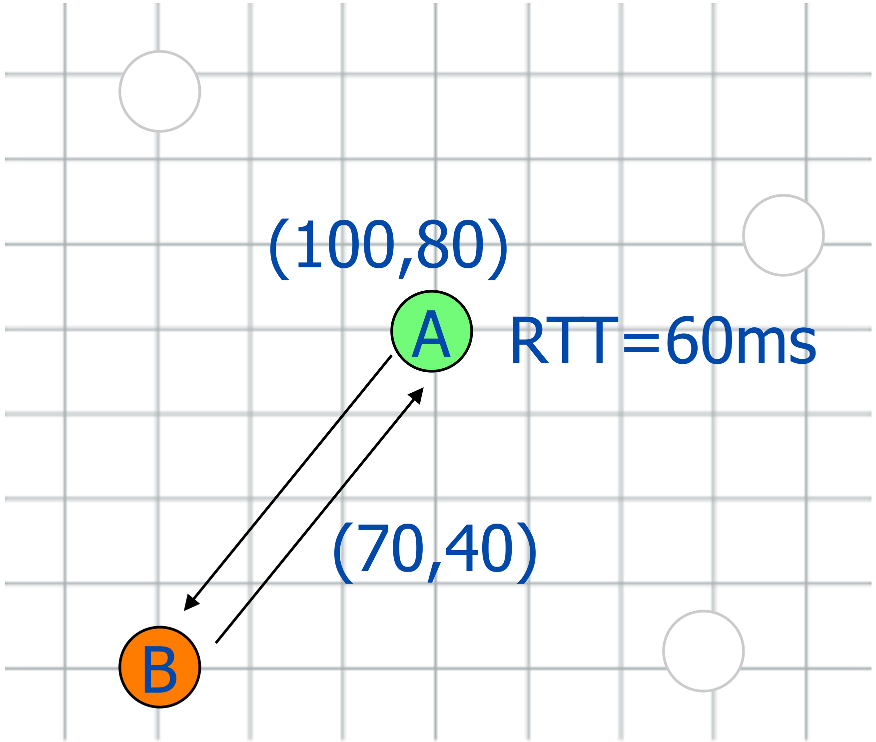
Vivaldi: Reply

1. A measures latency to B.
2. B replies with its coord. A deduces RTT.



Vivaldi: Computation

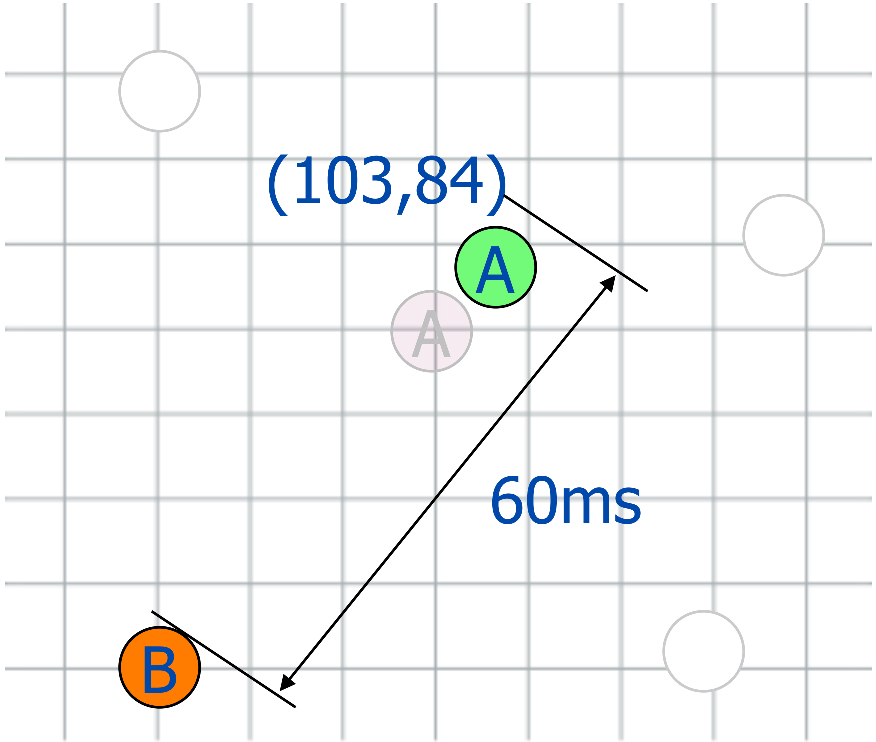
1. A measures latency to B.
2. B replies with its coord. A deduces RTT.
3. A computes estimate and error.



Estimate = |(100,80)-(70,40)|=50ms
Error = (60 - Estimate) = 10ms

Vivaldi: Adjustment

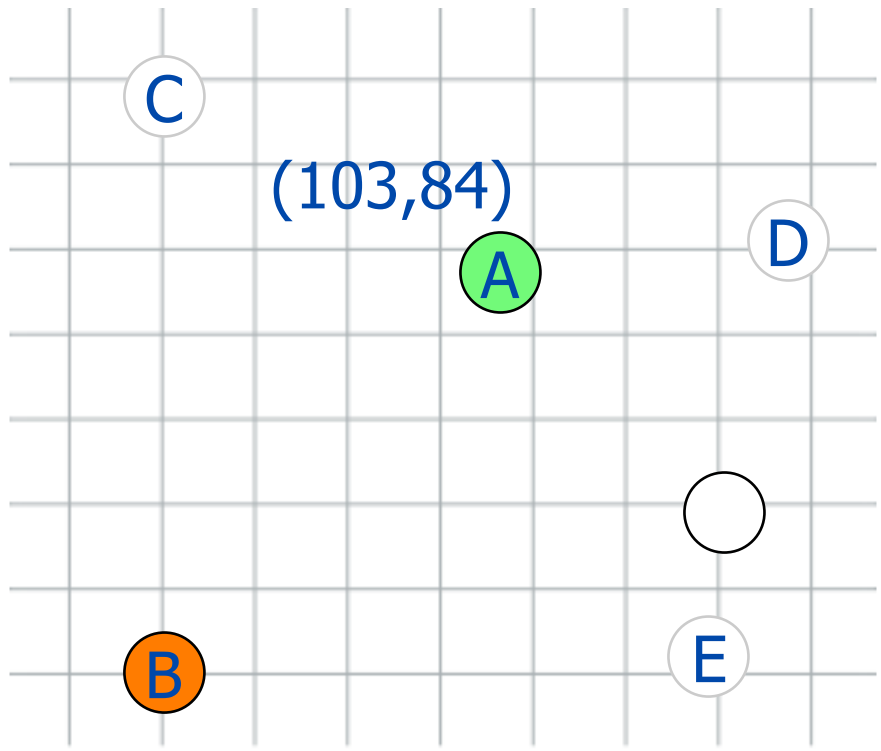
1. A measures latency to B.
2. B replies with its coord. A deduces RTT.
3. A computes estimate and error.
4. A moves toward ideal coord, relative to B.



Estimate = |(100,80)-(70,40)|=50ms
Error = (60 - Estimate) = 10ms

Vivaldi: Repeat

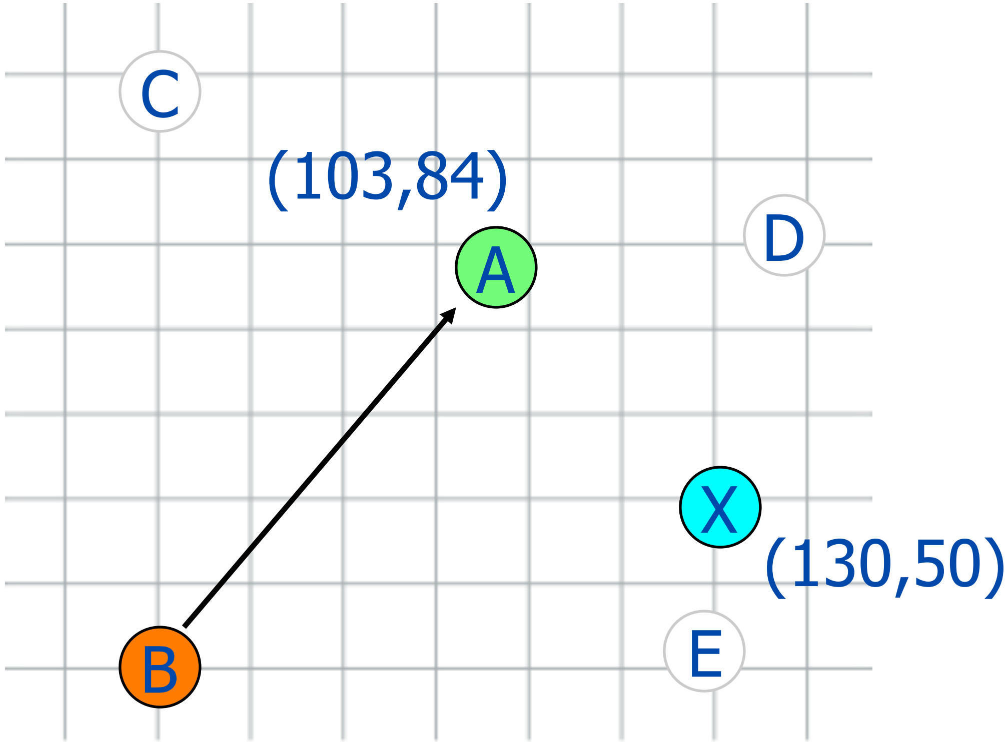
1. A measures latency to B.
2. B replies with its coord. A deduces RTT.
3. A computes estimate and error.
4. A moves toward ideal coord, relative to B.
5. Repeat with C, D, E.



Vivaldi: Predict

A has never seen or measured RTT to **X**

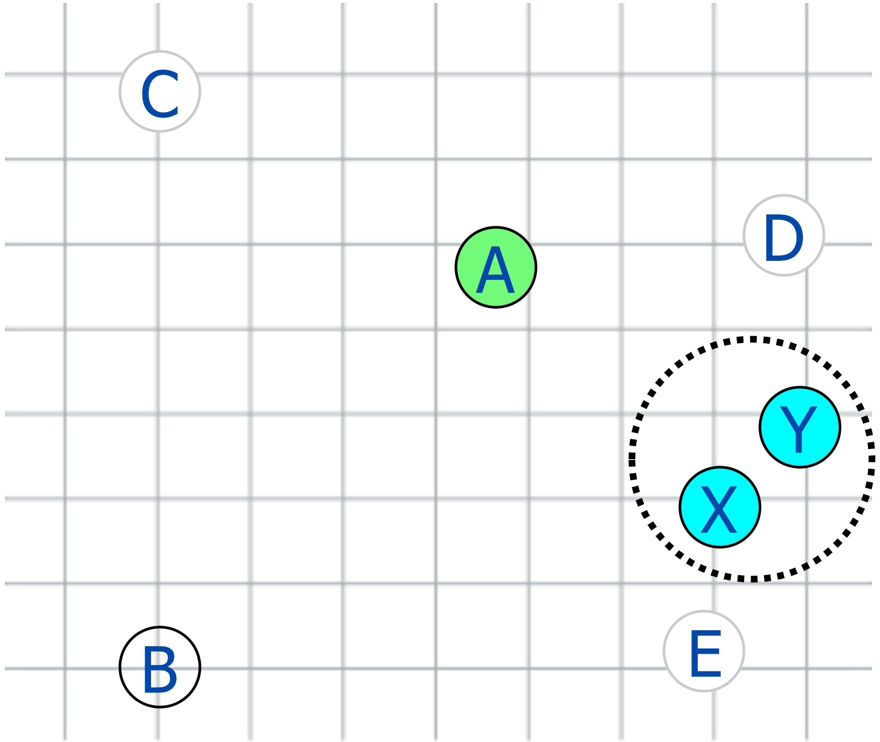
1. A measures latency to B.
2. B replies with its coord. A deduces RTT.
3. A computes estimate and error.
4. A moves toward ideal coord, relative to B.
5. Repeat with C, D, E.
6. Predict to X



Vivaldi: Predict

A can predict locality of **X** and **Y**.

1. A measures latency to B.
2. B replies with its coord. A deduces RTT.
3. A computes estimate and error.
4. A moves toward ideal coord, relative to B.
5. Repeat with C, D, E.
6. Predict to X



Practical Challenges

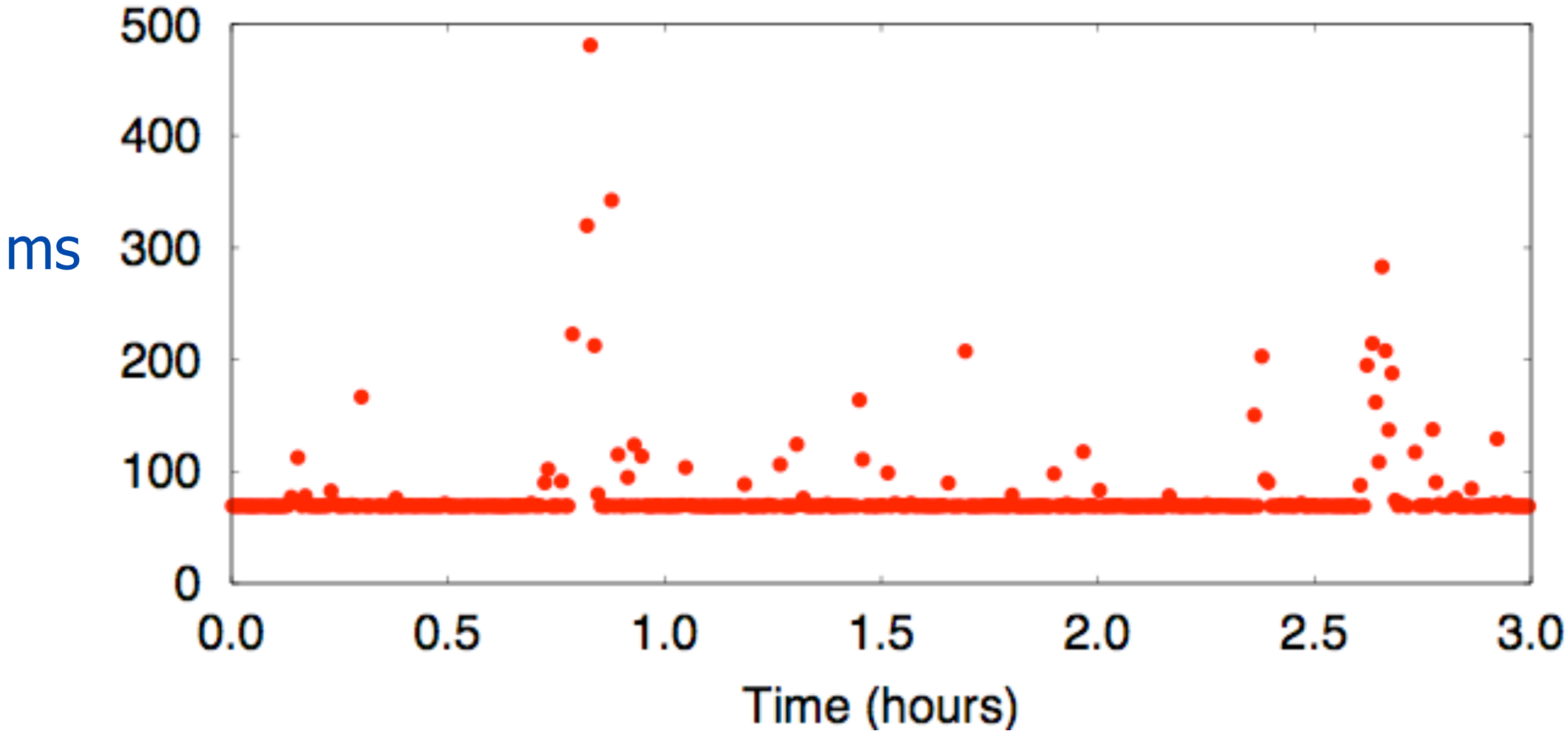
Problem 1: Latency measurements vary

Problem 2: Applications want stable coordinates

Problem 3: Selecting overlay nodes for measurements

Problem 1: Measurement changes I

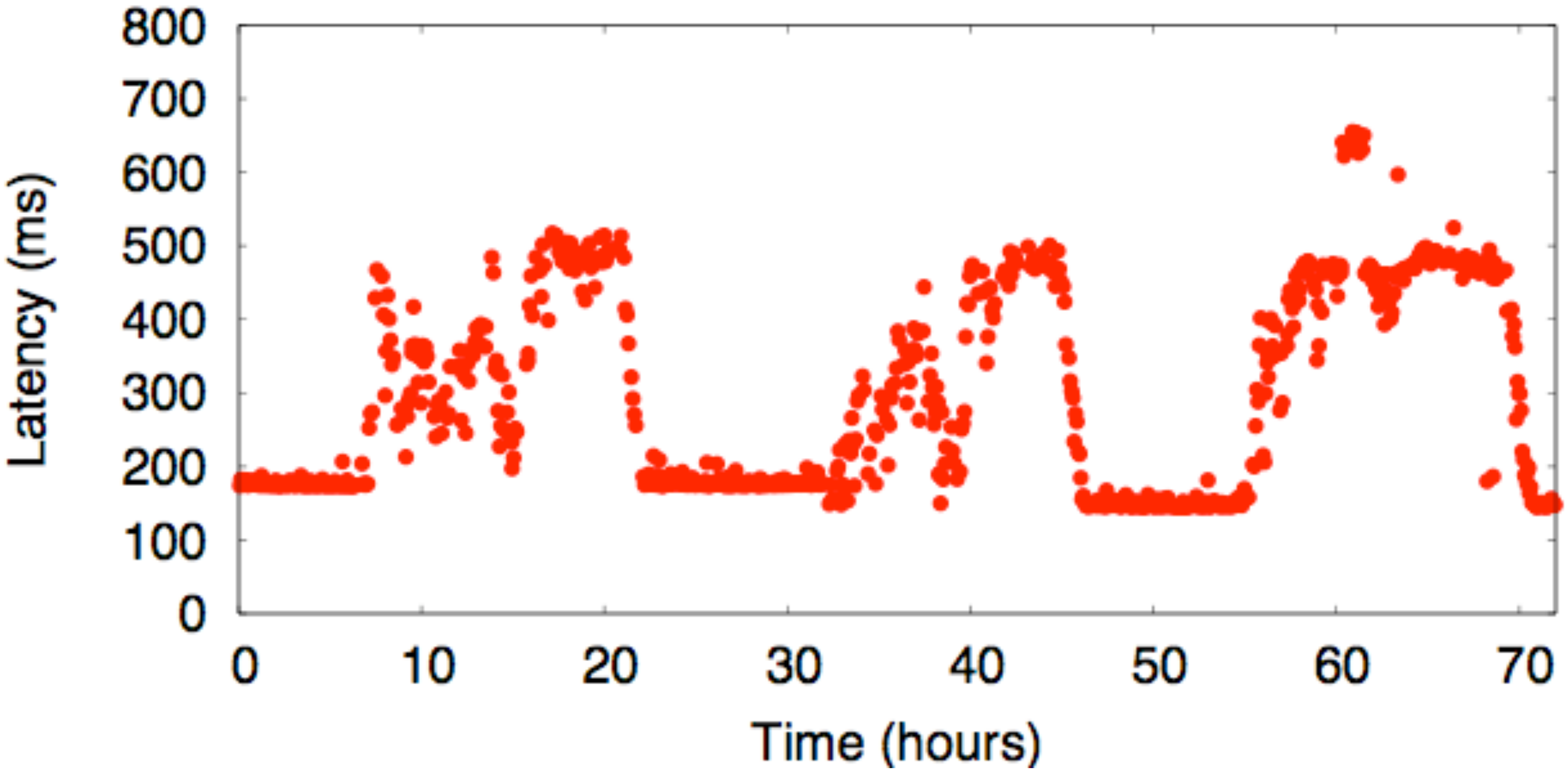
Three hours of measurements from berkeley to uvic.ca



82% of measurements within 1 ms of median

Problem 1: Measurement changes II

3 days of measurements from ntu.edu.tw to 6planetlab.edu.cn



Need to remove noise, but remain adaptive

Moving Minimum as Latency Filter

Remove outliers and respond to latency change

Time		age	newest			oldest
t_0		ms	60	61	59	62
t_1	Receive 1000ms RTT		1000	60	61	59
t_2	Receive 200ms RTT		200	1000	60	61

Other simple techniques did not work

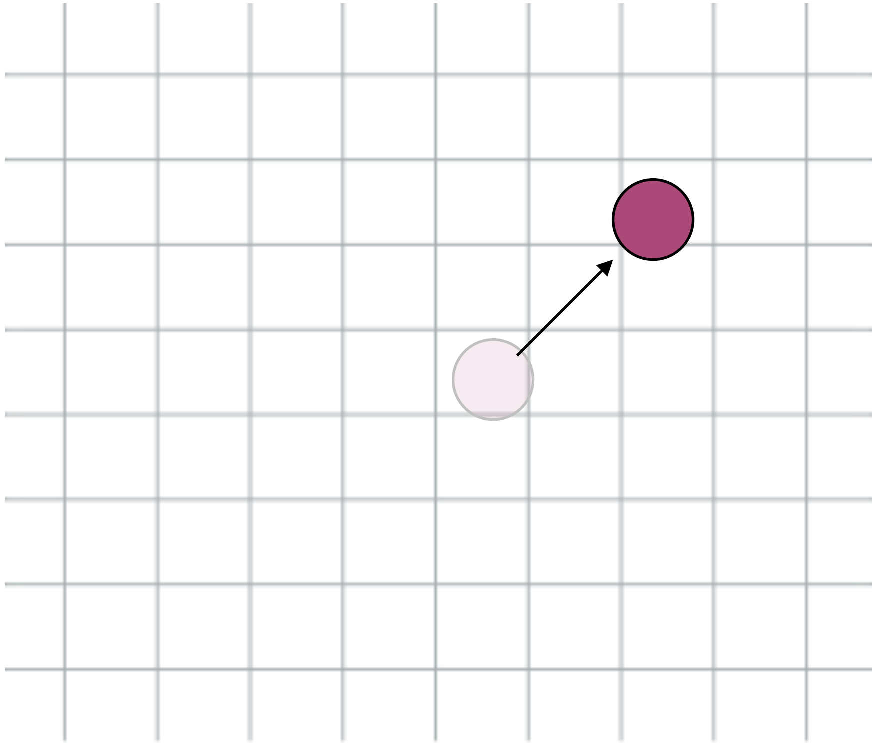
Problem 2: Stability

Problem: coordinate change expensive

– Application must determine if change needs action

Short-term variations should not cause coordinate changes

– But need to track longer-term changes (e.g. BGP updates)
– Possible to tell apps less frequently and retain high accuracy?



Coordinate Windows as Update Filters

1. Keep history of recent coordinates
2. Divide history into two windows (sets):

   current (newest) and **start** (oldest)

3. When current and start diverge (by some metric), update application with new coordinate

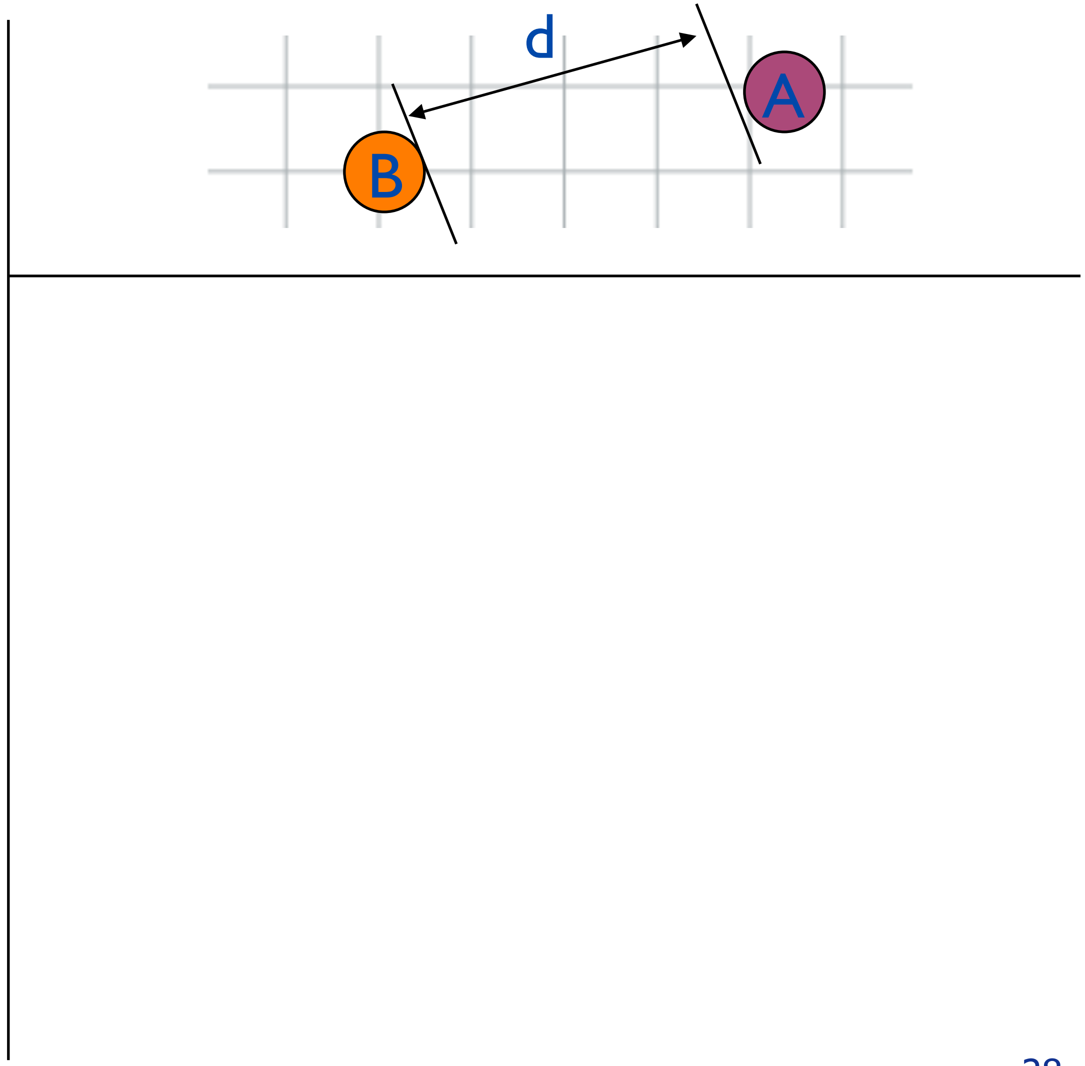
Two possible metrics:

- Local Relative Distance
- Energy

Update Filters

Base update on distance moved relative to nearest neighbor

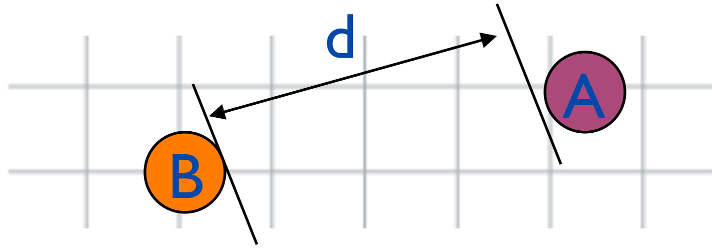
1. Remember nearest known neighbor



Update Filters

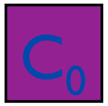
Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s

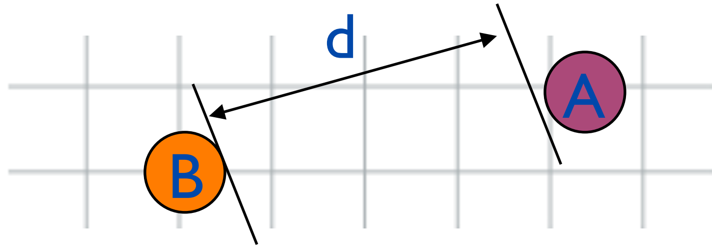


Current W_c

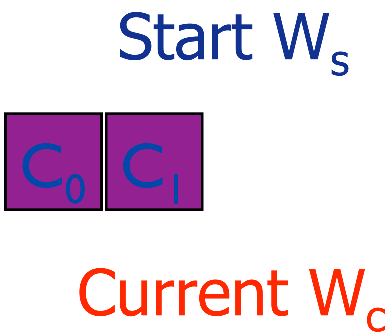
Update Filters

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



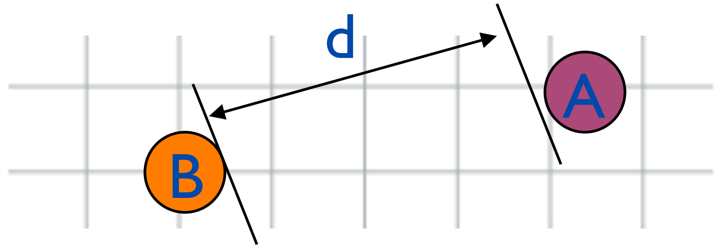
2. Add coordinates to start and current windows



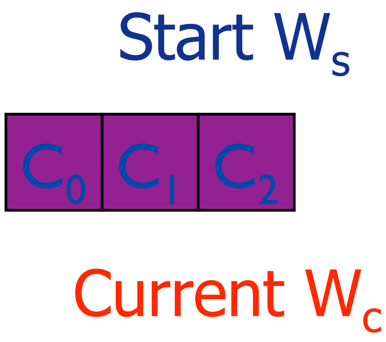
Update Filters

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



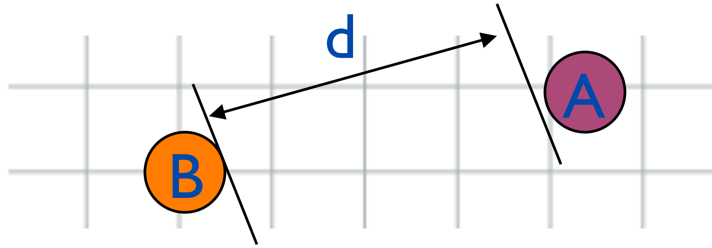
2. Add coordinates to start and current windows



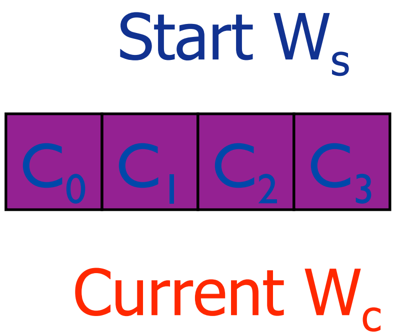
Update Filters

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



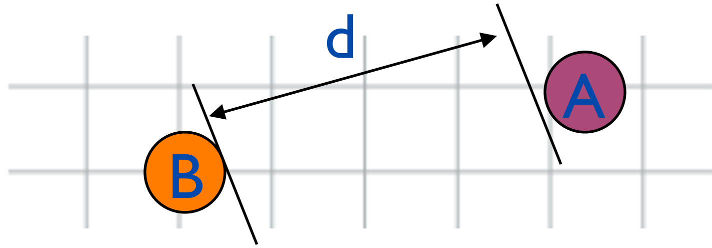
2. Add coordinates to start and current windows



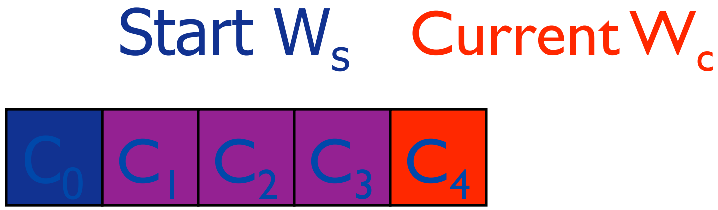
Update Filters

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



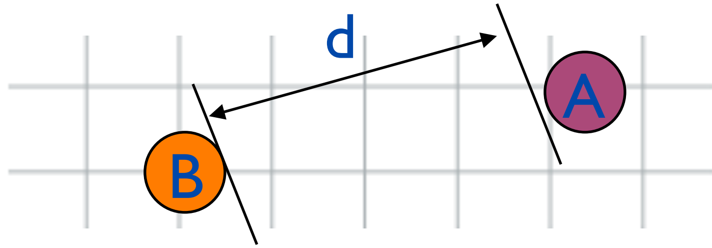
2. Add coordinates to start and current windows



Update Filters

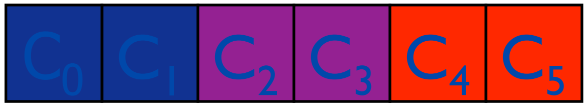
Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows

Start W_s Current W_c



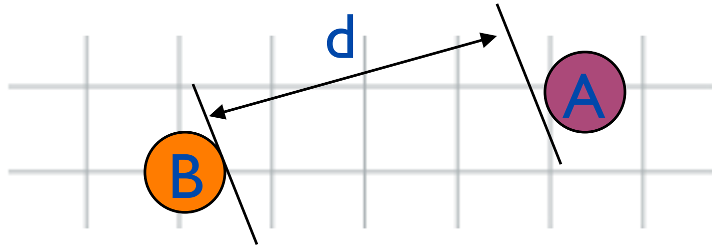
3. Compare centroids of windows

If $Centroid(W_s) - Centroid(W_c) > d \times \varepsilon$

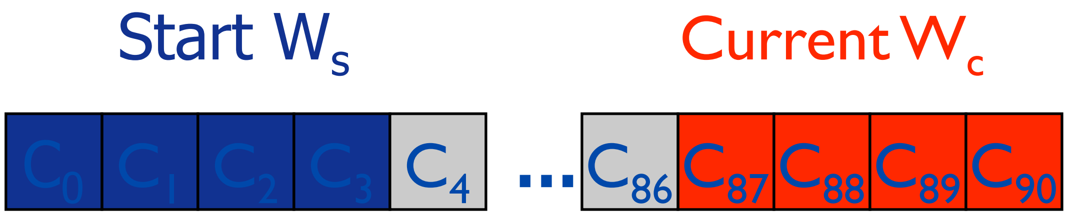
Update Filters

Base update on distance moved relative to nearest neighbor

1. Remember nearest known neighbor



2. Add coordinates to start and current windows



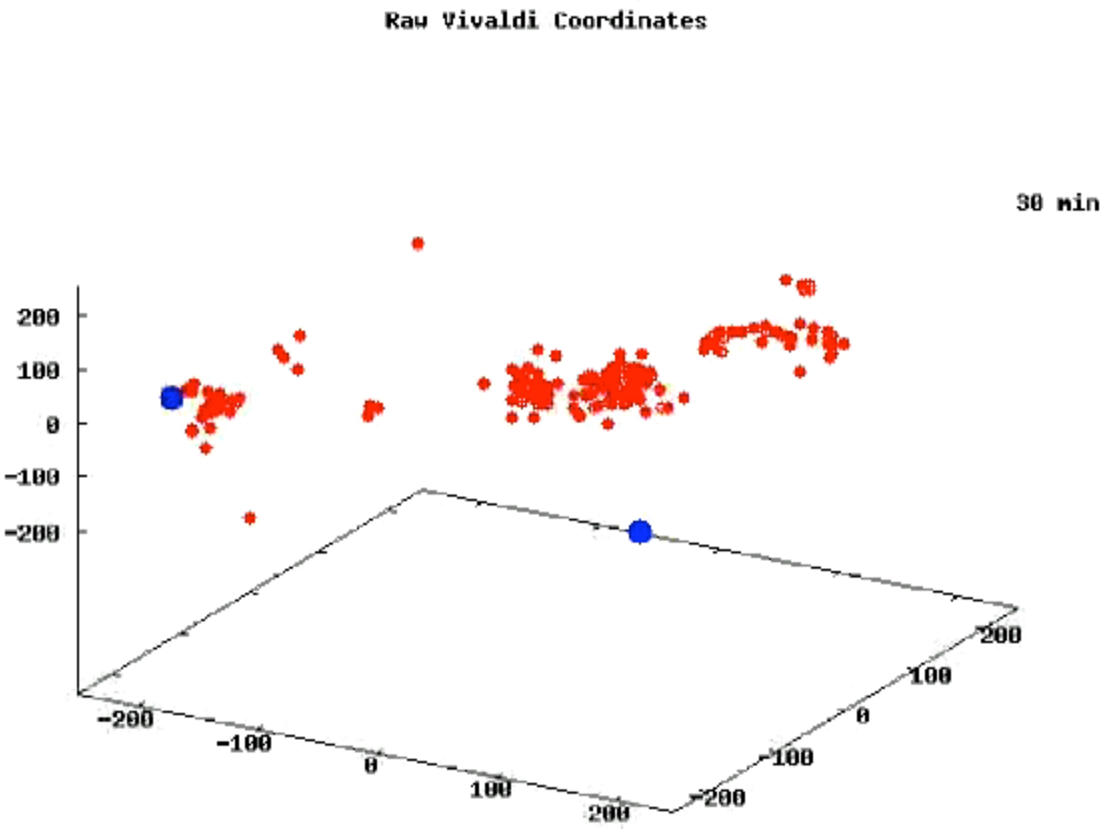
3. Compare centroids of windows

If $Centroid(W_s) - Centroid(W_c) > d \times \varepsilon$

4. Update app-level coordinate

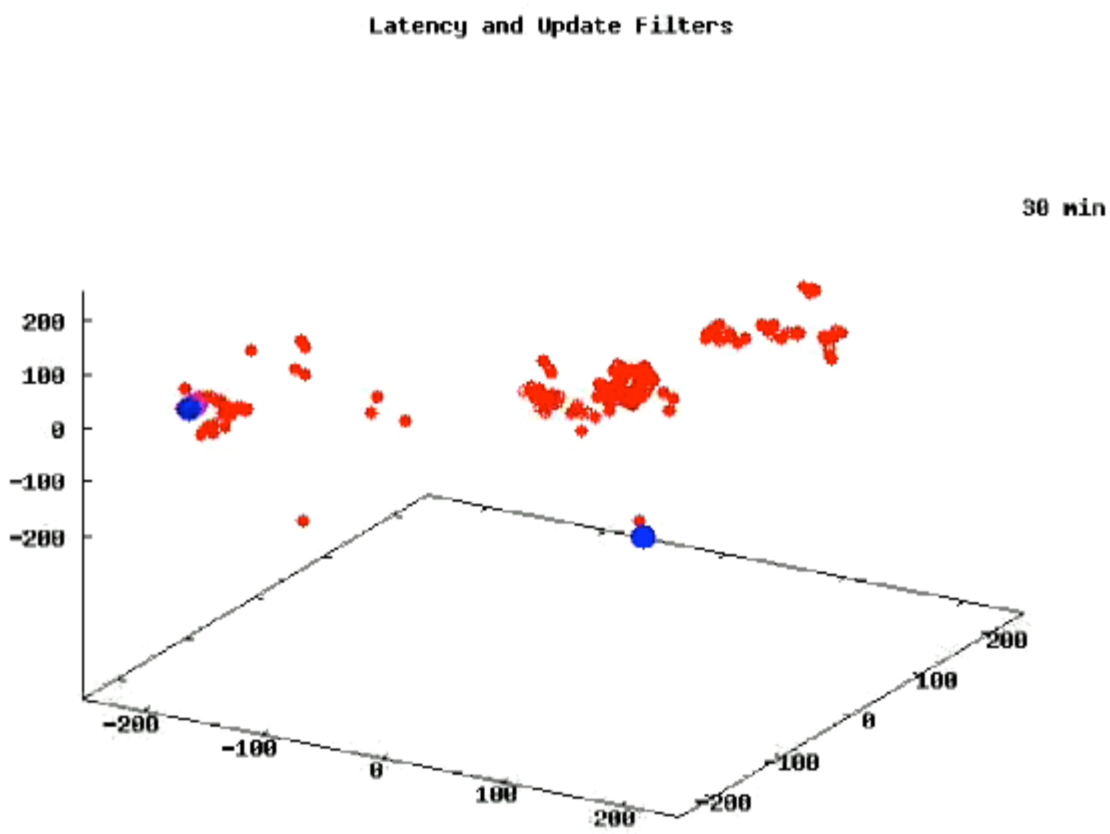
App coord = $Centroid(W_c)$

Video: Raw NCs



226 PL nodes with “raw” NCs
10 min video after NCs stabilised

Video: Latency and Update Filters



226 PL nodes with NCs using latency and update filters

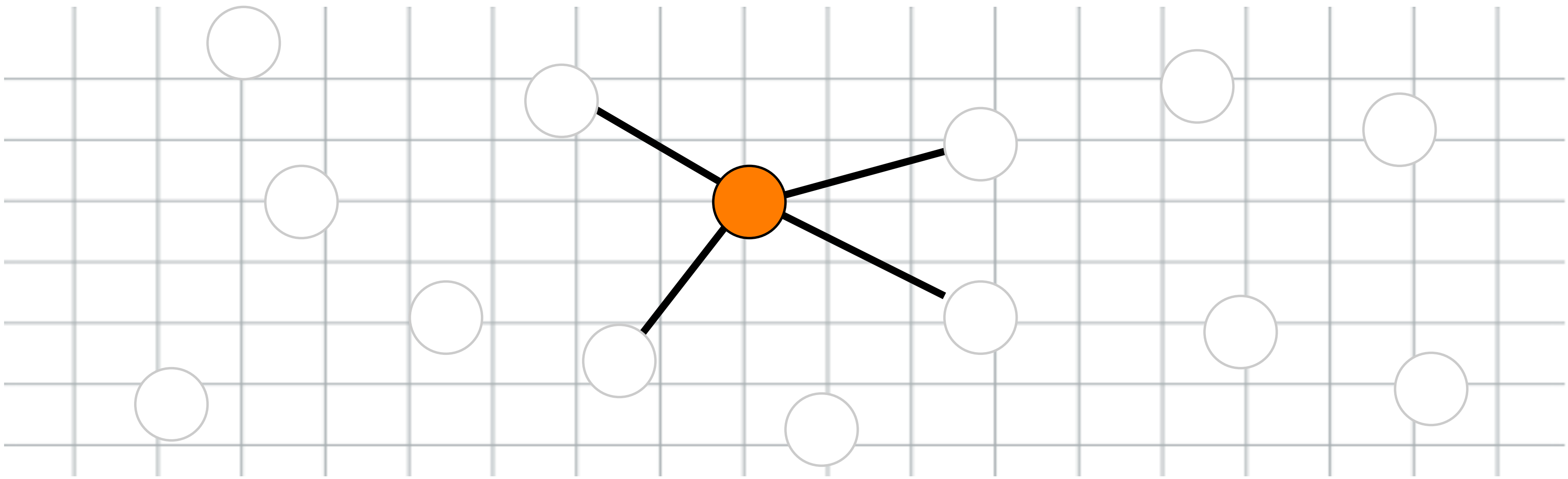
Problem 3: Choice of Measurement Nodes

Often NC measurements "piggybacks" on app-level messages

- Good: Zero additional messages
- Bad: Limits view of network to routing table

Measurement set biased to nearby nodes

- Local bias damages accuracy
- Creates islands: poor estimation beyond horizon



Idea: Expand Horizon over Time

Make tug proportional to distance

- Boost impact of (occasional) long range contacts
- But what's the right weight?

Scale push/pull per neighbor by age

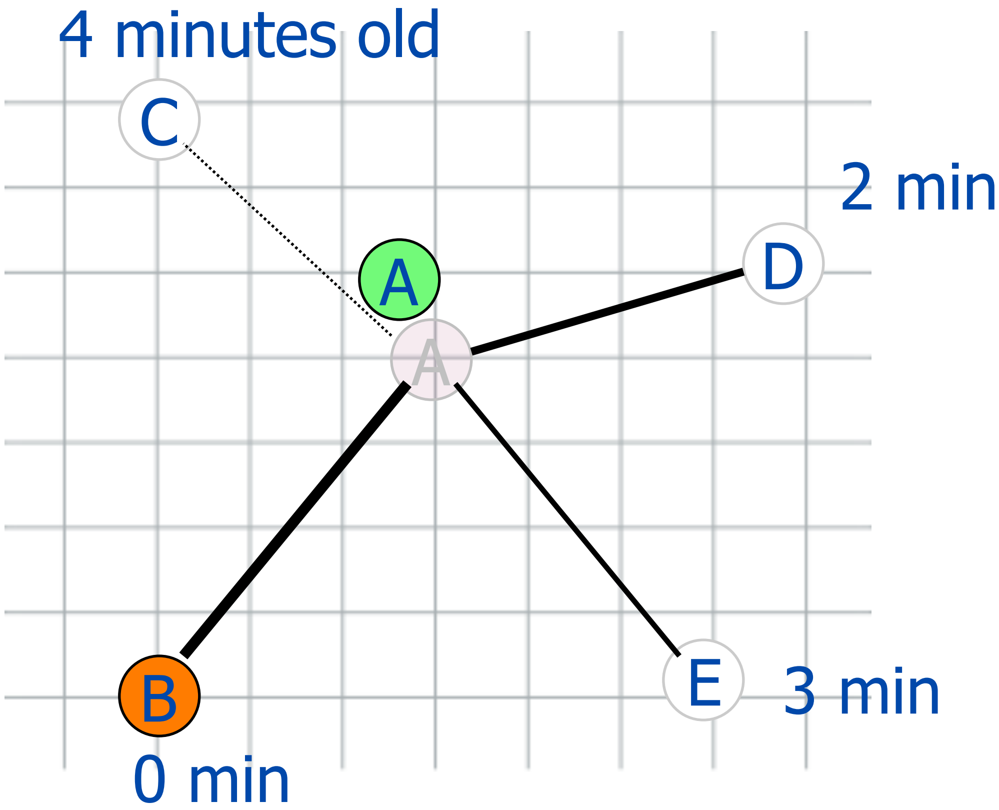
- Decaying tug of neighbor over time: **neighbor decay**

In Effect

- Limits impact of high frequency (nearby) neighbors
- Extends impact of low frequency (longer-distance) ones

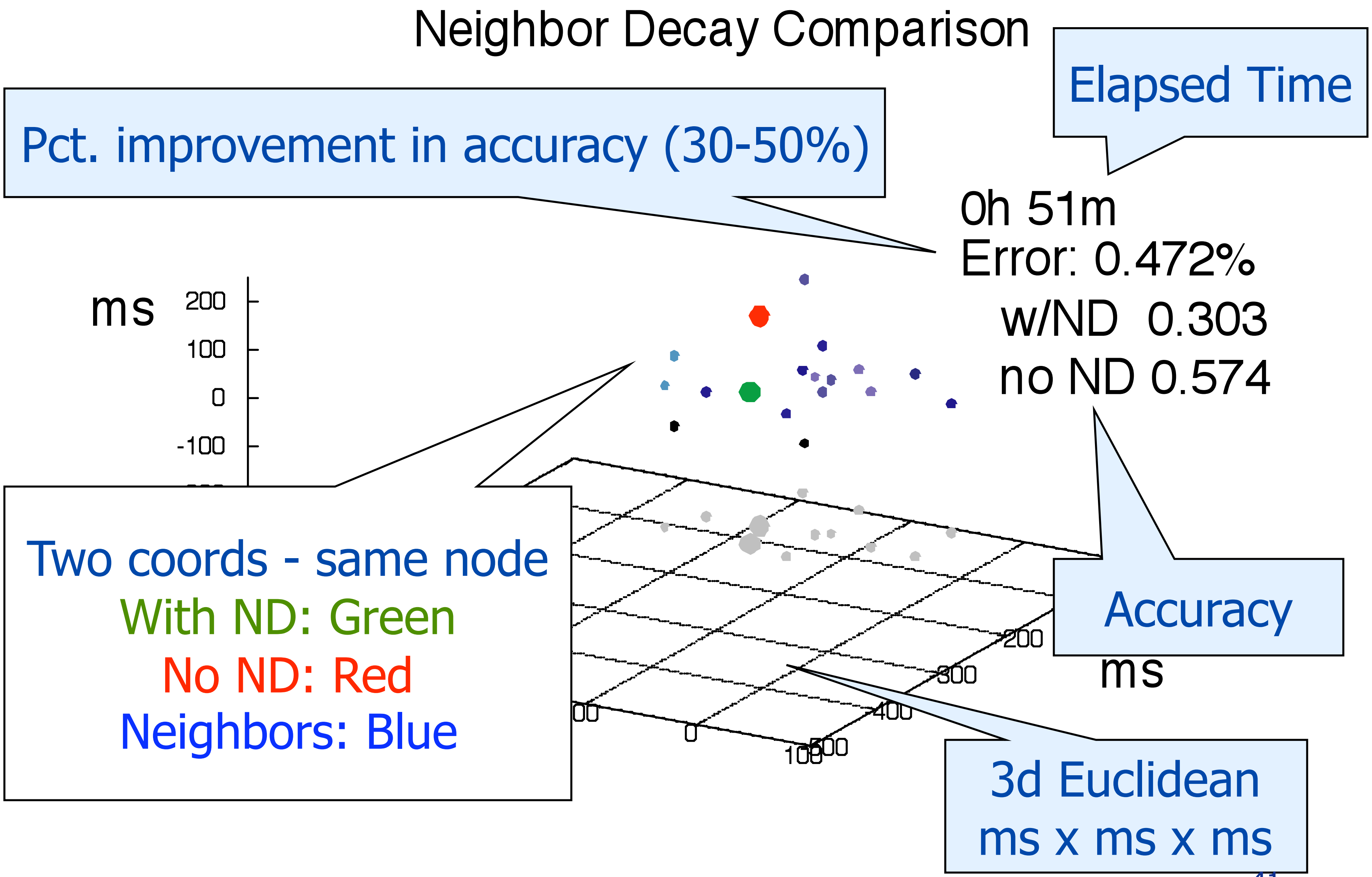
Neighbor Decay Adjustment Step

As springs age, they loosen

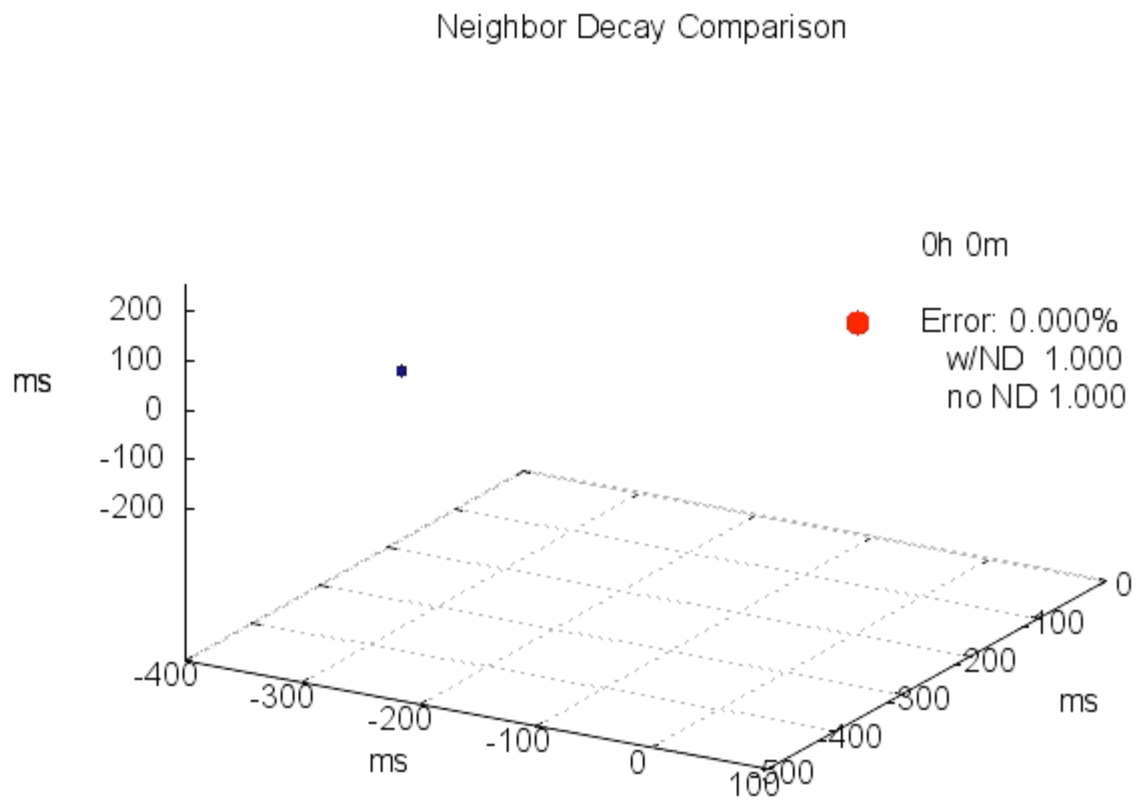


Older information gets less weight

Reading the Neighbor Decay Video



Video: Neighbor Decay



First half hour in life of two NCs on Azureus with/without neighbor decay

Overview

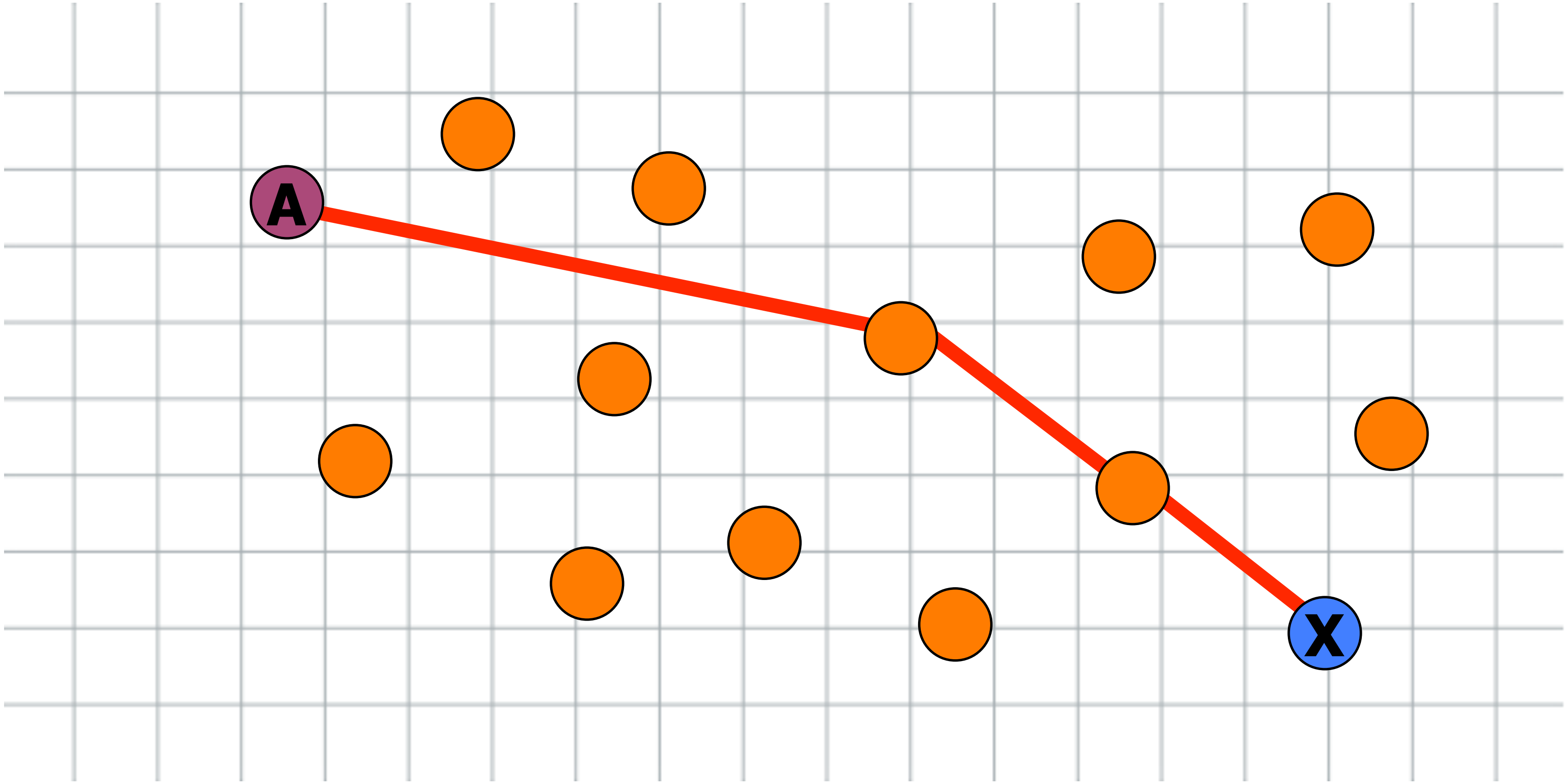
Introduction
- Network Coordinates
- Decentralised NC computation: Vivaldi

Practical NCs: Accuracy and Stability
- Challenges and Solutions

Applications of NCs
- Routing overlays
- Placement of stream operators
- Locality-awareness in Bittorrent

Open Questions and Conclusions

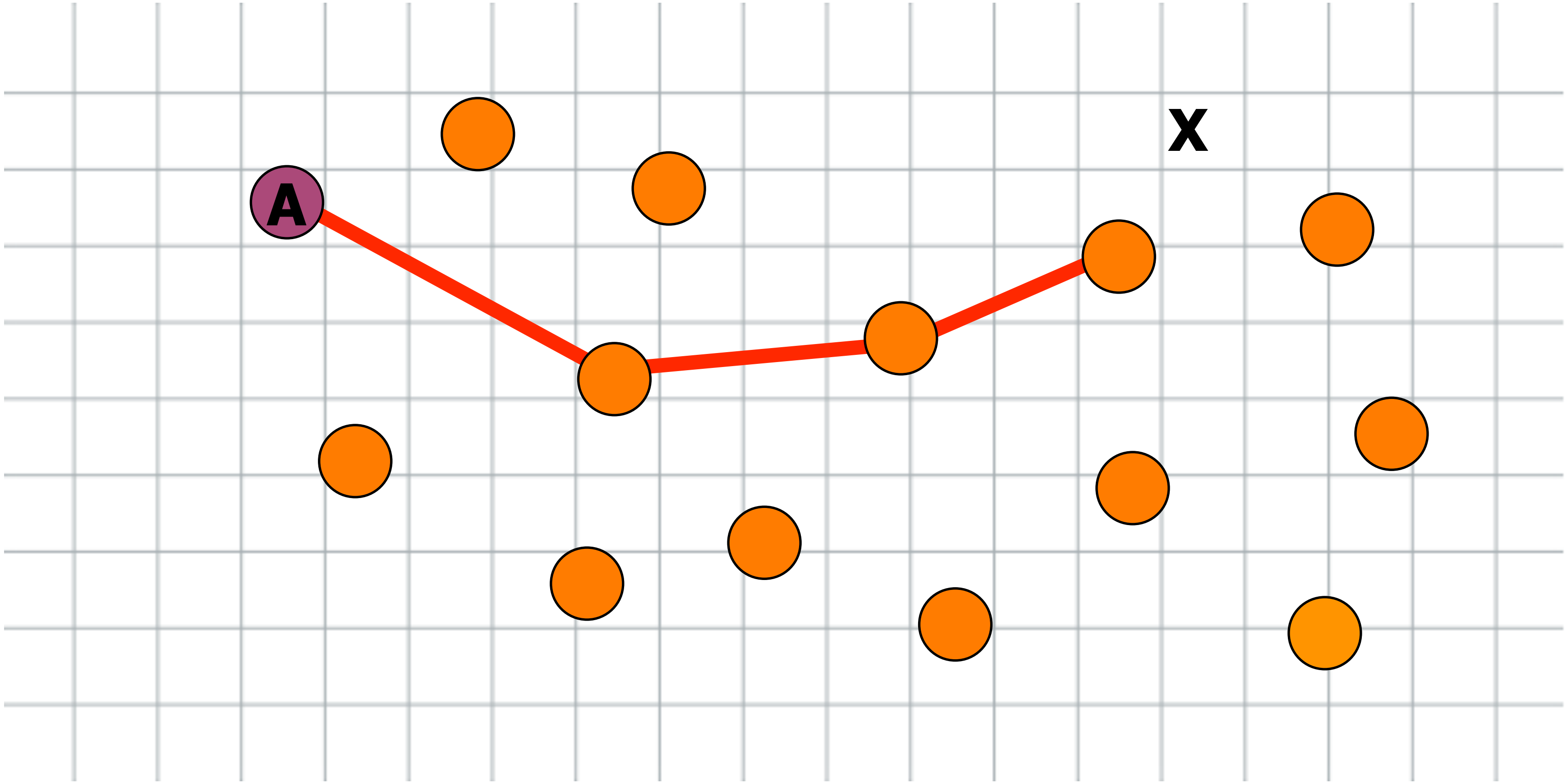
Application 1: NC Routing Substrate



Route message to overlay node location **X**

- Analogous to `route(key,msg)` in DHTs
- But routing path has low latency between **A** and **X**

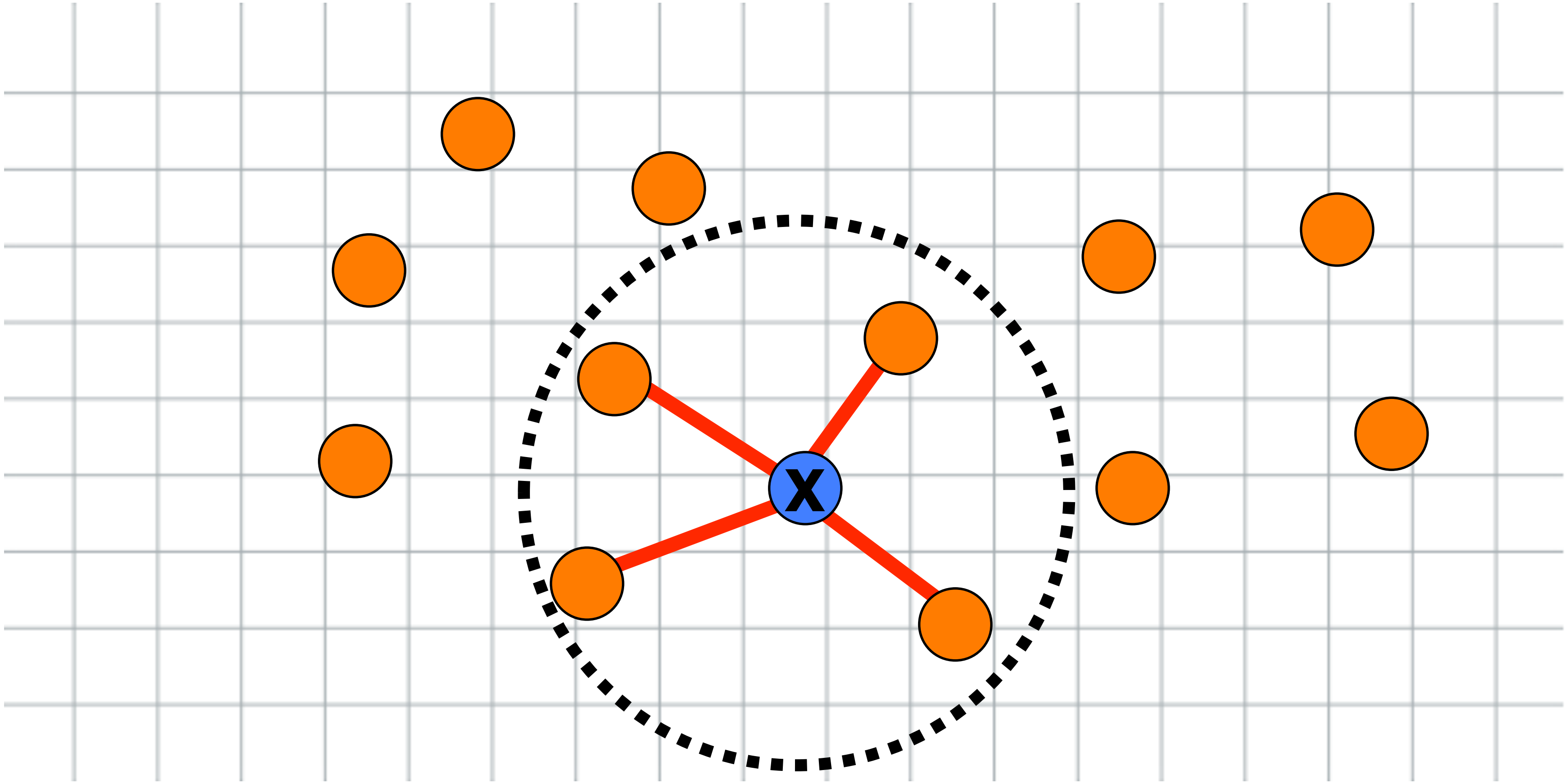
NC Routing: To Nearest Neighbour



Route message to closest existing overlay node

- Useful when location is external to overlay network
- e.g. finding closest web crawler to web server **X**

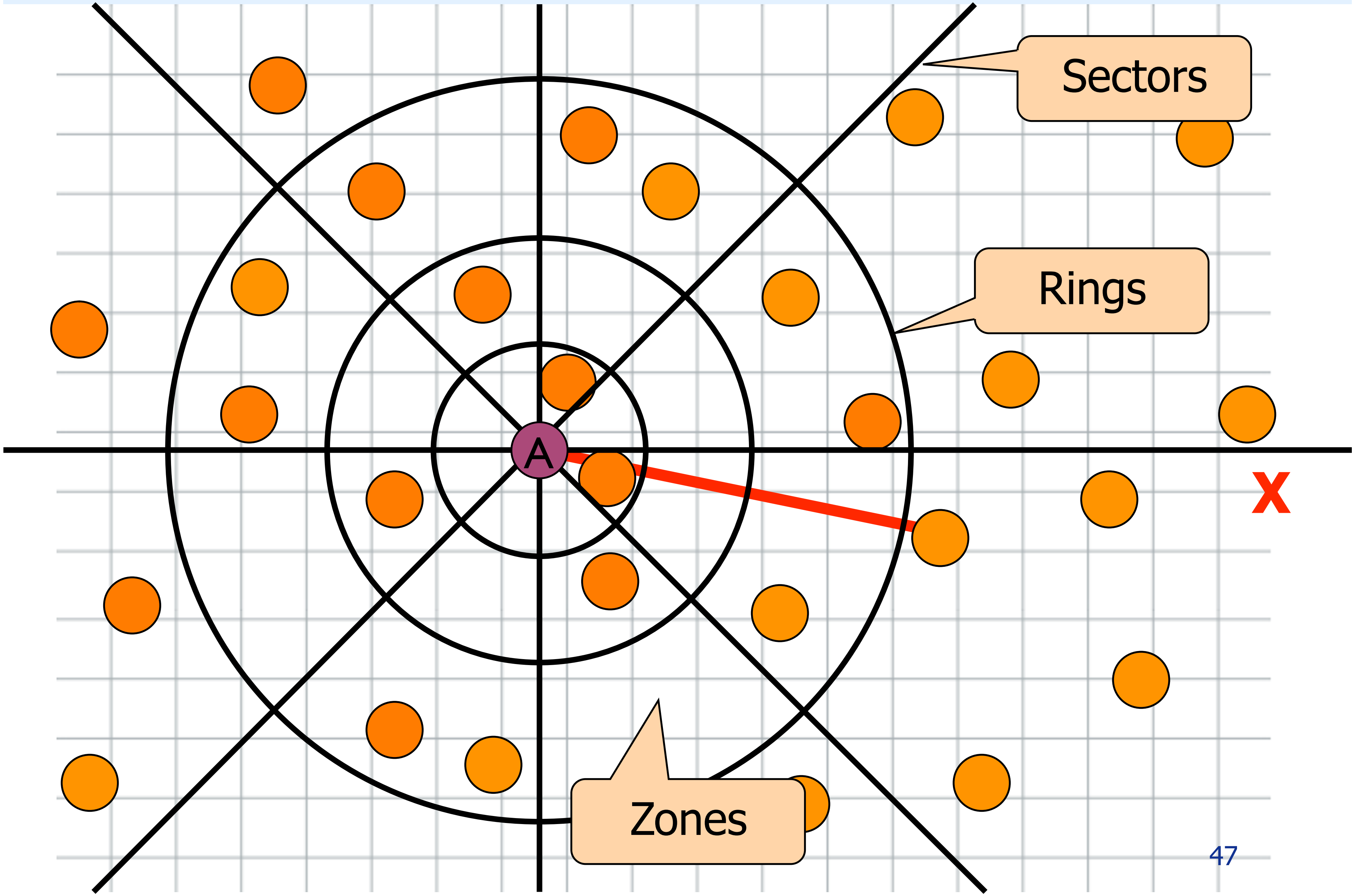
NC Routing: Local Broadcast



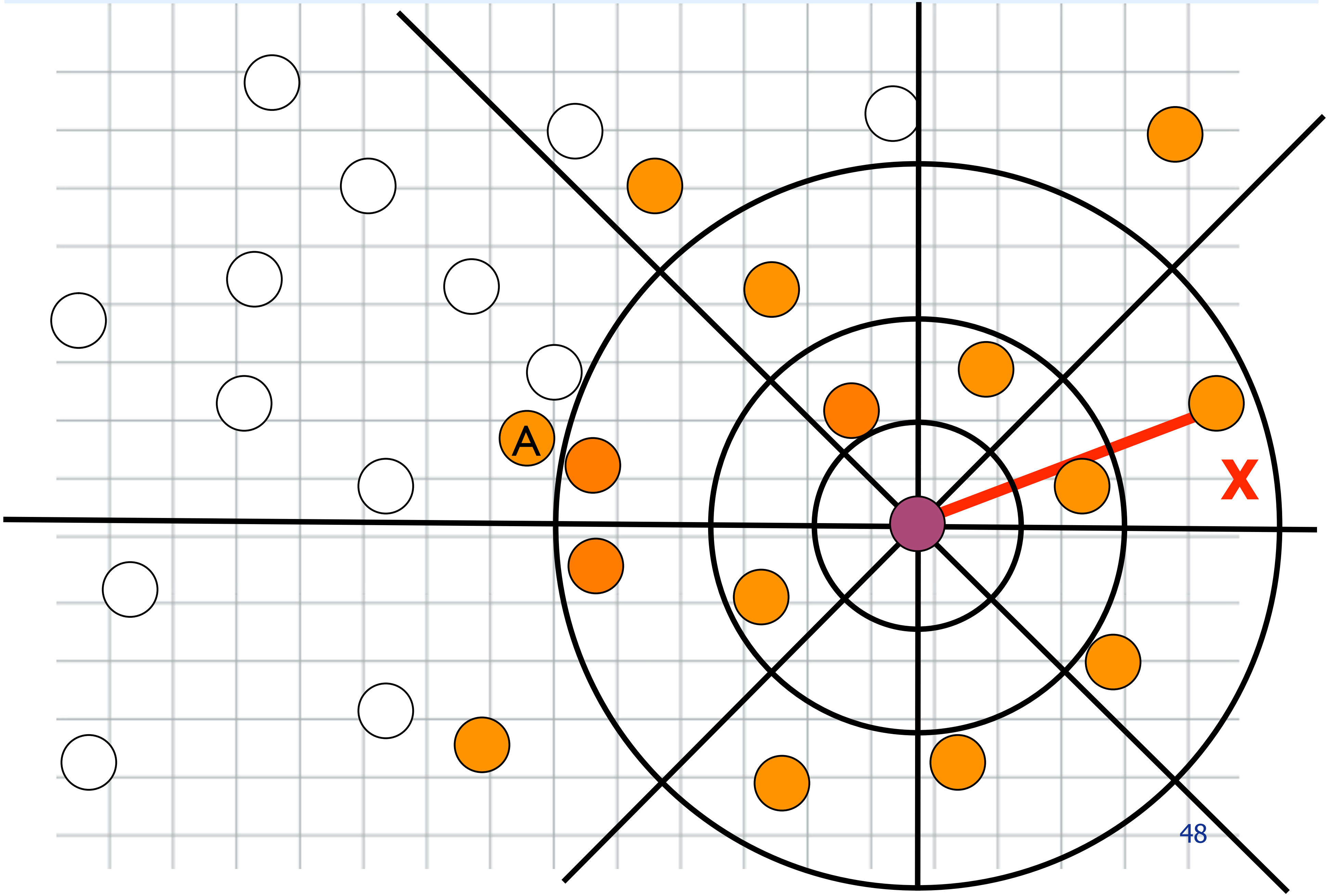
Finds nodes in neighbourhood

– e.g. replicate popular content across web caches

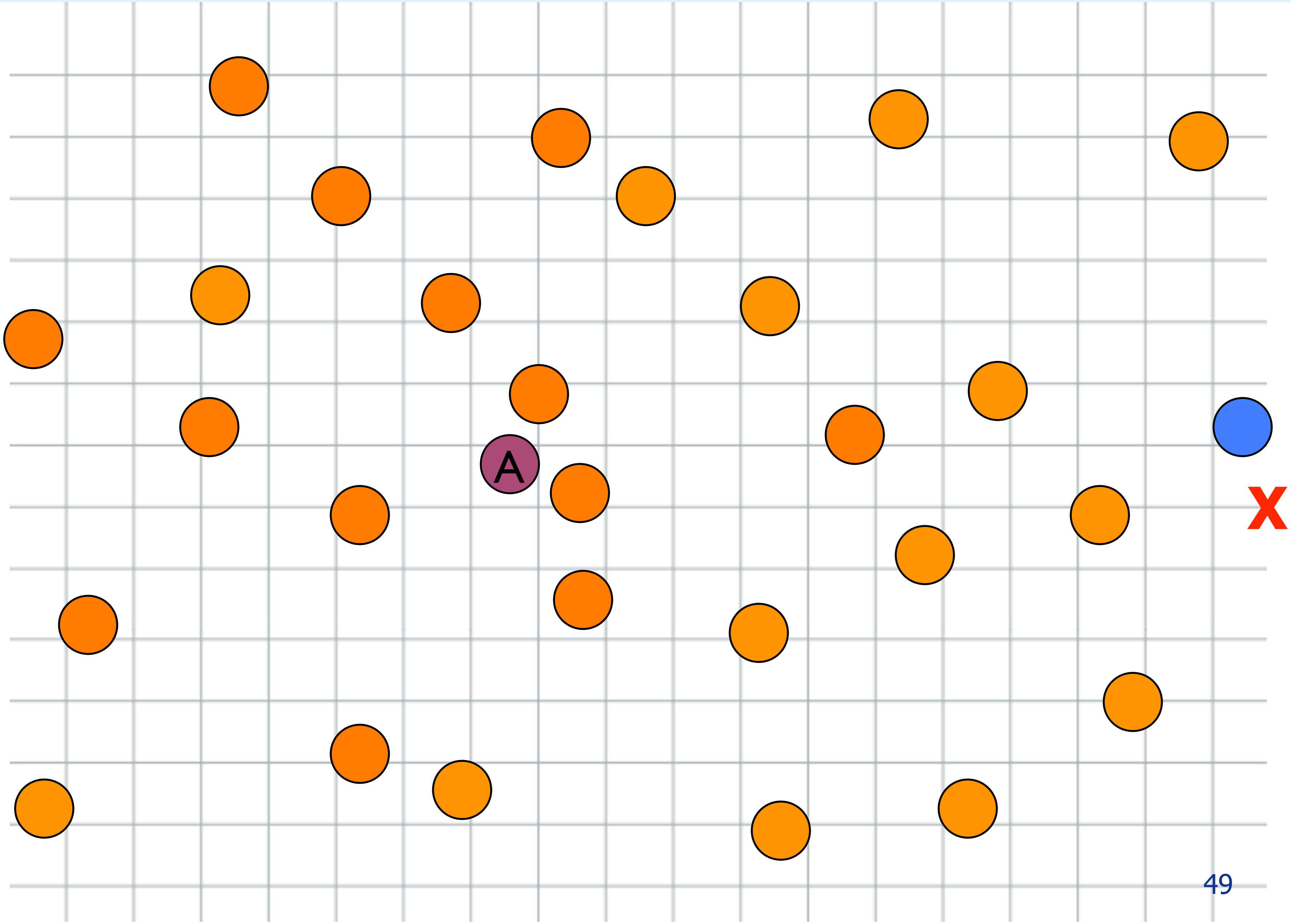
Scaled Theta Routing [Hassin01]



Scaled Theta Routing [Hassin01]



Scaled Theta Routing [Hassin01]

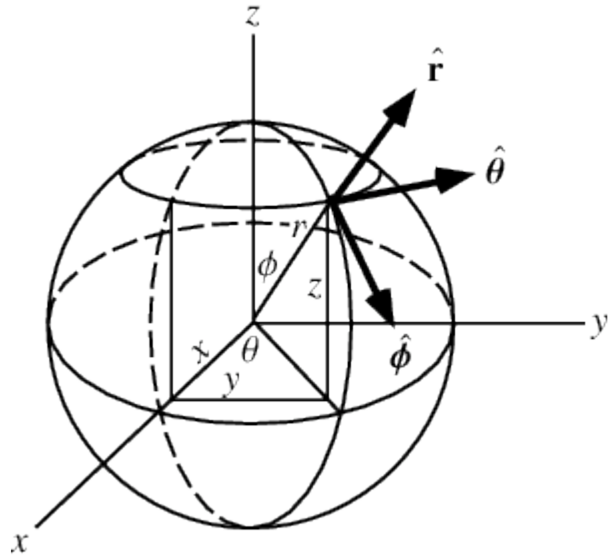


Practical Routing on Network Coordinates

From theory to practice

✓ Generalized k^d zone assignment

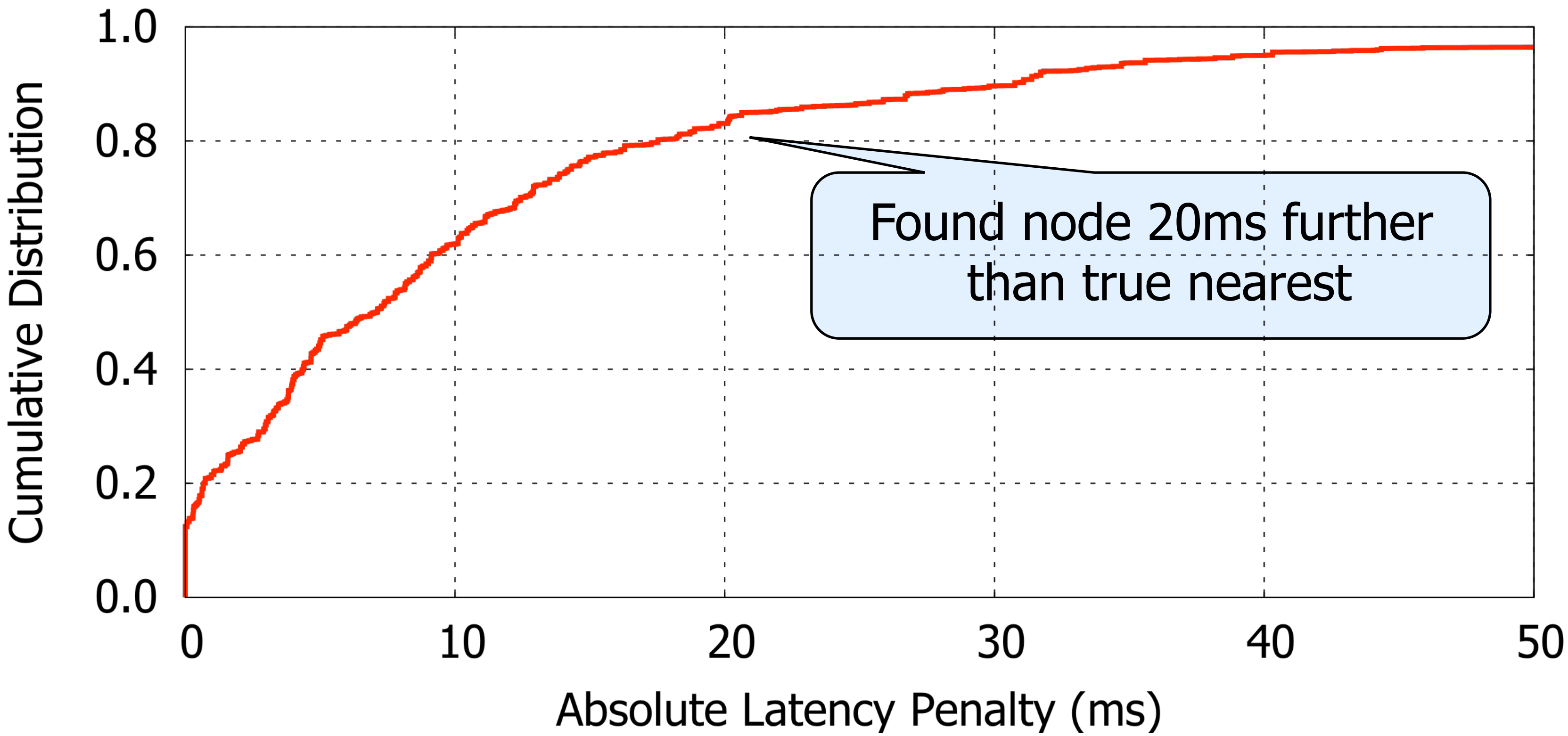
- Use hyperspherical coordinates: $\phi_0, \ldots, \phi_{d-1}$
- Each dimension "slices" sectors of prior dimensions



✓ Non-omniscient routing table formation

- New nodes need to build (good) routing tables
  - New nodes route message to own location
  - Collect routing tables along path
- Gossip mechanism to exchange routing tables

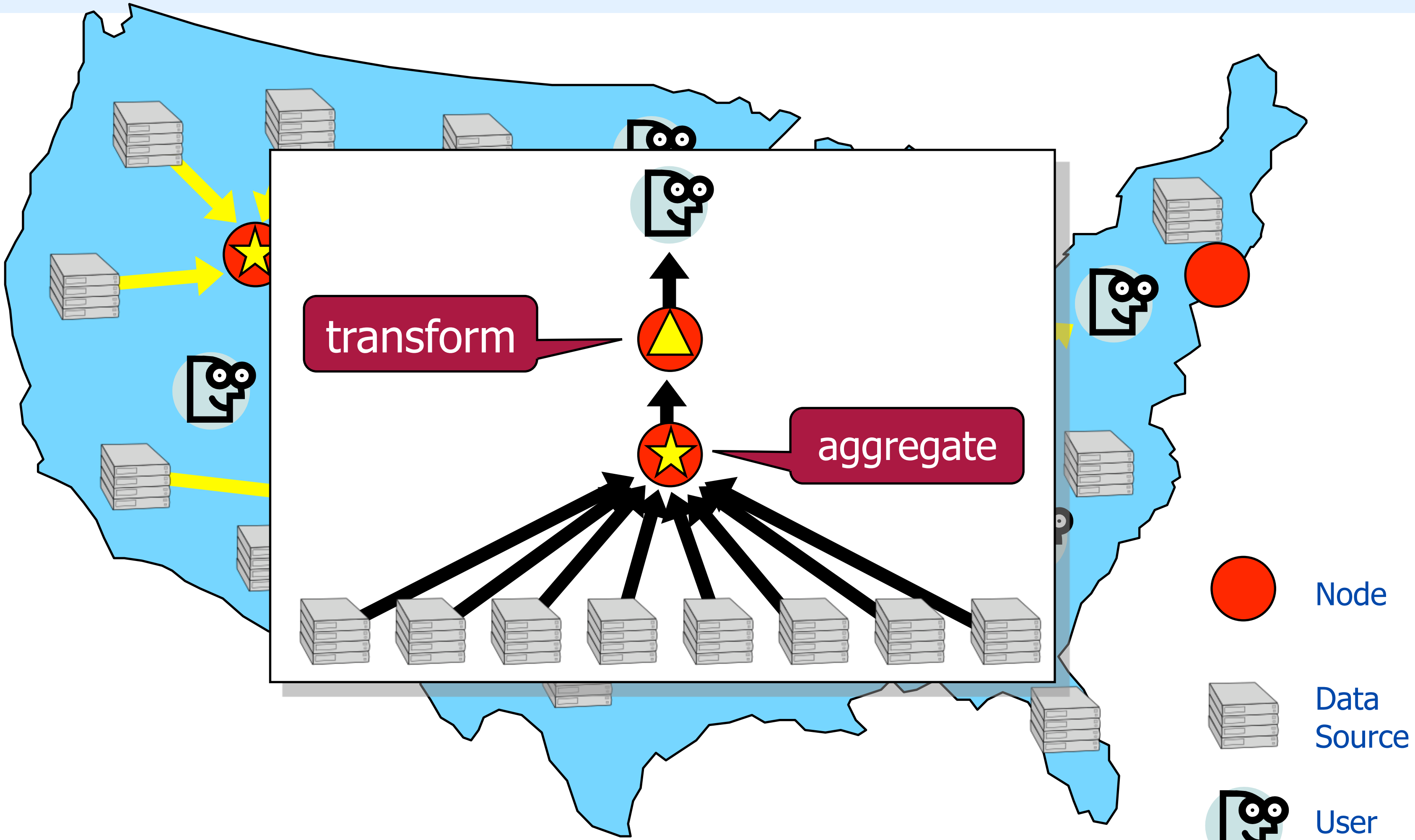
Evaluation: Nearest Neighbour



Nearest coordinate vs. true nearest neighbor

- 4d+h embed of MIT King data set (1740 DNS Servers)
- Designate 10% as targets
- Assigned "perfect" routing tables (rings=8; base=4; sectors=6)
- Find nearest coordinate (1/10000); thus: embed error dominates

Application 2: Operator Placement



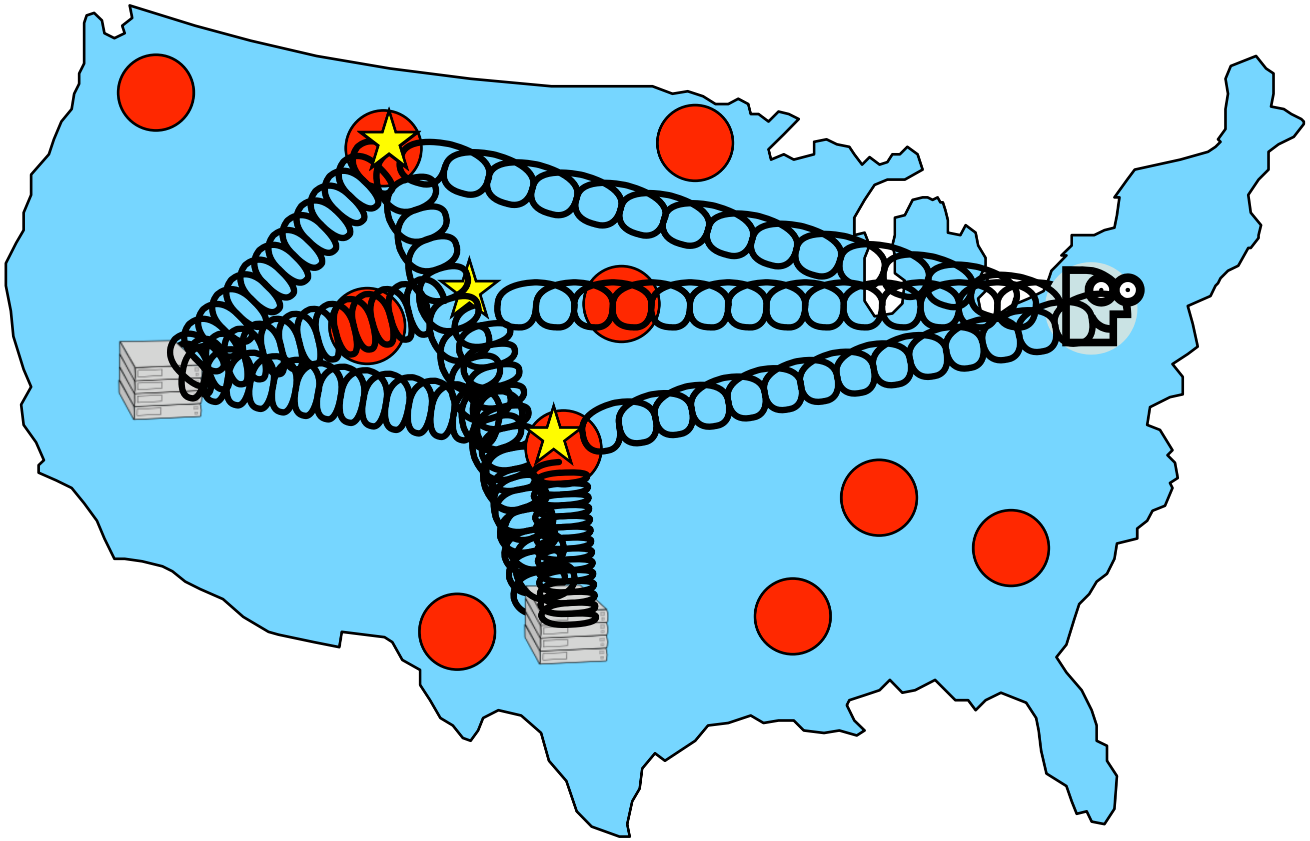
Find good placement for query operators

1. Load balance between processing nodes
2. Keep network traffic low and local

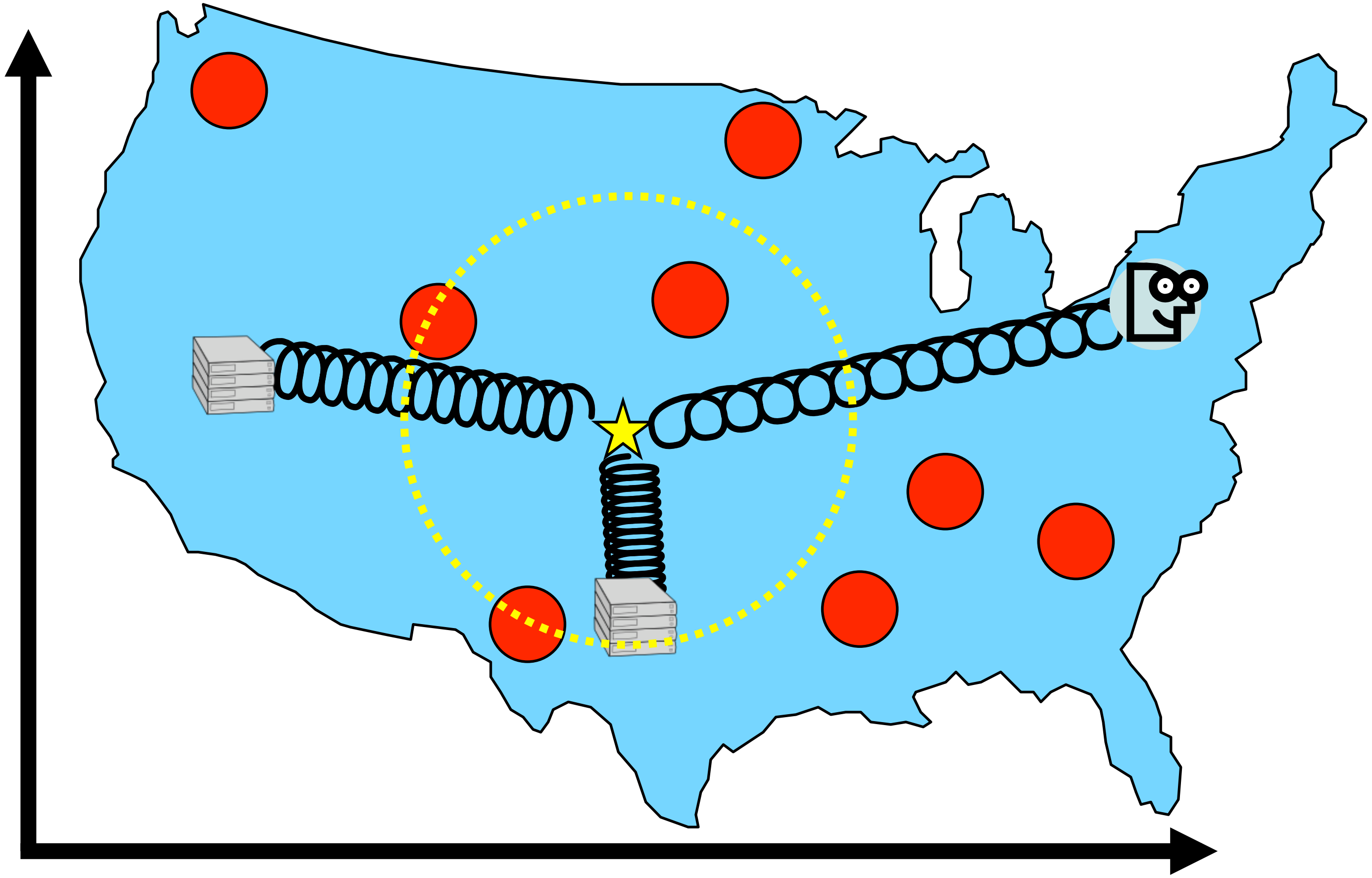
Network-Aware Operator Placement

Treat as decentralised optimisation problem

- Use approximation algorithm based on energy minimisation of springs



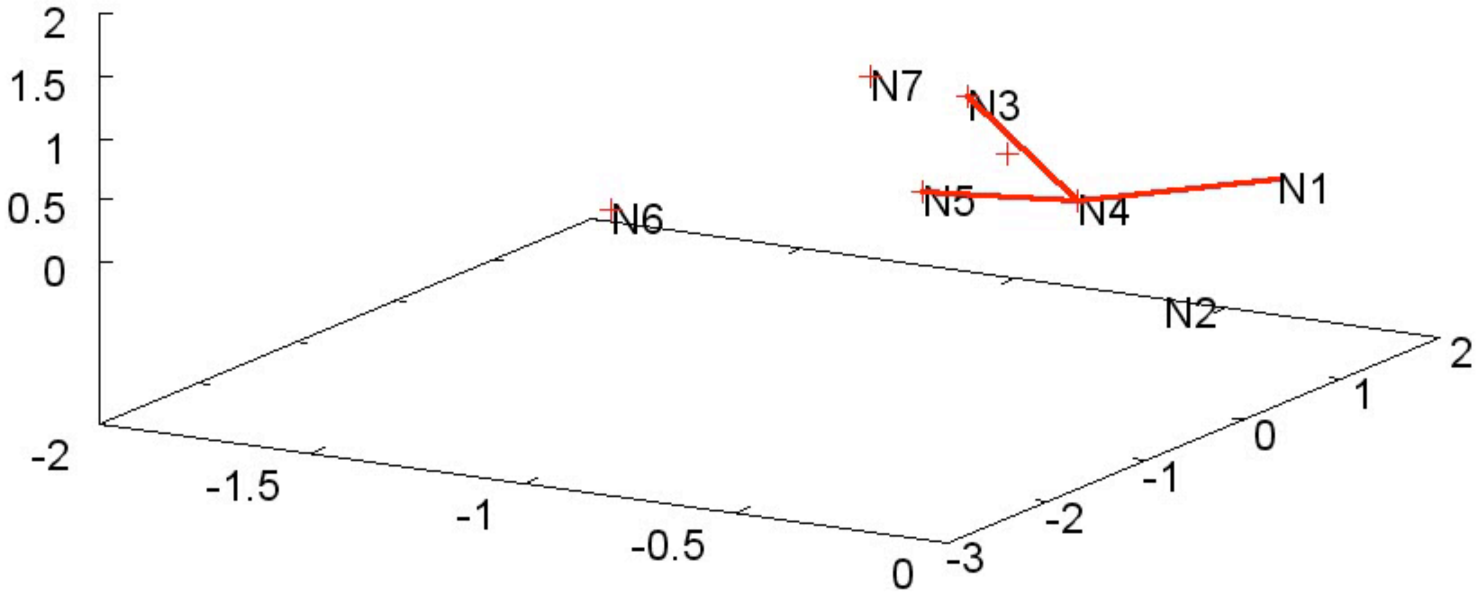
Relaxation Placement



Use k-nearest neighbor search for mapping of coordinates

– (see Application 1)

Video: Operator Migration

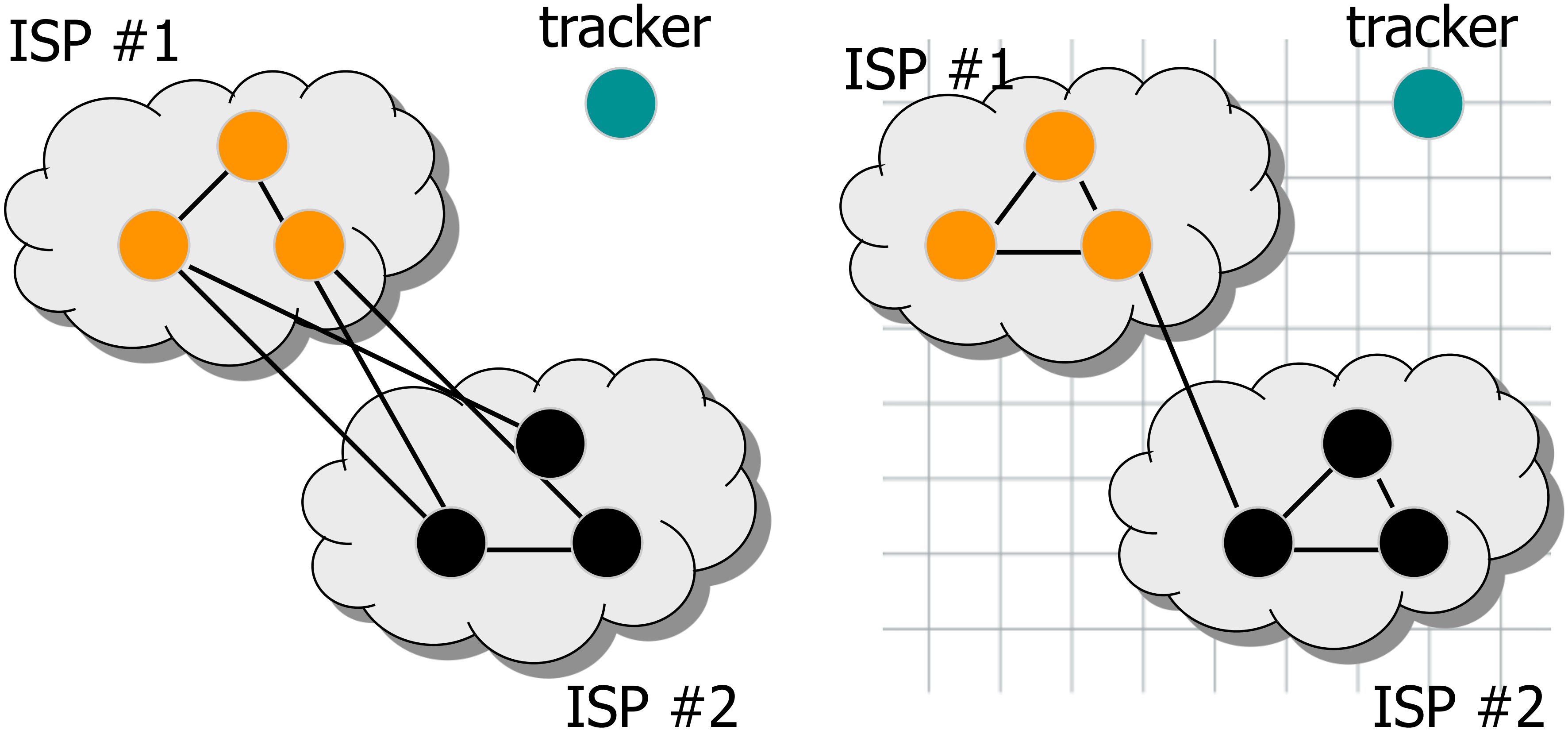


7 SBON nodes shown in latency space over several hours

– Query with one migrating aggregation operator

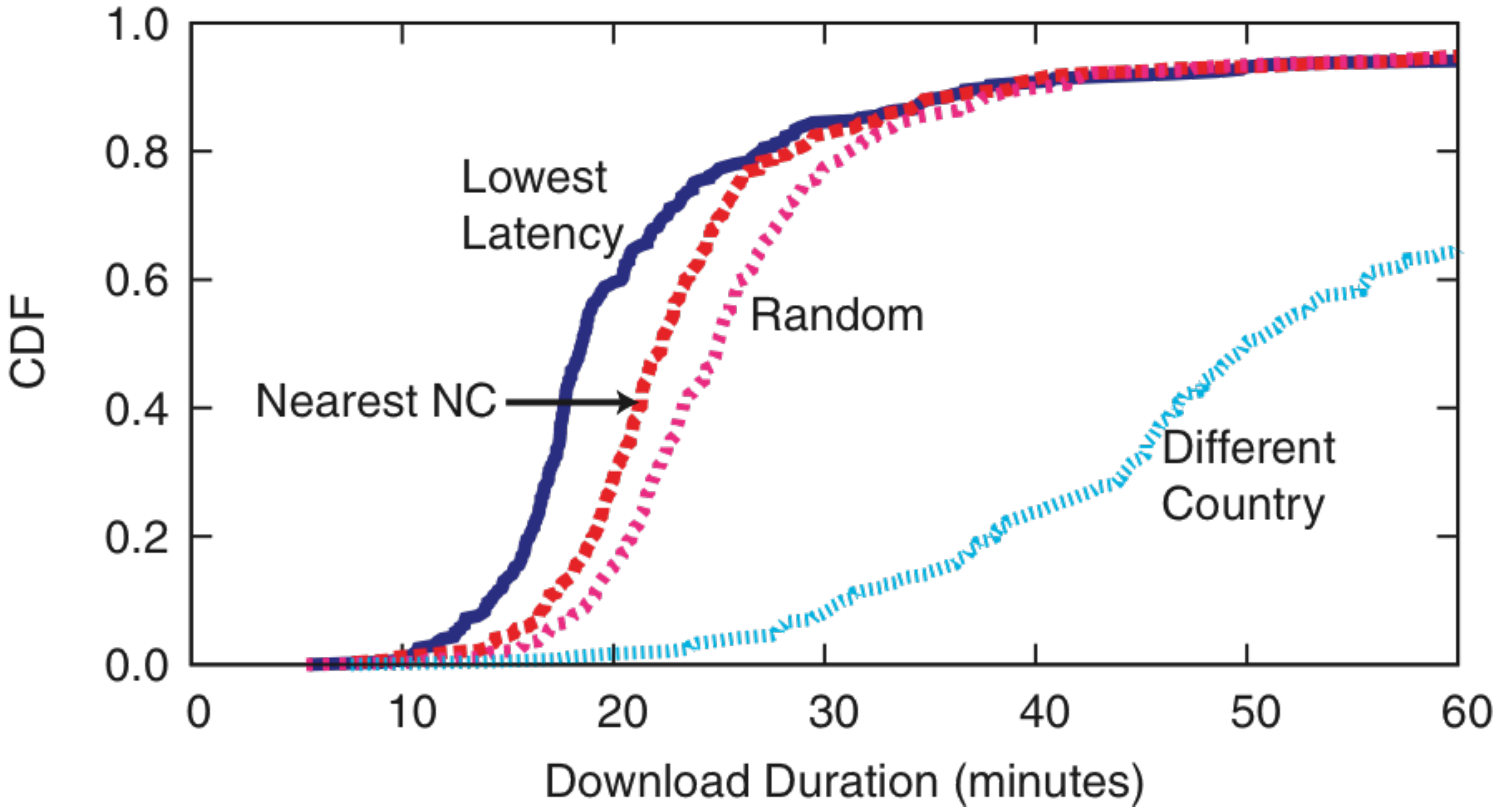
Application 3: Locality-aware Bittorrent

Unbiased swarms / **Locally-biased swarms**



Reduced inter-ISP traffic
Improved bandwidth for peers

Results: Locality-aware BT



328 PL nodes downloading 180Mb file using Azureus BT and modified tracker
26% median improvement with lowest latency peers
11% median improvement with nearest NC peers

Open Questions

What else can NCs be used for?

- Express distributed systems problems in geometric terms?
- Look at other metrics for measurements?

What do NCs reveal about network properties?

- PeerWise [HotNets'07]: use TIVs to identify mutually beneficial detours to reduce latency

Trade-off between reactive and proactive approaches for locality?

- Background NCs maintenance vs. active measurements

Conclusions

Locality-awareness becomes increasingly important for overlay applications
- Performance, network utilisation, ...

NCs reduce cost of network measurements
- But need to be practical in terms of accuracy and stability

NCs can be used to add locality-awareness to existing applications
- Bittorrent, stream-processing systems, …

NCs can provide geometric solutions to problems in large -scale distributed systems…

Shameless Plug

Interested?

Open PhD/post-doc positions to work on
Large-scale Distributed Systems at Imperial

Please tell your students/post-docs!
http://www.doc.ic.ac.uk/~prp/research

Thank you. Any Questions?

Peter Pietzuch
Department of Computing
Imperial College London
http://www.doc.ic.ac.uk/~prp
prp@doc.ic.ac.uk