

# Estimating Internet latency

Marco Slot ([marco@few.vu.nl](mailto:marco@few.vu.nl))

# Contents

Why care about latency?

Measuring internet latency

- Using existing infrastructure: cing

Predicting internet latency

- Centralized latency prediction: IDMaps, Triangles, GNP
- Decentralized latency prediction: Vivaldi

Conclusions

# Why care about latency?

High latency annoys the user

High latency handshaking decreases performance

Latency-sensitive applications:

- Web
- Instant messaging
- Online games
- Peer-to-peer routing
- ...

Typical ranges: [0-100], [100-250], [250-500], [500-\*] ms

# Why care about latency?

Why do we need latency information?

- Replica placement
- Server selection
- Peer selection
- Route optimization
- ...

End-to-end delay not always sufficient knowledge

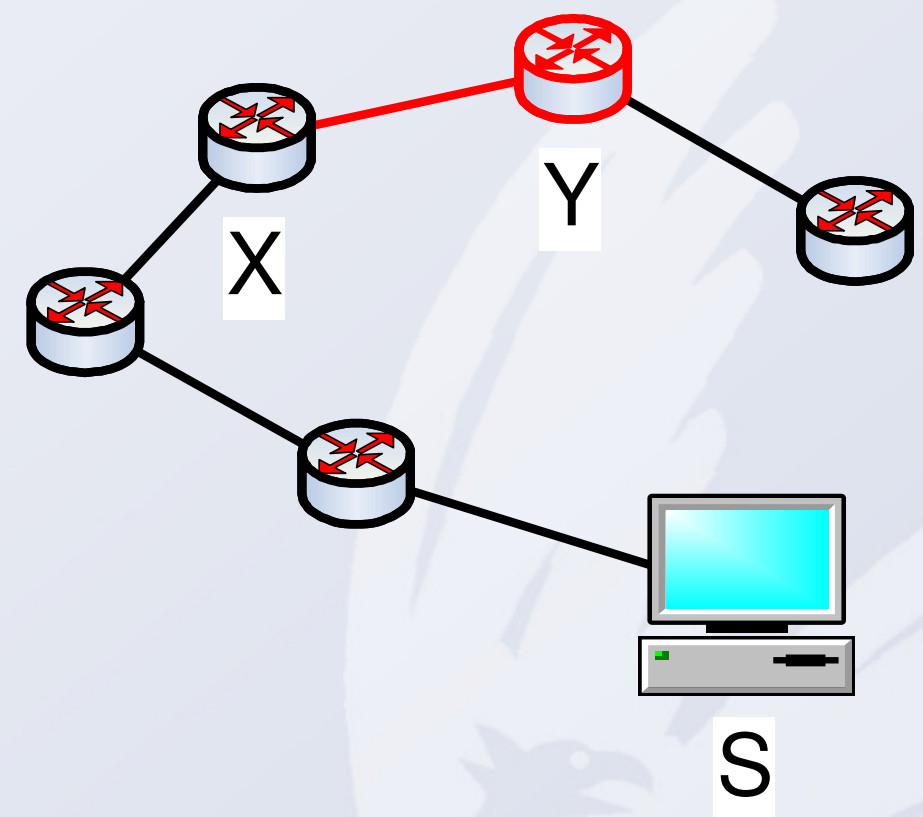
We often want to know latency between 2 other hosts

Measuring is often too expensive

# Cing: Measuring network-internal latency

Measure latency of a single link  $X \rightarrow Y$  from node  $S$

- Assume path  $S \rightarrow X$  is a prefix of  $S \rightarrow Y$



Use only existing infrastructure

- ICMP Timestamp
- ICMP TTL

(supported by 93% of routers, 38% of full paths)

# Cing: Measuring links

Send back-to-back timestamp messages to X and Y

Latency estimation of  $X \rightarrow Y$ :

$$\delta = t_y - t_x = t_{\text{queuing}} + t_{\text{link}} + \text{offset}_{x,y} \approx t_{\text{queuing}} + \text{offset}_{x,y}$$

Assume: Clocks run at same speed

Assume: Link delay is close to 0

Assume: Queuing delay is sometimes close to 0

Repeat experiment  $m$  times. Take minimum  $\delta$  (close to 0).

$$\text{Latency}_{\langle x,y \rangle} \approx \delta_{\text{current}} - \delta_{\text{min}}$$

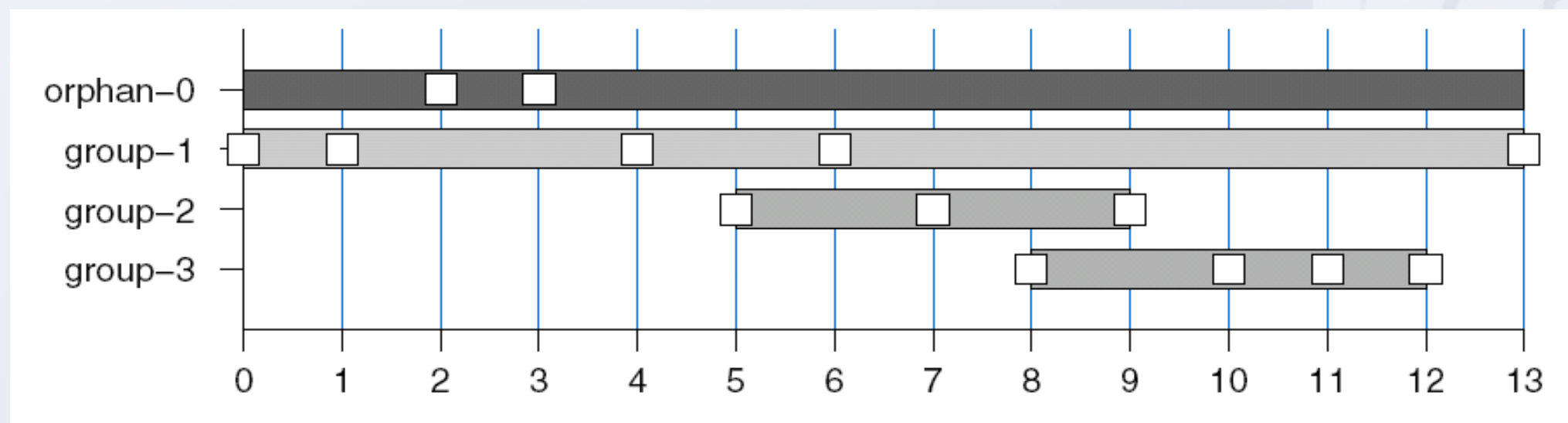
# Cing: Measuring paths

Get path to destination D using `traceroute`

Get path to each node in path using `traceroute`

Distinguish between groups in which for any pair X,Y:

- $S \rightarrow X$  is a prefix of  $S \rightarrow Y$
- $X \rightarrow Y$  lies in the path to D



# Cing: Summary

## Cing conclusions:

- Existing router infrastructure can be used to measure specific network links and paths
- Cing could be accurate when assumptions hold

## Problems:

- Many highly questionable assumptions
- Poor results
- Accuracy not very clear

# Predicting internet latency

## Problem:

Measuring latency is expensive for latency-sensitive applications

## Solution:

Heuristic for estimating latency using relative distances

## Possible models:

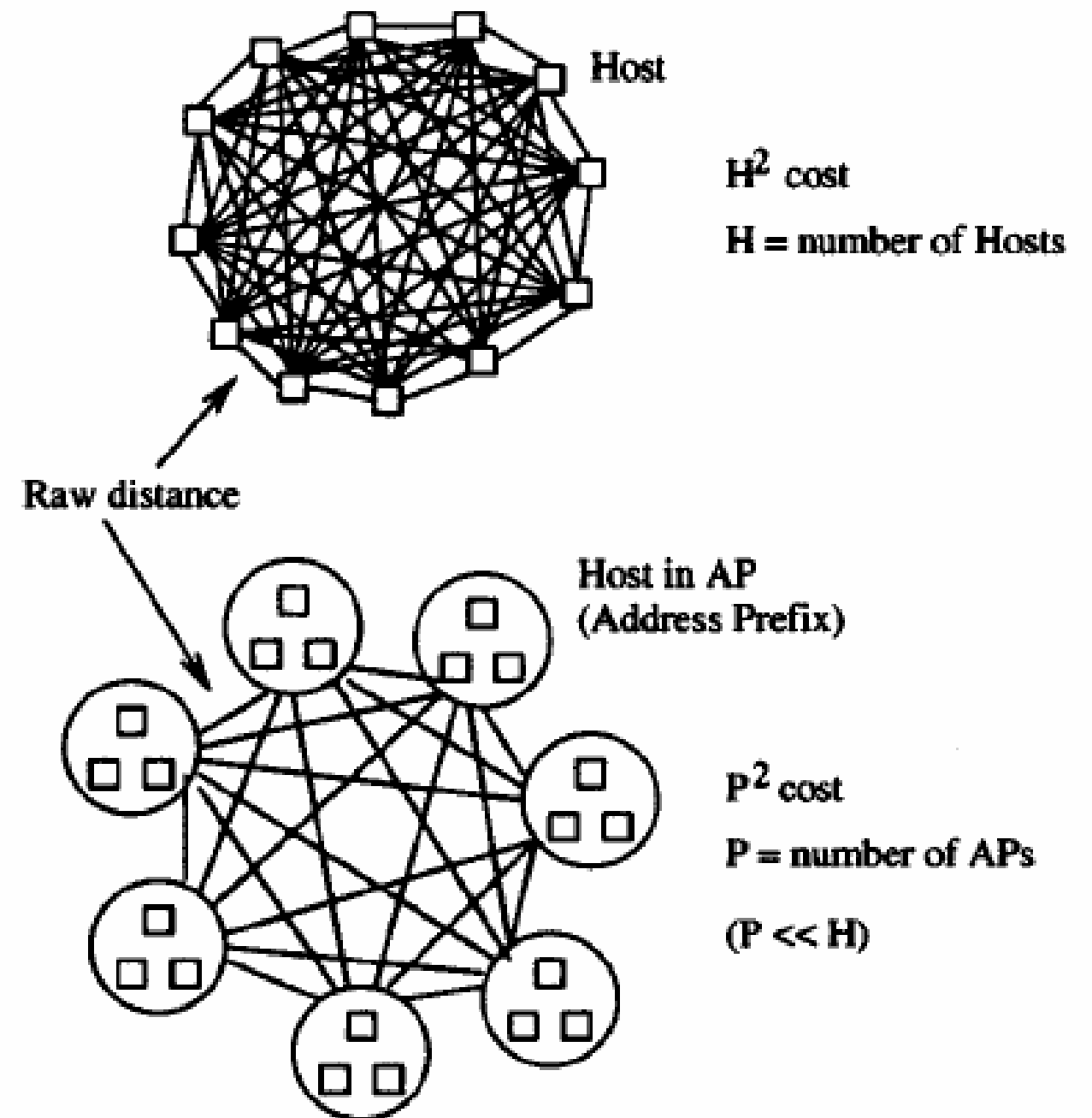
- Topology map
- Triangles
- Euclidean space
- ...

# IDMaps: Topology map

Idea: Group nearby nodes

- Choose  $n$  'tracer' hosts
- Measure latency between all pairs of tracers
- Host measures latency with nearest tracer

$$d(A, B) = d(A, T_1) + d(T_1, T_2) + d(T_2, B)$$



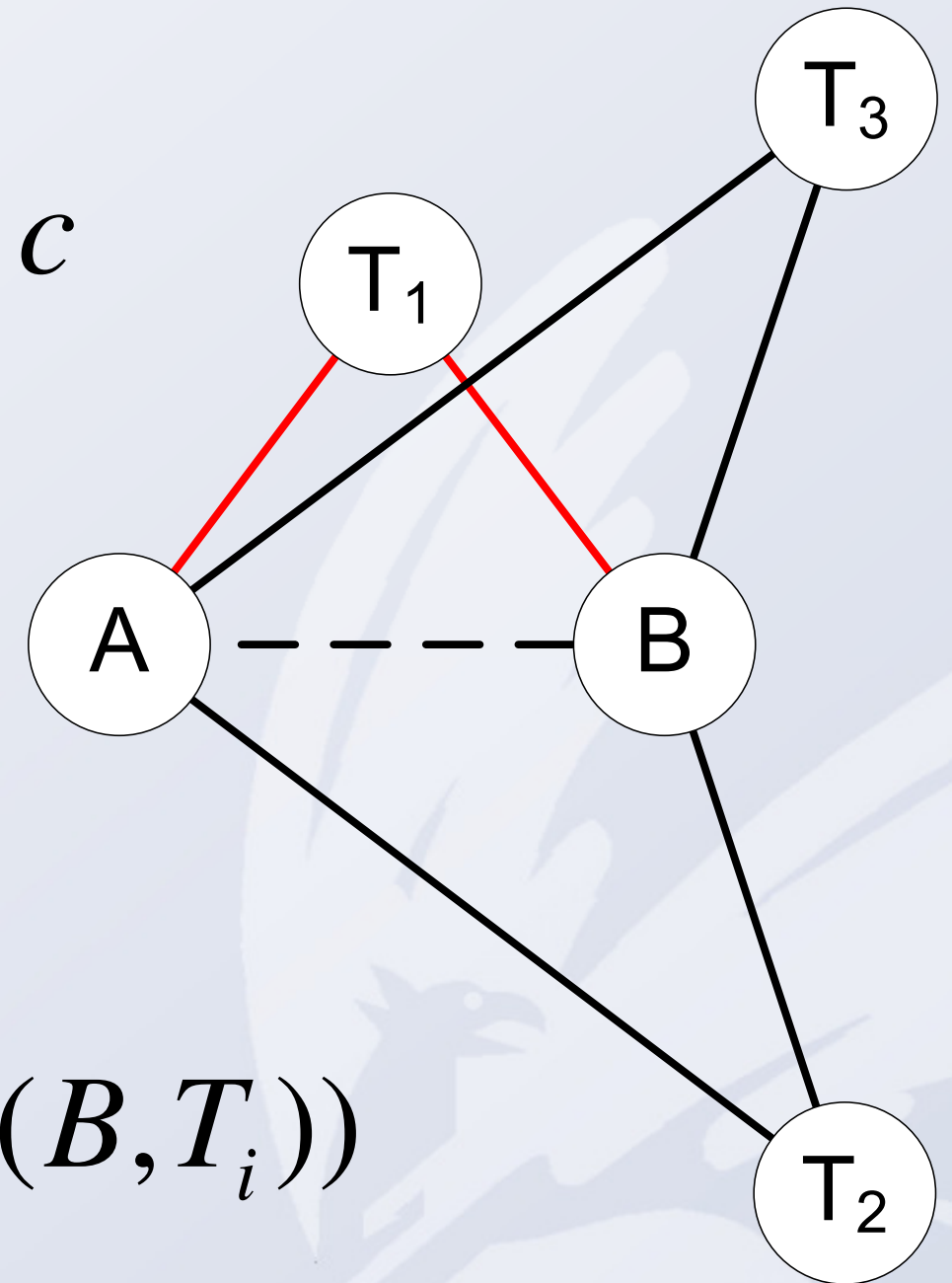
# Triangulated heuristic

Idea: Use triangular inequality

$$|b - c| < a < |b + c| \text{ in triangle } a, b, c$$

Latency from each base node gives set of relative coordinates.

$$d(A, B) = \min_{i \in \{1, 2, \dots, N\}} (d(A, T_i) + d(B, T_i))$$



# Global Network Positioning

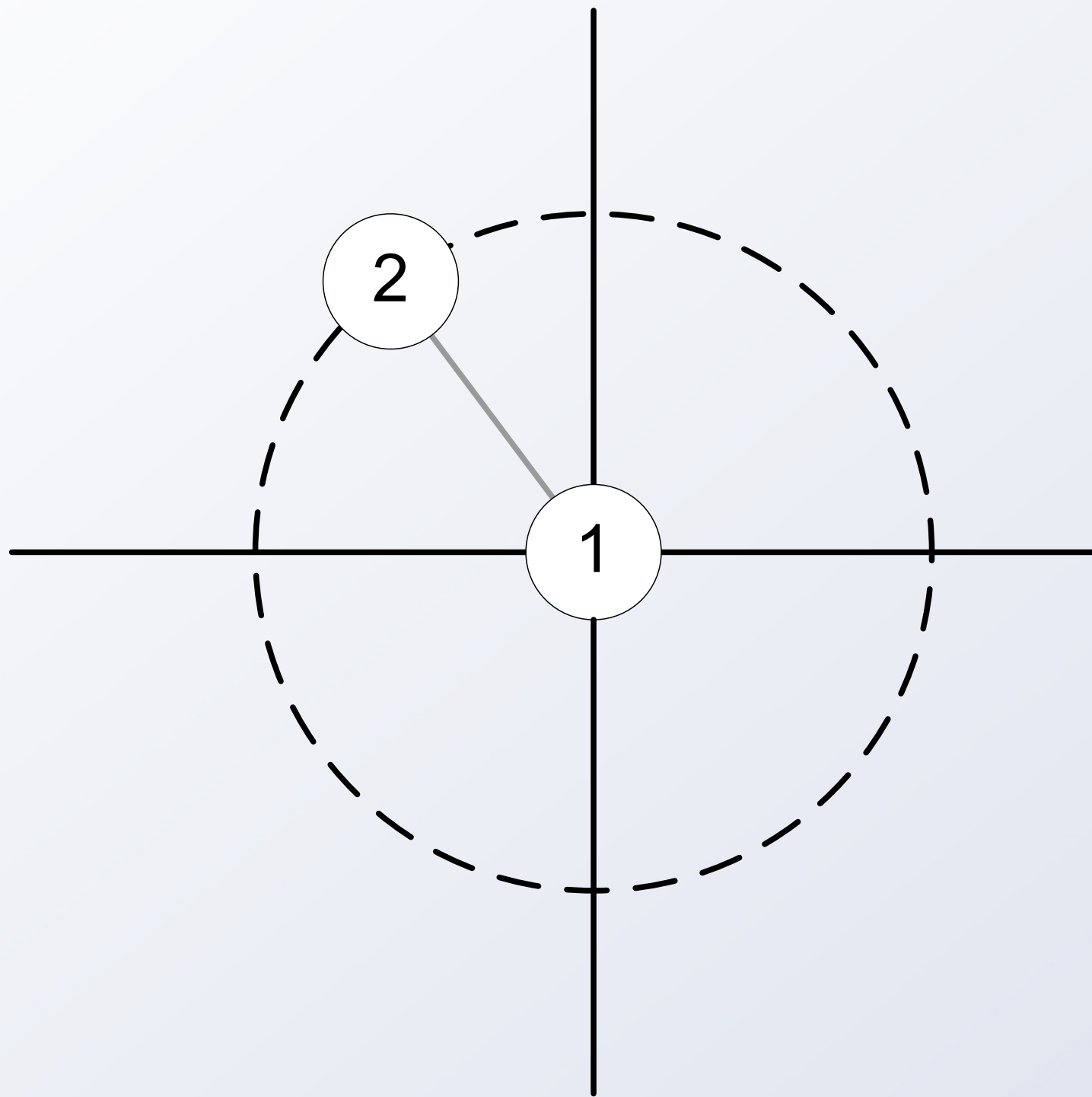
Idea: Give nodes n-dimensional Euclidean positions

Nodes measure distance to base nodes and choose the position with the smallest overall error

Latency prediction is distance between coordinates A, B:

$$d(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

# Global Network Positioning: Landmarks

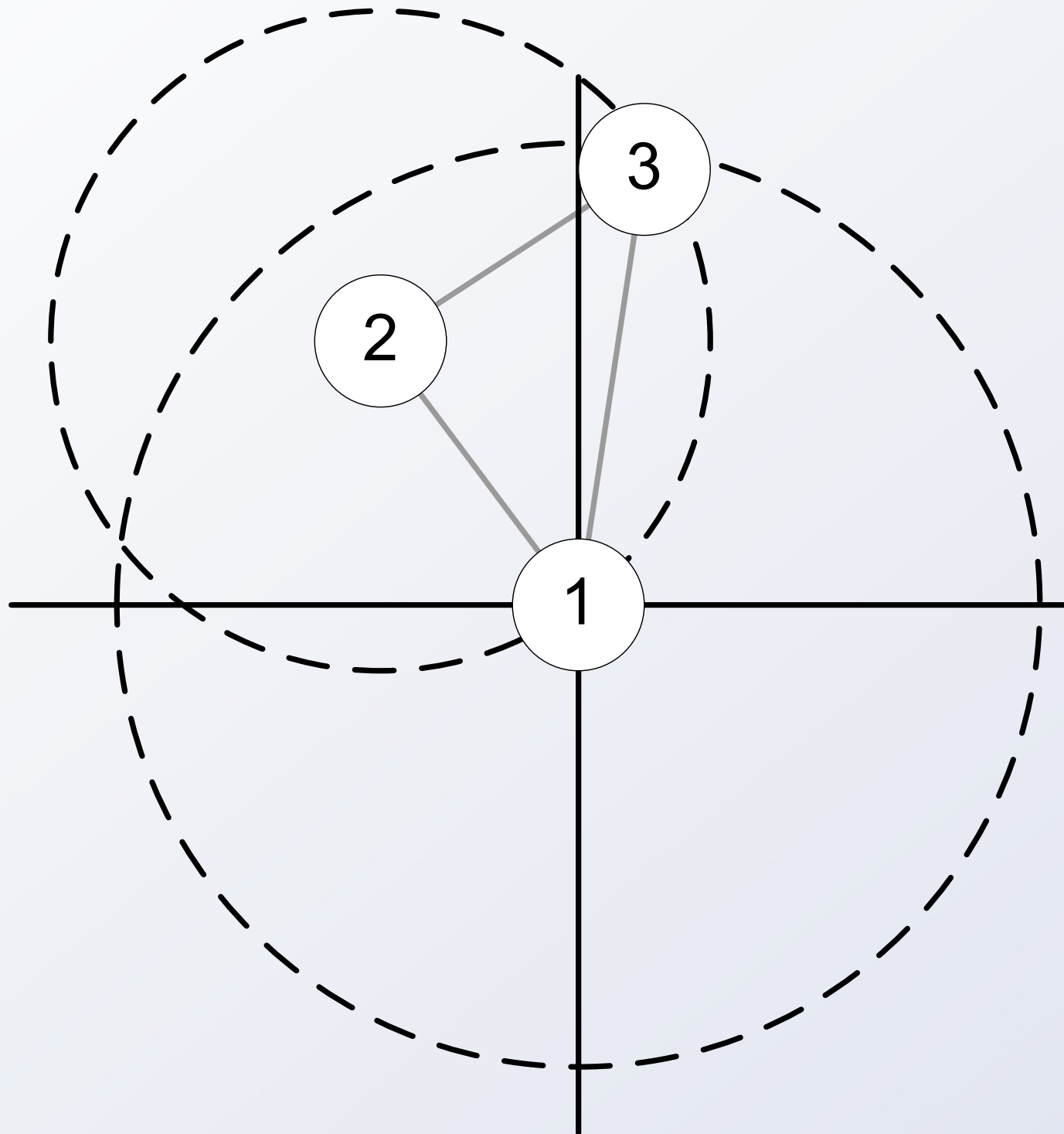


Landmark initialization:

Place 1 anywhere

Place 2 at distance  $d(T_1, T_2)$

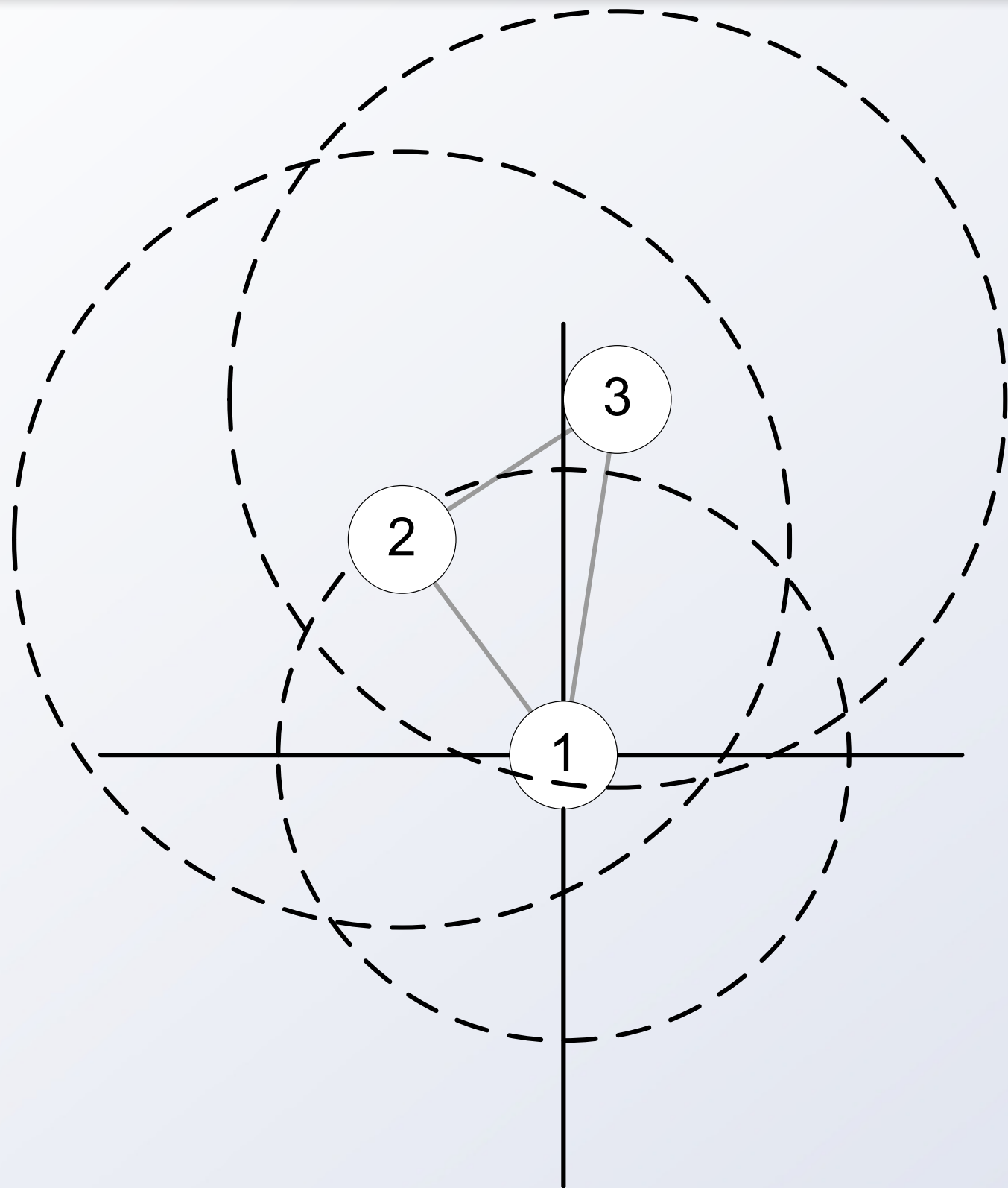
# Global Network Positioning: Landmarks



Landmark initialization:

Place 3 at  
distance  $d(T_2, T_3)$  from 1 and  
distance  $d(T_1, T_3)$  from 2

# Global Network Positioning: Landmarks



Landmark initialization:

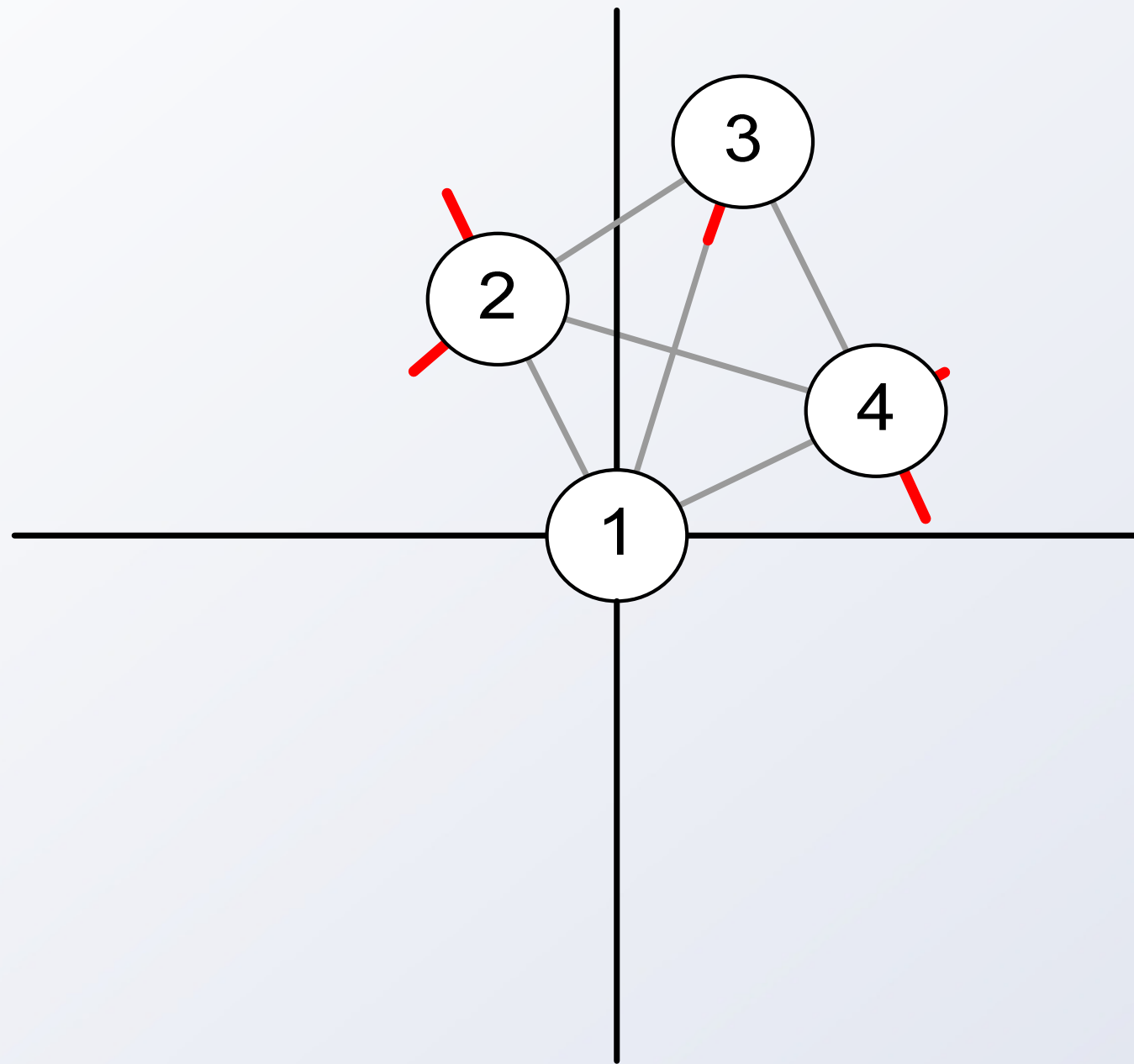
No correct position for fourth landmark in 2 dimensions

Add another dimension

or

Choose positions to minimize overall error

# Global Network Positioning: Landmarks



Landmark initialization:

Minimize the function

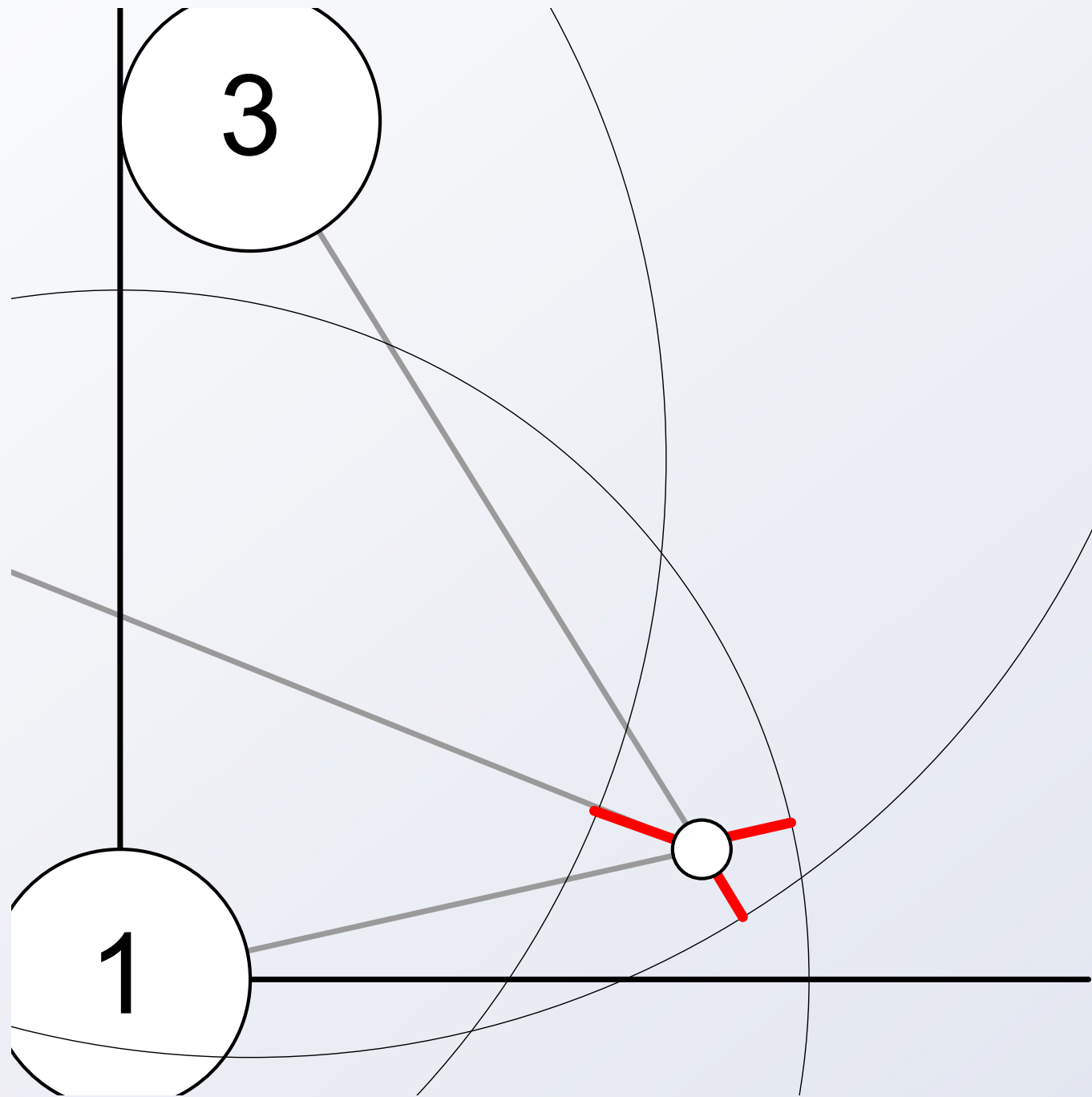
$$\sum_{T_i, T_j \in \{T_1, T_2, \dots, T_N\} | i > j} \varepsilon(d(T_i, T_j), d_S(T_i, T_j))$$

with some error function, e.g.

$$\varepsilon(d_1, d_2) = \left( \frac{d_1 - d_2}{d_1} \right)^2$$

Use solver such as Simplex  
Downhill

# Global Network Positioning: Ordinary host



Find position for A:

Minimize function

$$\sum_{i \in \{1, 2, \dots, N\}} \mathcal{E}(d(T_i, A), d_s(T_i, A))$$

with some error function, e.g.

$$\mathcal{E}(d_1, d_2) = \left( \frac{d_1 - d_2}{d_1} \right)^2$$

Use solver such as Simplex  
Downhill

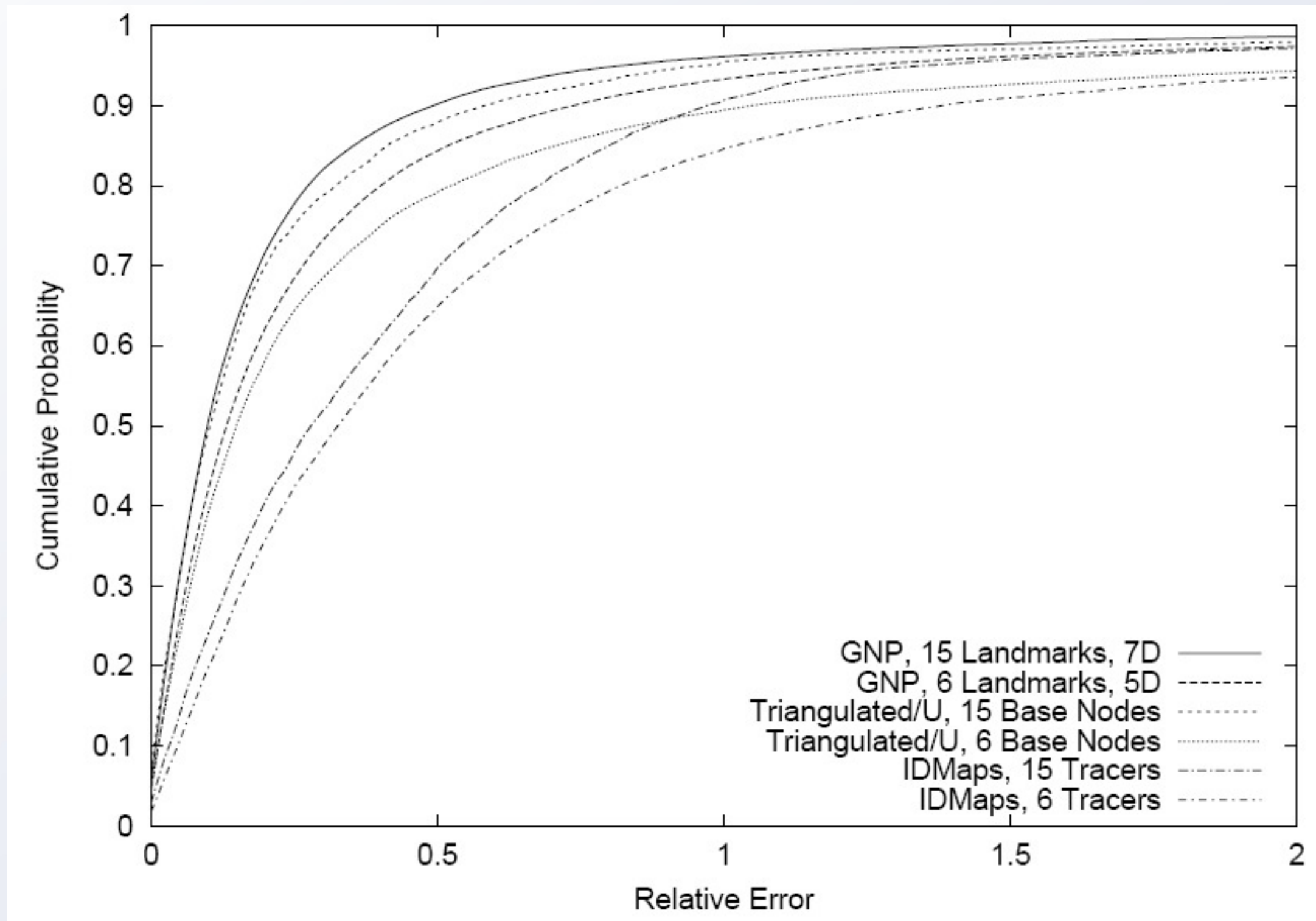
# Global Network Positioning: Design options

Various design options:

- Error function: logarithmic, **normalized**, squared,
- Minimization function: **simplex downhill**, ...
- Geometric space: **flat**, spherical, cylindrical, ...
- Dimensions: 2, 3, **4**, **5**, **6**, **7**, ...
- Nr. Of landmarks: 3, 4, **5**, **6**, **7**, **8**, ...
- Landmark placement: max sep, **cluster-medians**, ...
- Landmarks used in measurement

Best choice largely depends on application

# Global Network Positioning: Accuracy



# Global Network Positioning: Summary

## GNP conclusions:

- GNP outperforms existing latency prediction systems
- Many configuration options to increase accuracy
- Euclidean space model works surprisingly well

## Problems:

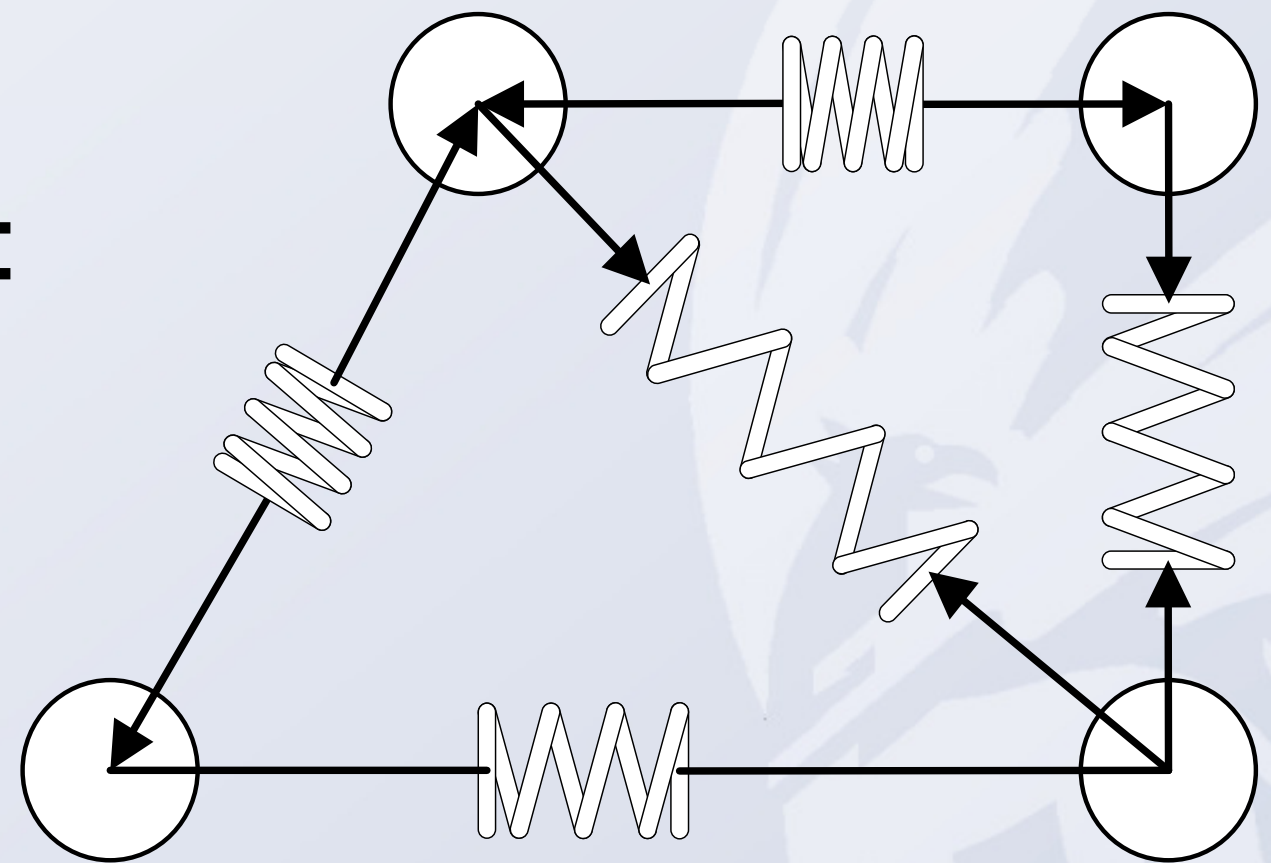
- Network infrastructure is assumed to be fixed
- Centralized landmarks may become bottlenecks

# Vivaldi: Decentralized latency prediction

Vivaldi is a distributed algorithm for latency prediction.

Nodes choose a position in geometric space and periodically exchange position and round-trip-time with a small set of neighbours.

Distributed error minimization:  
**Iterative spring system**



# Vivaldi: Simple algorithm

## Simple Vivaldi algorithm for pulling the strings

```
var myCoor
simple_vivaldi(rtt, hisCoor) {
  error      = |rtt - |myCoor - hisCoor||
  force      = error * direction(myCoor, hisCoor)
  myCoor     = myCoor + DELTA * force
}
```

# Vivaldi: Advanced Algorithm

Also take accuracy into account

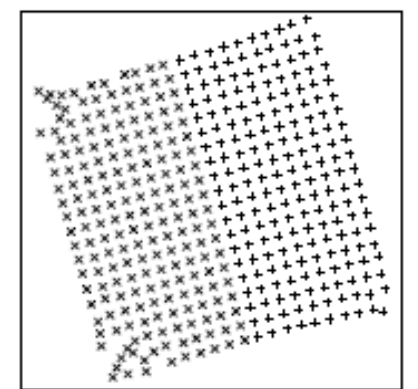
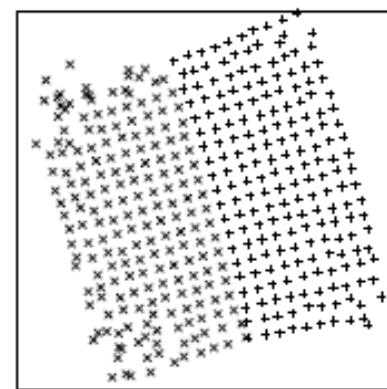
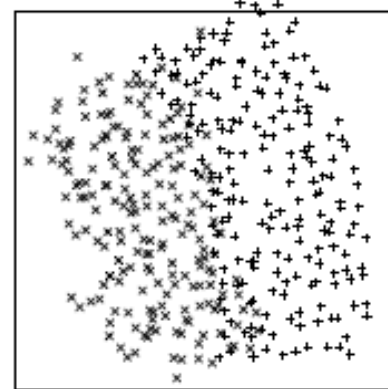
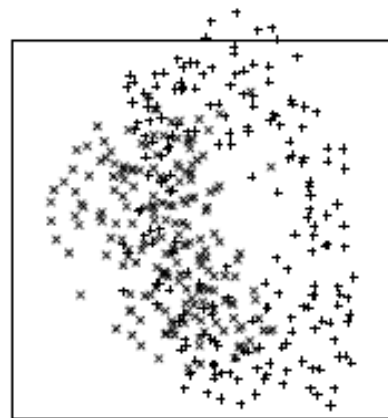
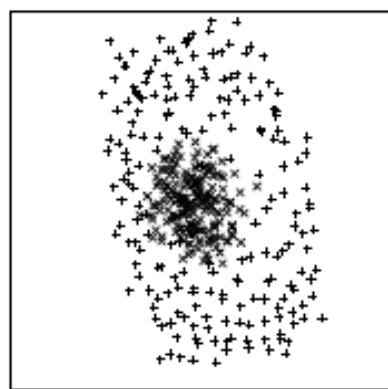
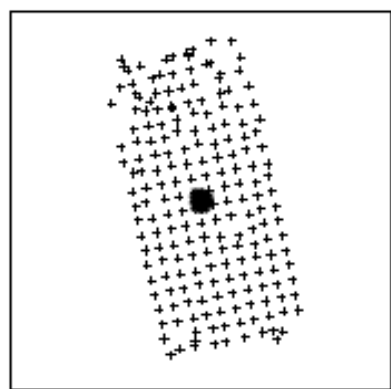
```
var myCoor, myError
vivaldi(rtt, hisCoor, hisError) {
  error      = |rtt - |myCoor - hisCoor||
  force      = error * direction(myCoor, hisCoor)
  weight     = myError / (myError + hisError)
  myCoor     = myCoor + DELTA * weight * force
  myError    = WMA(C * W, error / rtt, myError)
}
```

Again many different design choices

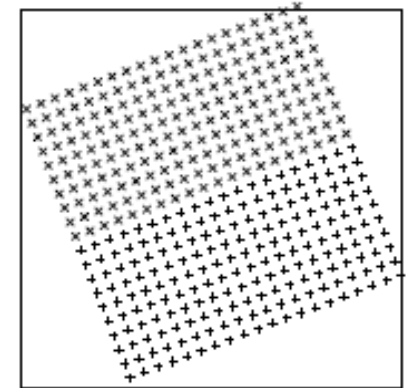
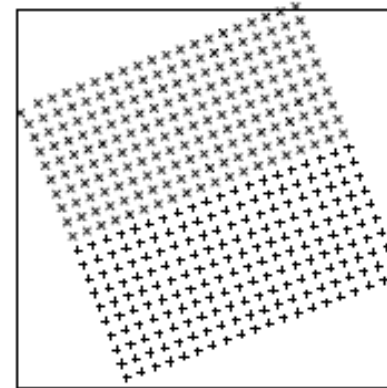
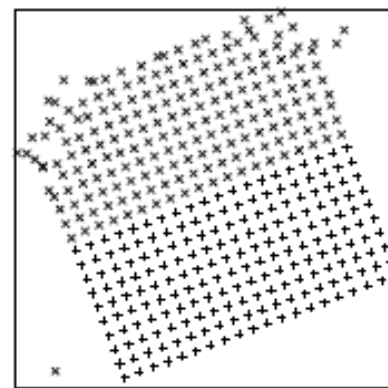
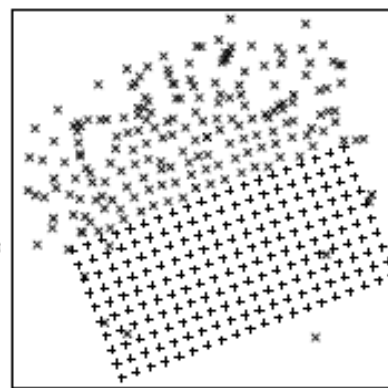
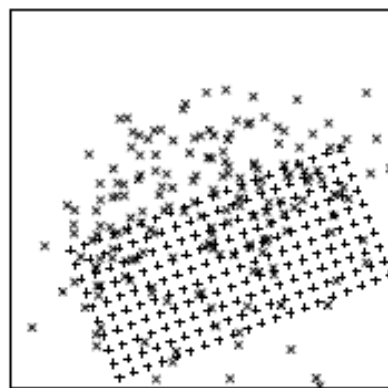
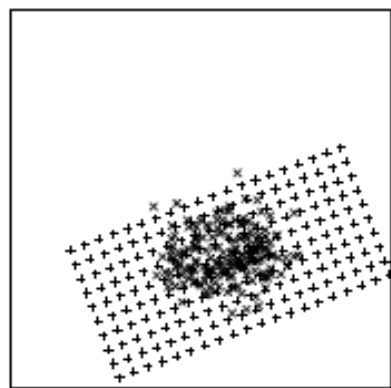
# Vivaldi: Robustness

Robustness of a simulated Vivaldi network (grid) when adding 200 nodes

$\delta = 0.05$



$\delta = 0.25 \times \text{local error} / (\text{local error} + \text{remote error})$



$t = 1$

$t = 10$

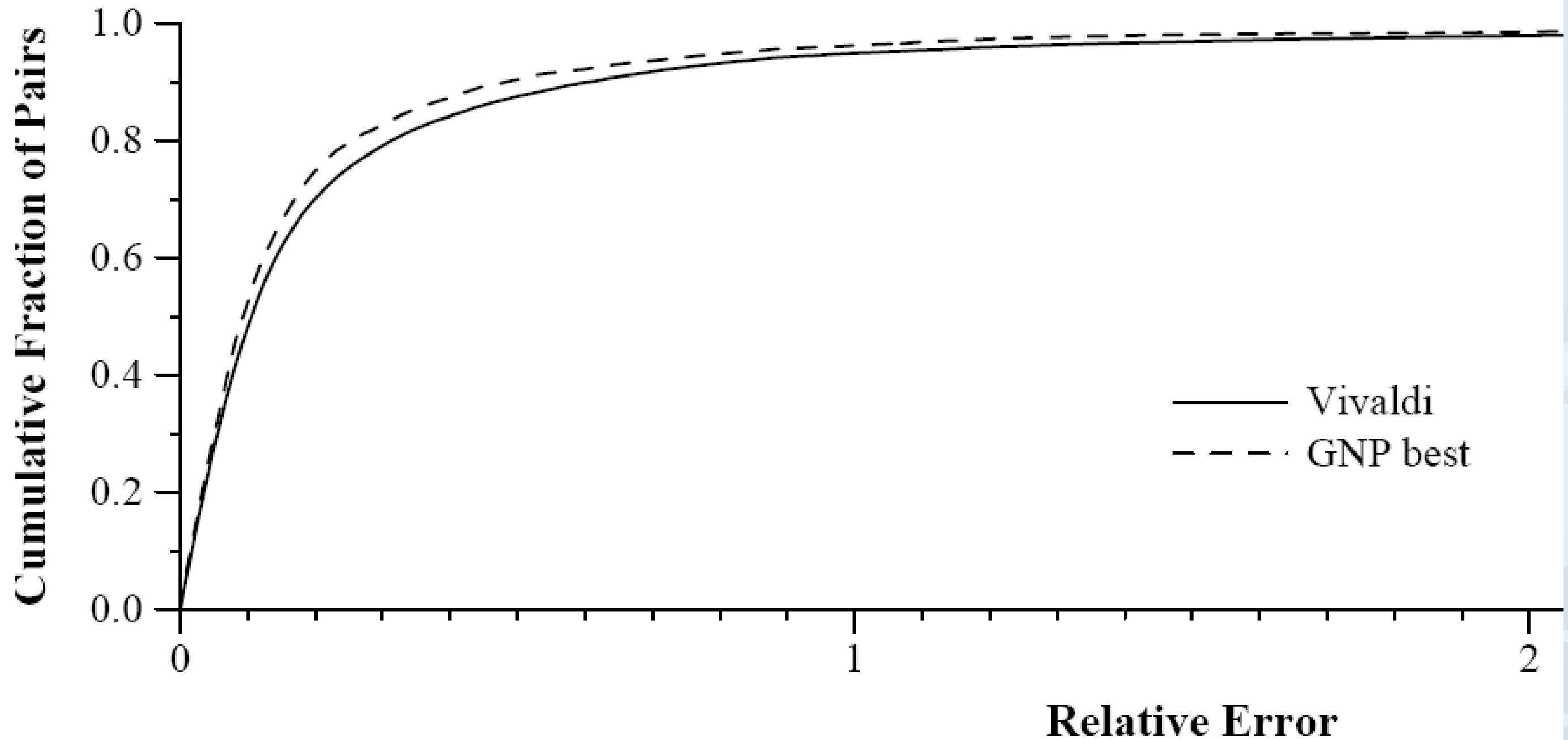
$t = 50$

$t = 100$

$t = 200$

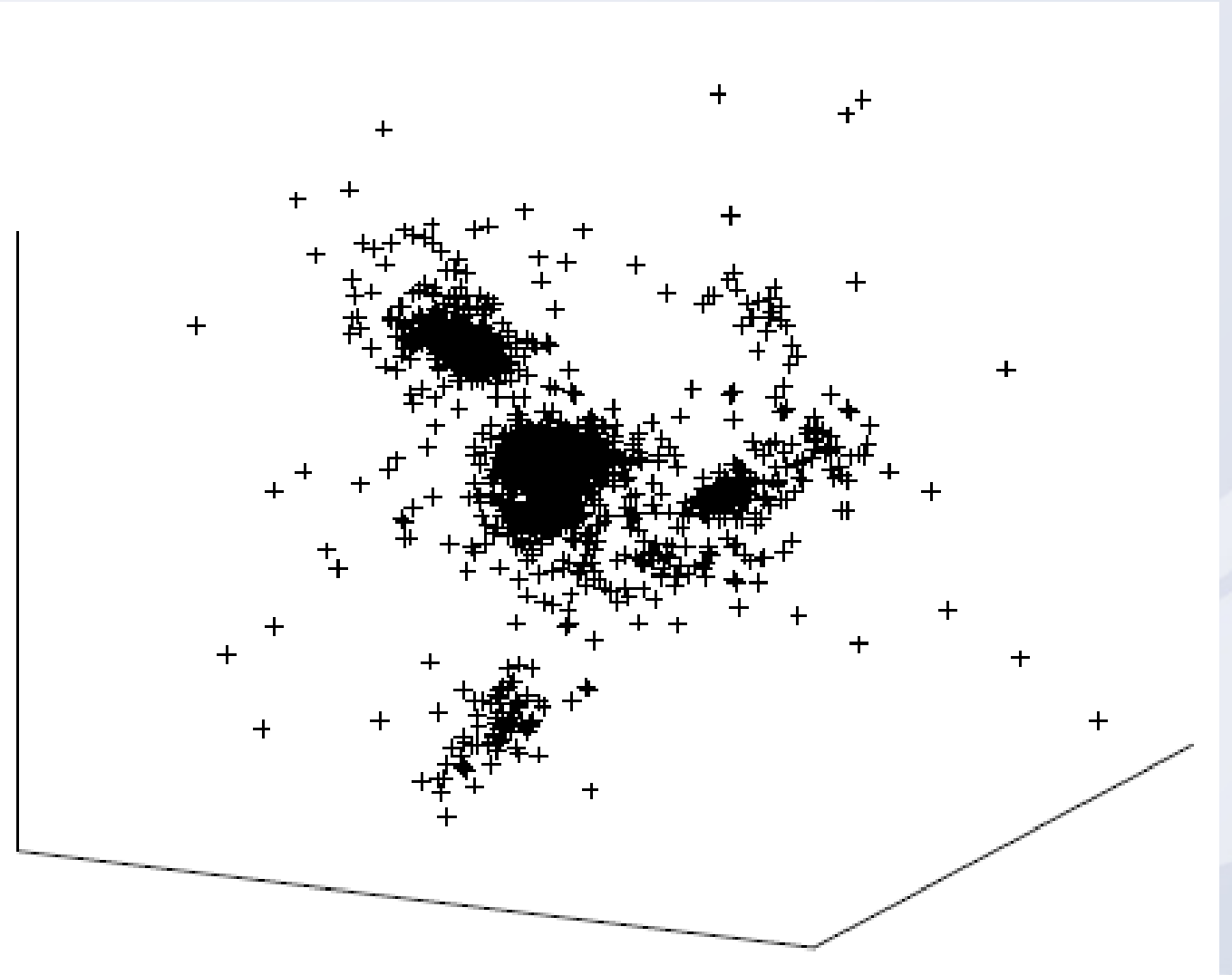
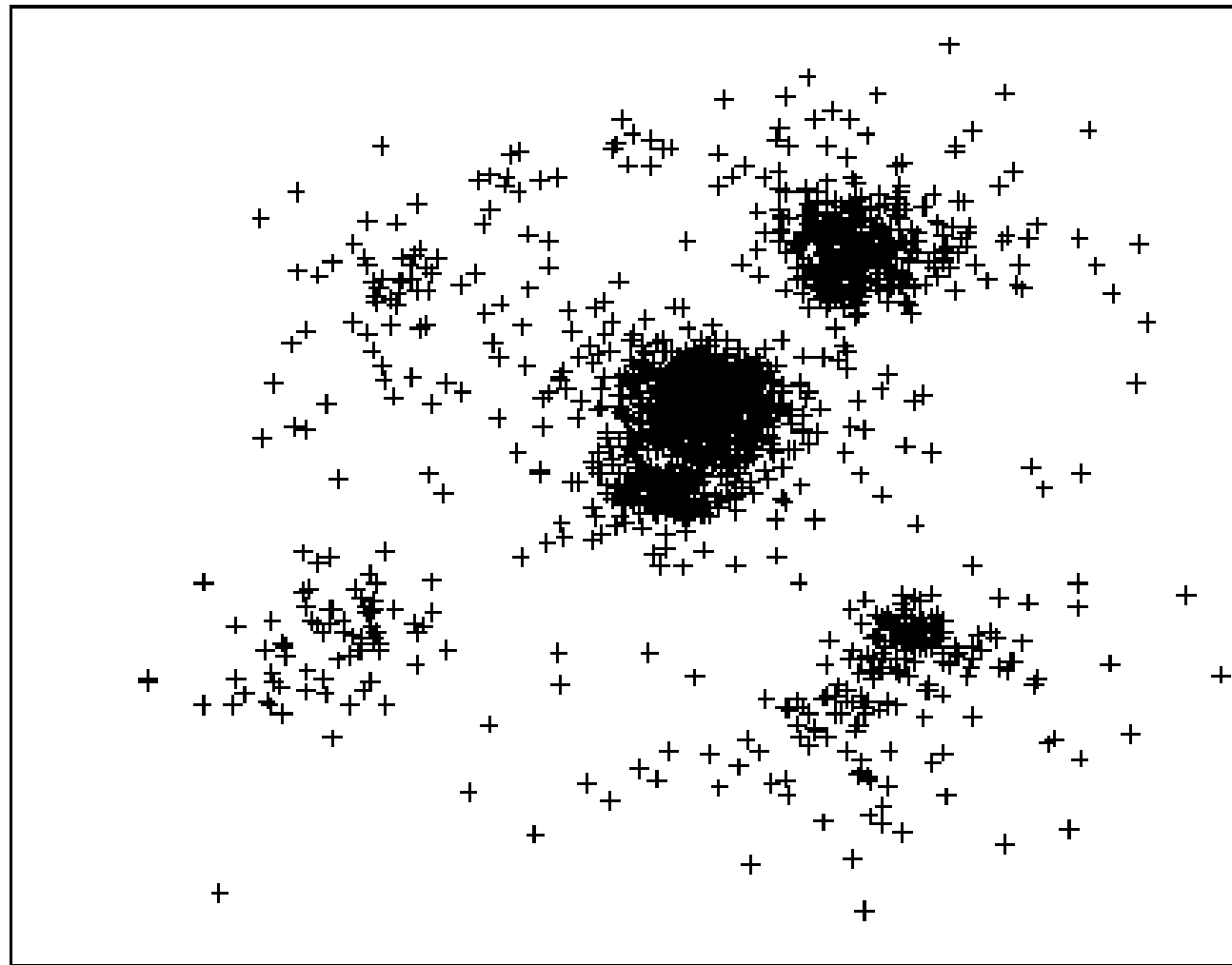
$t = 300$

# Vivaldi: Comparison to GNP



# Vivaldi: Results

Result map of DNS server dataset



Clusters for Europe, United States, **Erroneous nodes**

# Vivaldi: Height vectors

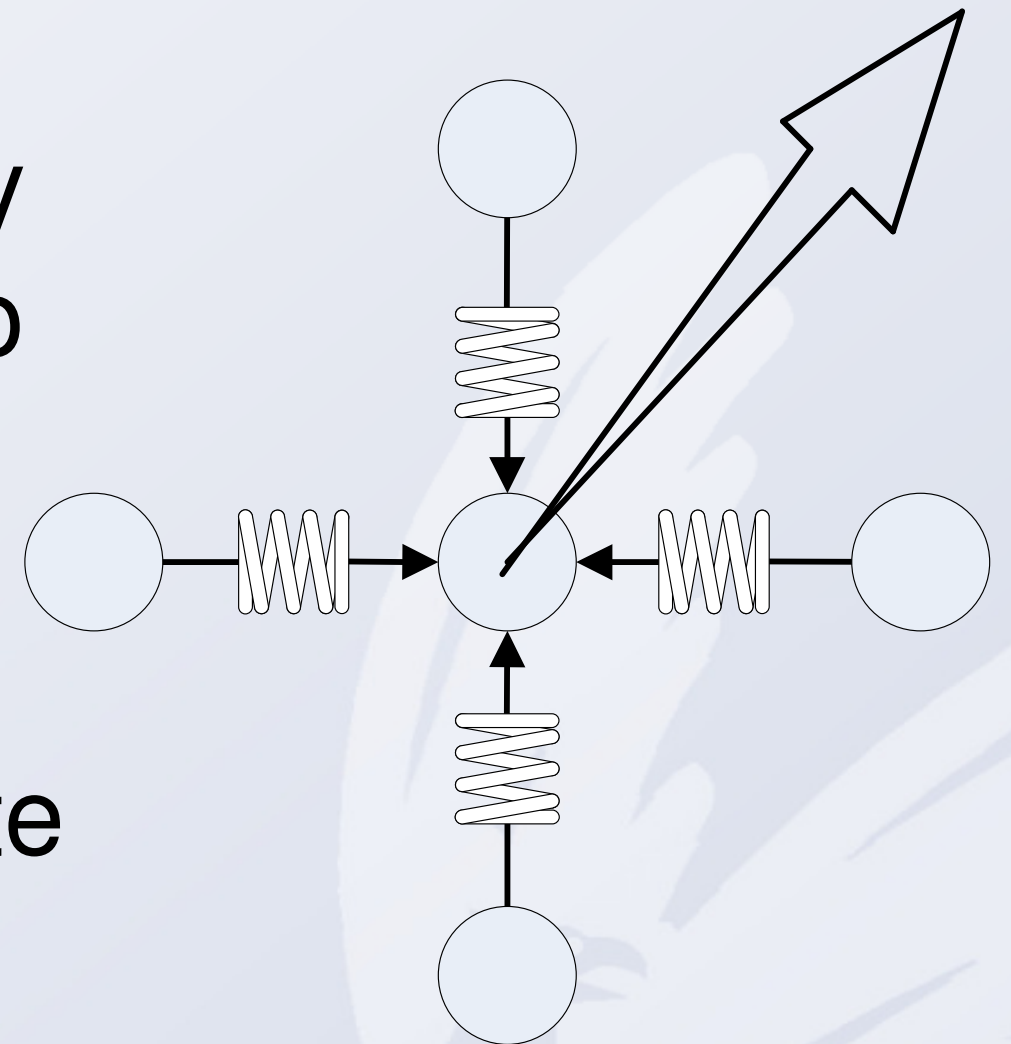
## Problems:

- Access link is independent of routing, but adds significant latency
- Distant erroneous nodes will be too close to all their neighbours

## Solution:

Add height to coordinates to indicate obligatory access time

Trapped nodes can always decrease their error by going up or down



# Vivaldi: Summary

## Vivaldi conclusions:

- Decentralized latency prediction can achieve GNP quality
- System of springs is adaptive to change
- Accuracy can be increased using directionless height

## Problems:

- Lying about your position disrupts coordinates of others
- Only applicable to large-scale peer-to-peer systems

# Conclusions

- Existing infrastructure suffices for advanced latency measurement
- Many different techniques exist for latency prediction
- Euclidean space model can be used to predict latency with reasonable accuracy
- Latency prediction can be decentralized

Questions / Discussion