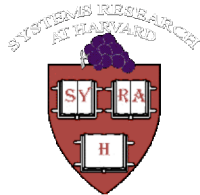


Network Coordinates in the Wild

Jonathan Ledlie

Margo Seltzer

Harvard University



Paul Gardner

Aelitis / Azureus



Hourglass Project

<http://www.eecs.harvard.edu/~syrah/hourglass>

Overview

Introduction

Azureus

- Very popular BitTorrent client
- 1-1.5 million current users worldwide!

Locality essential!

- Uses network coordinates (NCs)

Overview

Introduction

Azureus

- Very popular BitTorrent client
- 1-1.5 million current users worldwide!

Locality essential!

- Uses network coordinates (NCs)

Problem: inaccurate and unstable :-)

- Developers found our work on PlanetLab [Ledlie '06]
- Worked with them to analyze and tame their NC system

BitTorrent

Introduction

Protocol for file sharing

- Peers don't download from "seed" (content source)
- Instead: discover other peers and download from them

Upside

- Users can provide content without bandwidth hit

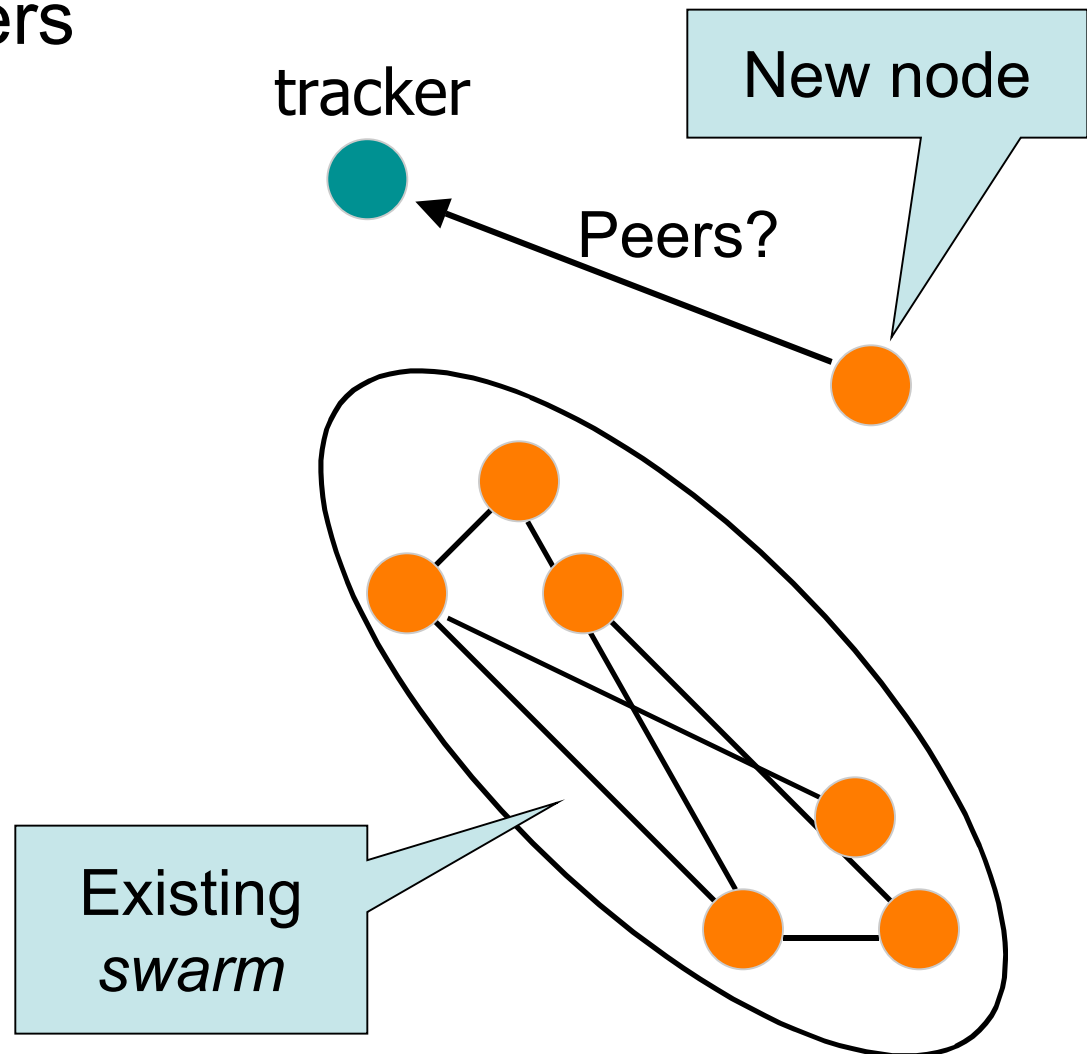
Downside

- Consumes huge portion of Internet bandwidth (18-55%)

BitTorrent: Peer Discovery

Introduction

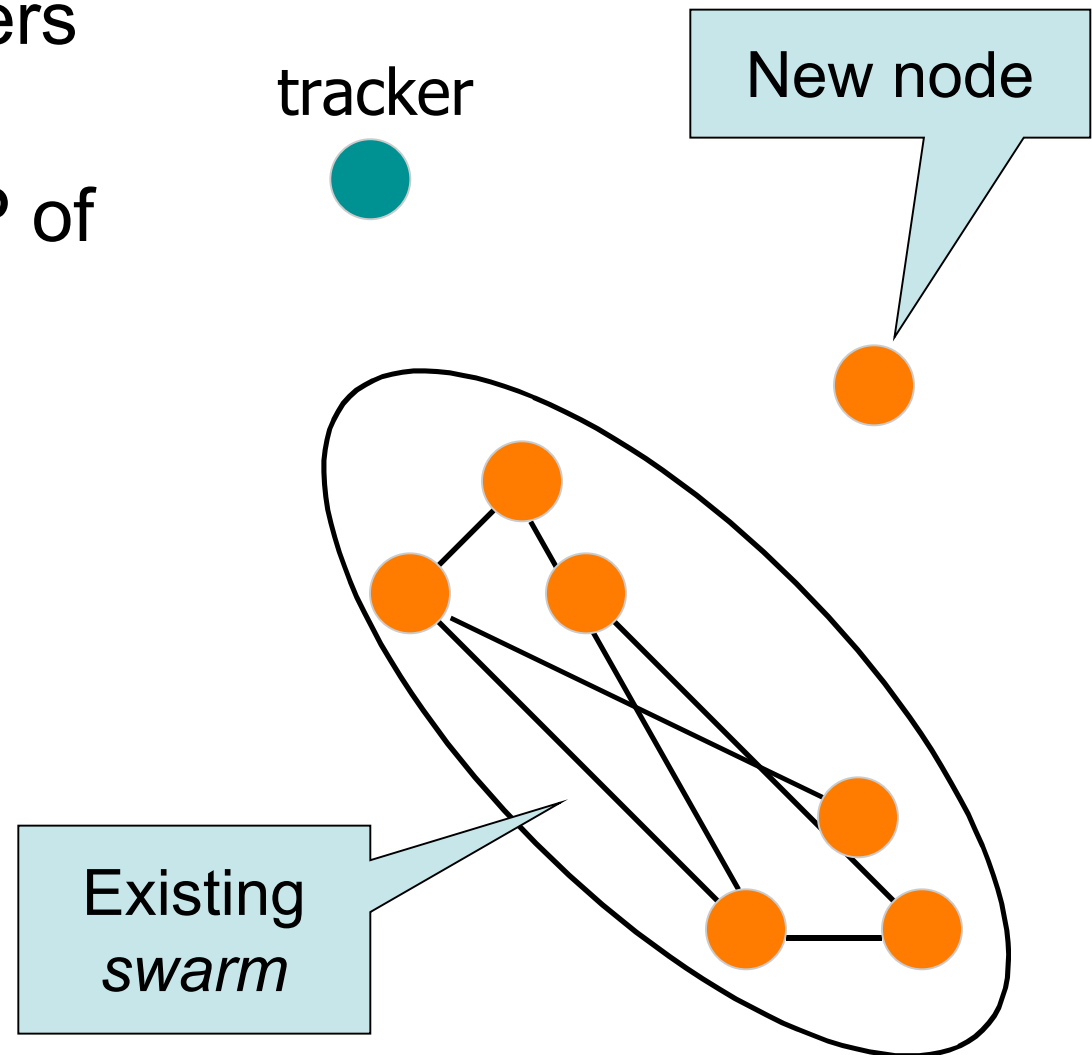
1. Request list of peers from tracker



BitTorrent: Peer Discovery

Introduction

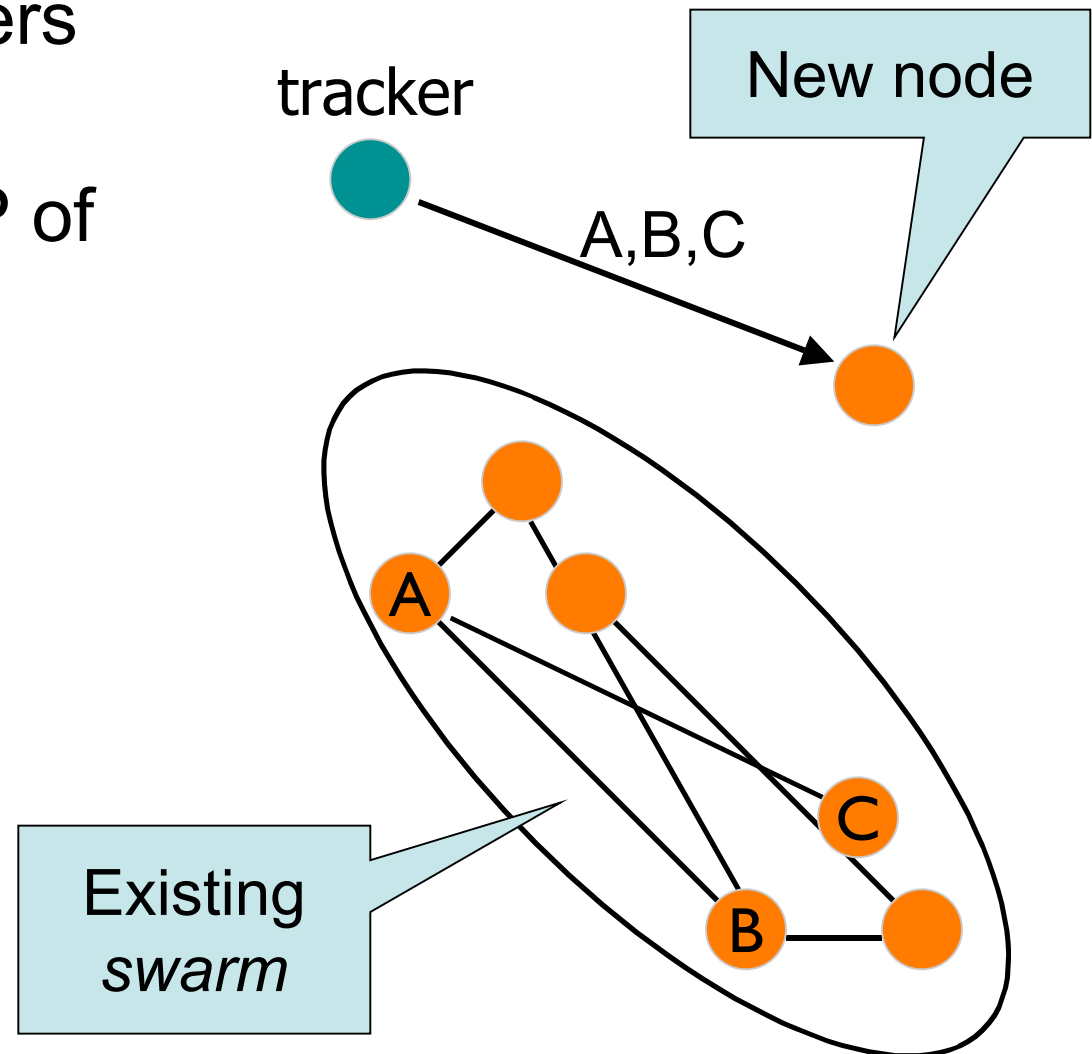
1. Request list of peers from tracker
2. Tracker records IP of requester



BitTorrent: Peer Discovery

Introduction

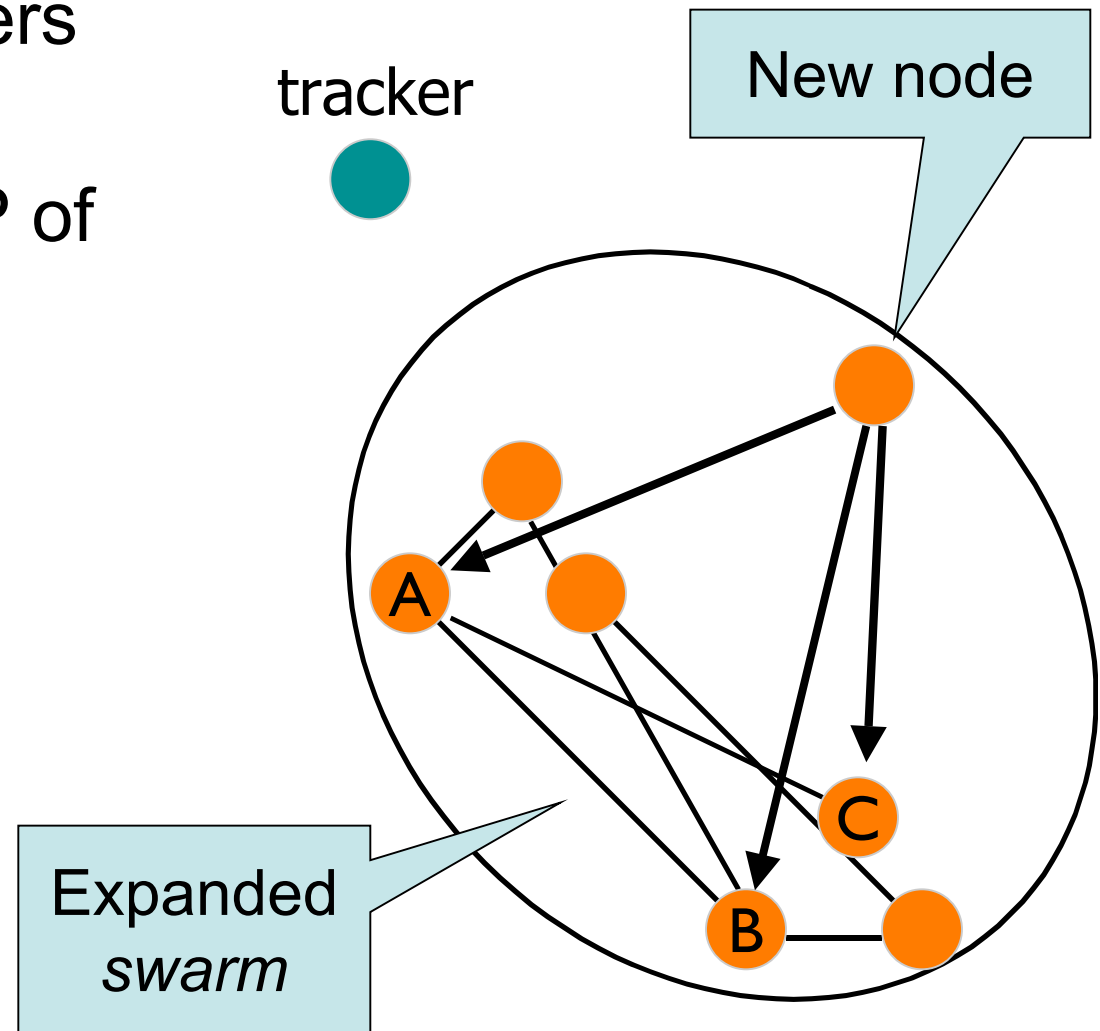
1. Request list of peers from tracker
2. Tracker records IP of requester
3. Tracker responds with list of recent peers



BitTorrent: Peer Discovery

Introduction

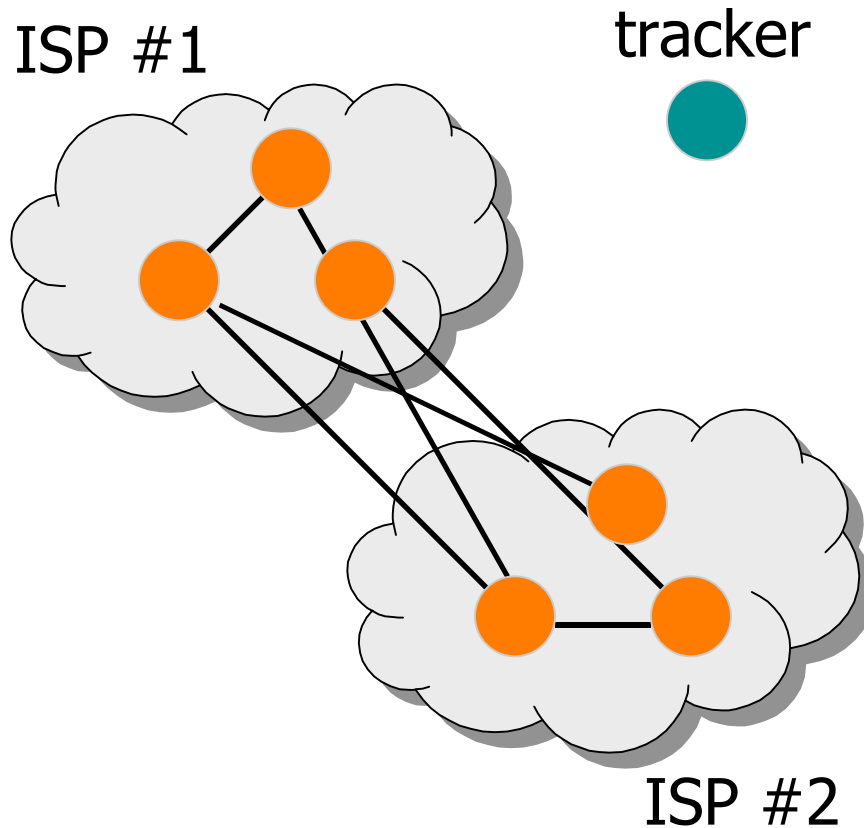
1. Request list of peers from tracker
2. Tracker records IP of requester
3. Tracker responds with list of recent peers
4. Contact



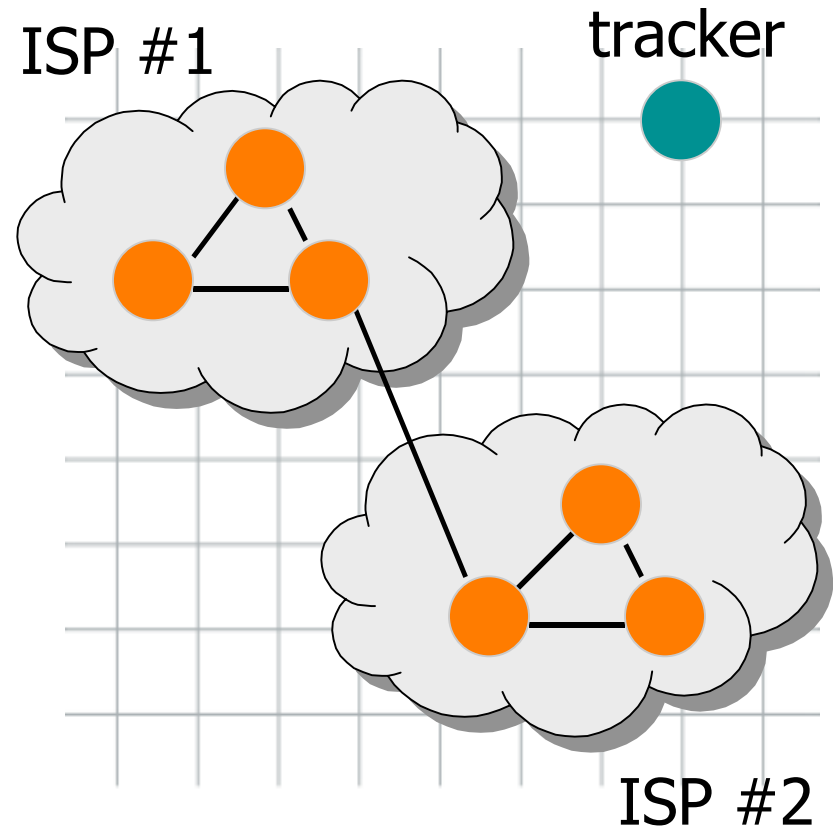
BitTorrent: Locality Matters

Introduction

Unbiased swarms



Locally-biased swarms



- Improved b/w for peers
- Reduced inter-ISP b/w

Contributions

Introduction

Evidence

- Use NCs for locality-aware “anycast” decisions

Developed new techniques

- Minimize overhead
- Churn

Data from Internet-scale system

- Large latency matrix from end-users

Outline

Introduction

A Network Coordinate Refresher

Methodology

Two Problems and our Solutions

- Limited Horizon
- Slow DHT Lookups

Conclusion

Network Coordinates

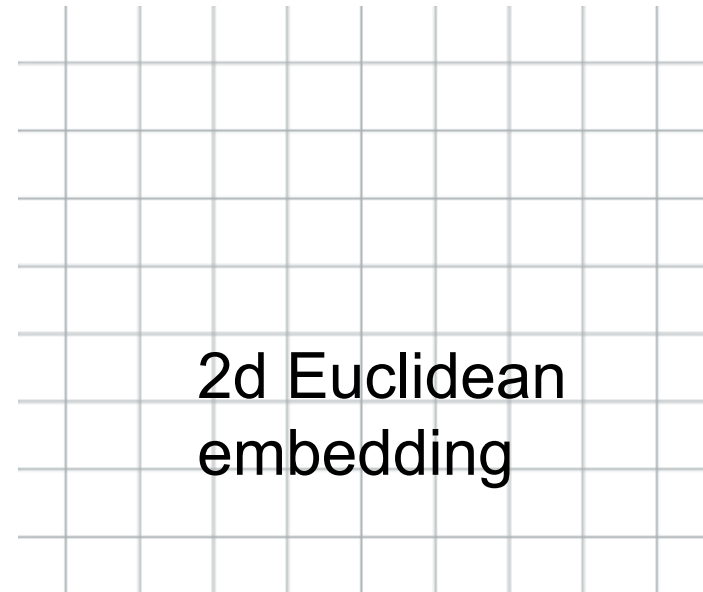
NC Refresher

Embed inter-node latencies into metric space

- Measure to (small) subset of network
- Establish coordinate
- **Predict** missing measurements

Low dimensional space

- 2-5 dimensions in practice

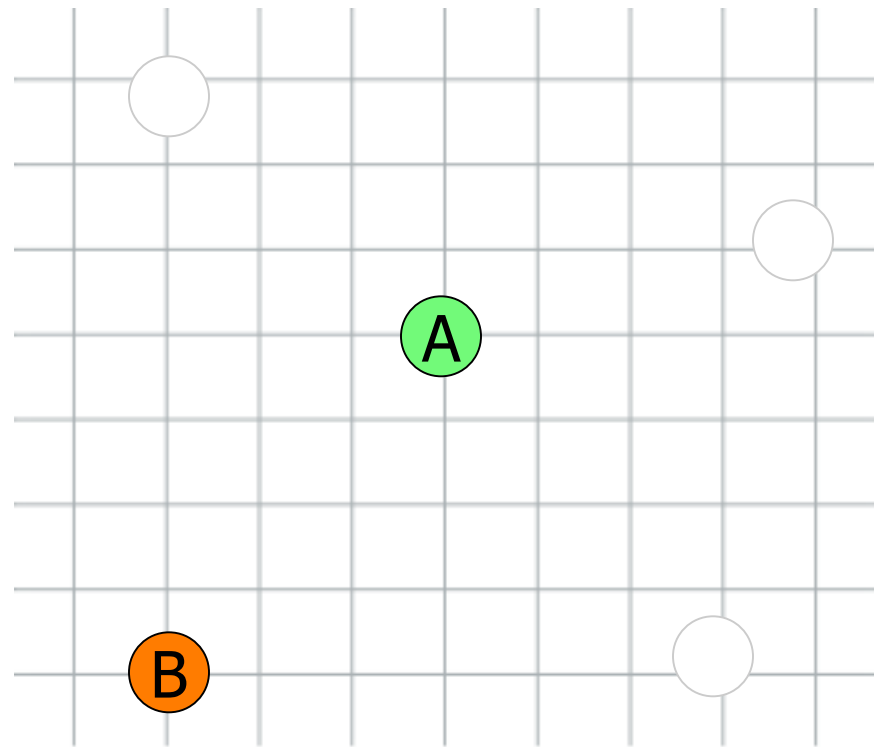


Vivaldi Refinement Process

NC Refresher

Incremental refinement: minimize global prediction error

* Vivaldi [Cox '03,Dabek '04]

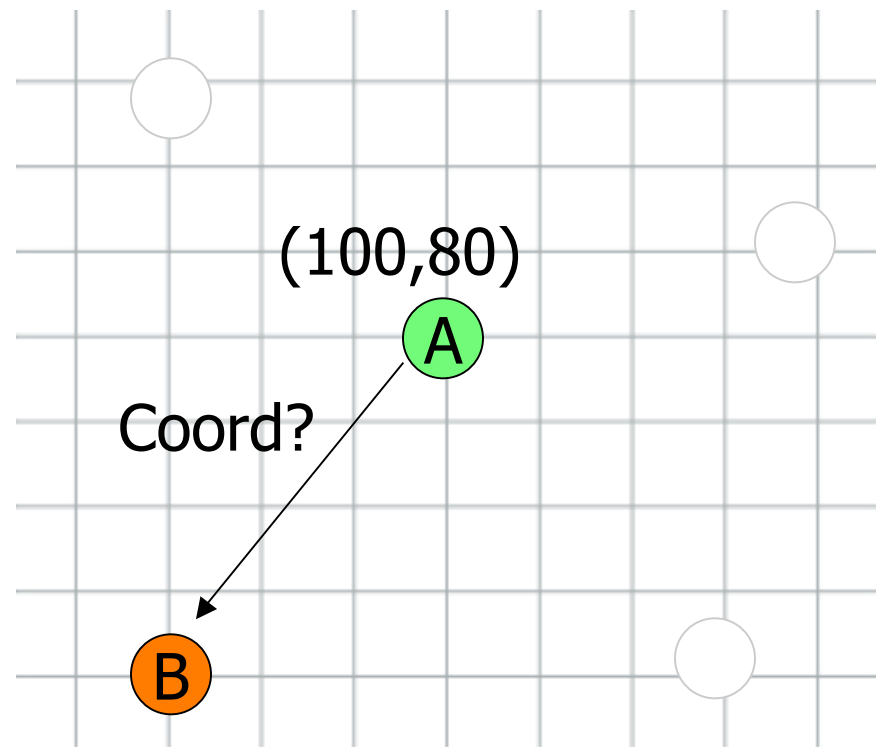


Measurement

NC Refresher

Incremental refinement: minimize global prediction error

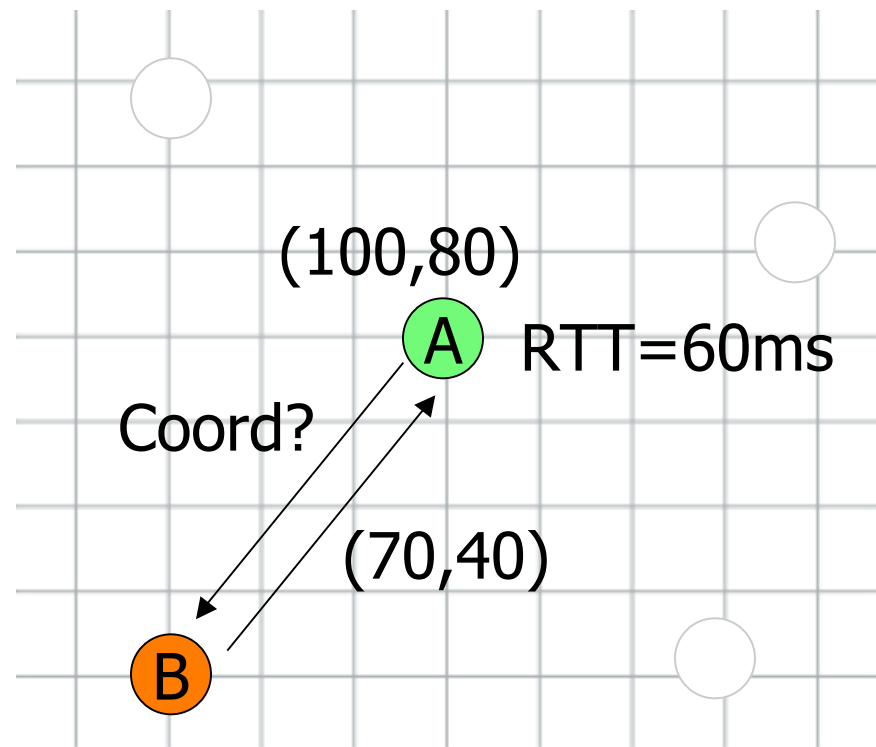
1. **A** measures latency to **B**.



Reply

NC Refresher

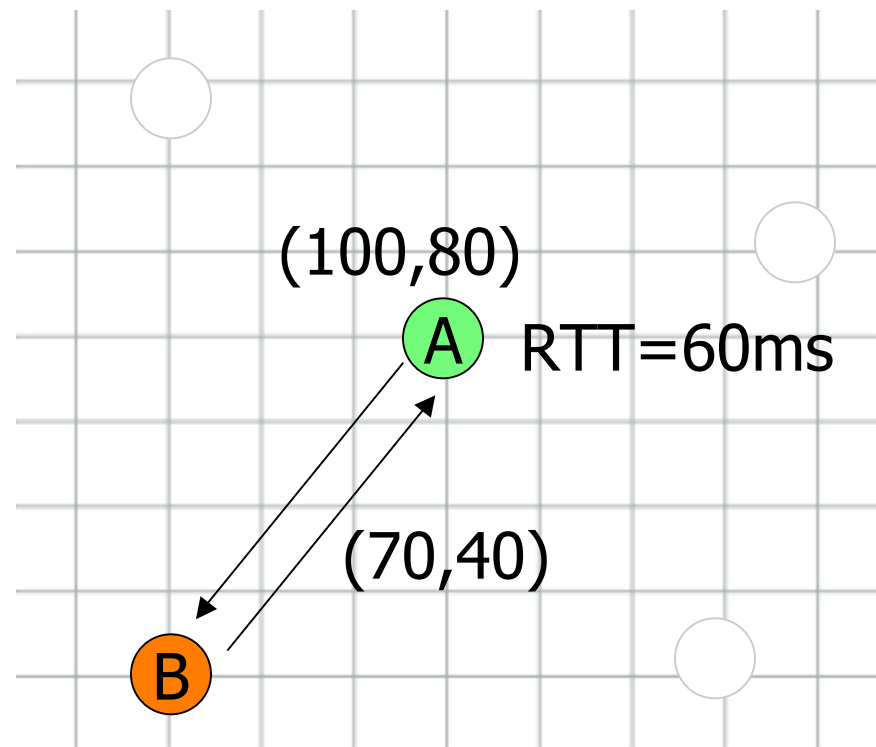
1. **A** measures latency to **B**.
2. **B** replies with its coord.
A deduces RTT.



Computation

NC Refresher

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



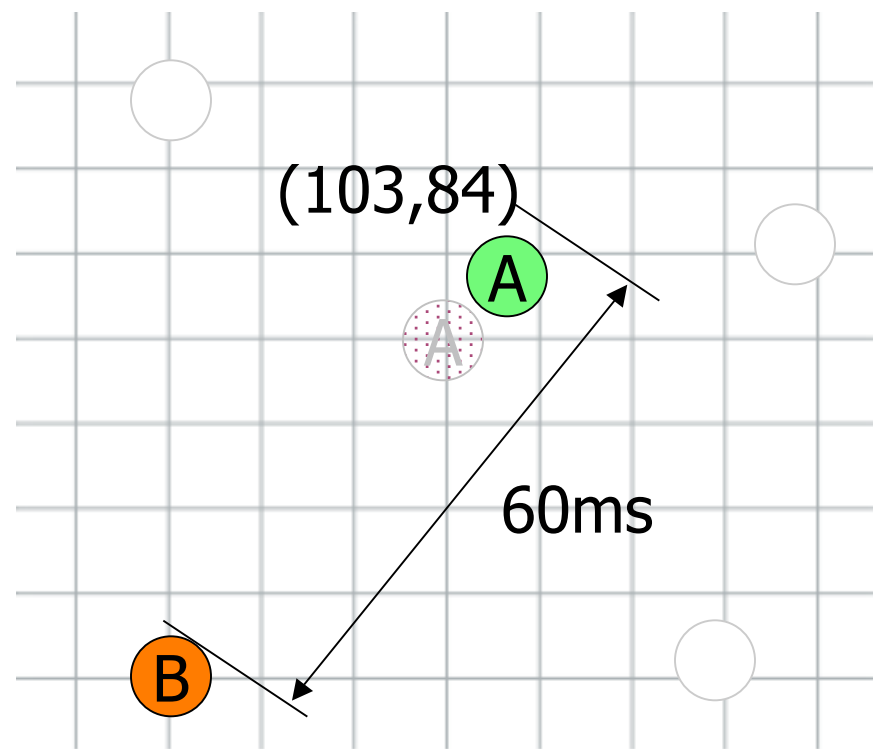
$$\text{Estimate} = |(100,80)-(70,40)| = 50\text{ms}$$

$$\text{Error} = (60 - \text{Estimate}) = 10\text{ms}$$

Adjustment

NC Refresher

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**.



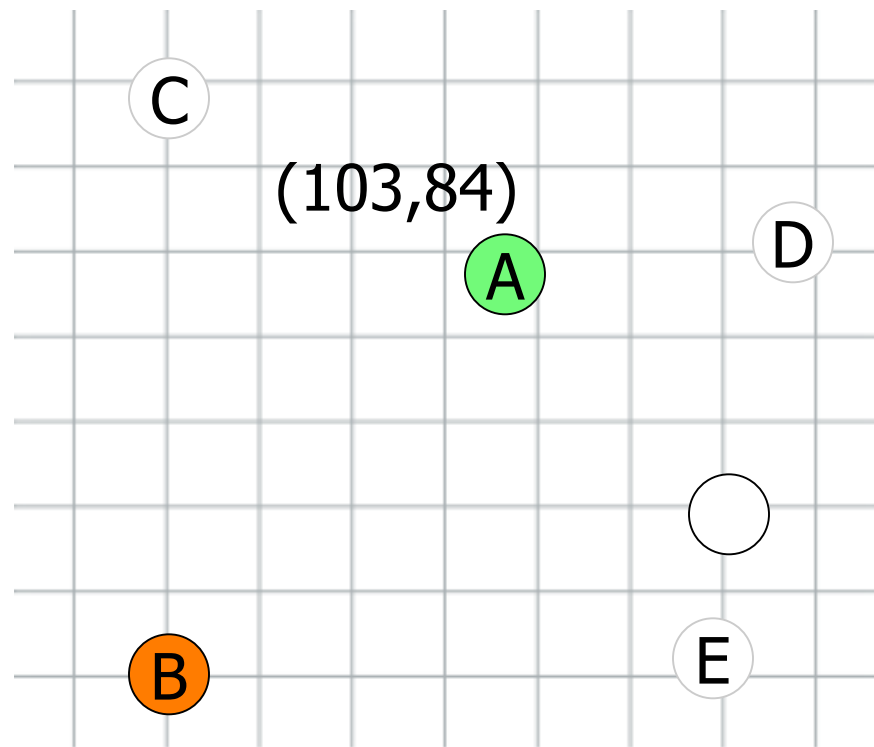
$$\text{Estimate} = |(100,80)-(70-40)|=50\text{ms}$$

$$\text{Error} = (60 - \text{Estimate}) = 10\text{ms}$$

Repeat

NC Refresher

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.

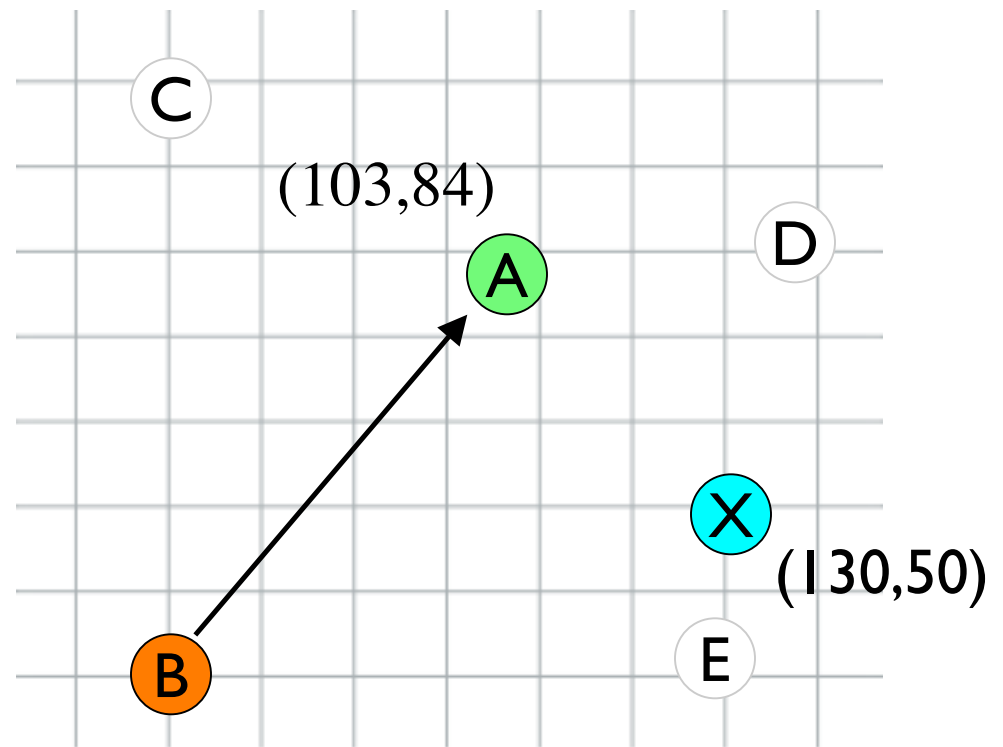


Predict

NC Refresher

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

1. **A** starts measurement 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**

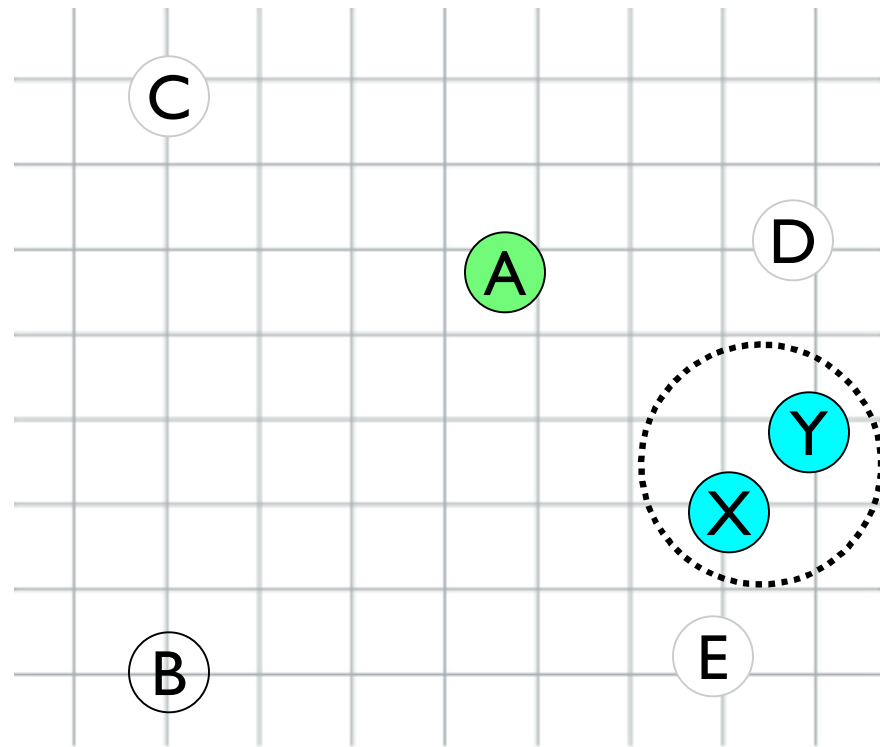


Predict

NC Refresher

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

1. **A** starts measurement 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**



Outline

Introduction

A Network Coordinate Refresher

Methodology

Two Problems and our Solutions

- Limited Horizon
- Slow DHT Lookups

Conclusion

Goals: Study and Refine NCs

Methodology

Twin goals: Study and refine Internet-scale network coordinate system

Study

- Observe and understand causes of inaccurate prediction

Refine

- Test new techniques in simulation / local clients / PL
- Release

Repeat

Two Data Collection Processes

Methodology

Fine: Instrumented Azureus clients run on PlanetLab

- Logged every update
 - » real RTT, remote coordinate, remote error, per update error
- Detailed picture + PL-to-non-PL latency matrix

Coarse: Added NC statistics message to Azureus

- Summarizes their coordinates' behavior
 - » instantaneous error, neighbor error, stability
 - » other metrics [Lua '05]

Measurement Challenges

Methodology

~1.3 million Azureus clients running at any given moment

Impossible to acquire all-to-all latency matrix

- Complicates simulation and testing
- Simulated with 249x2902 rectangular matrix

Impossible to force clients to run latest version

- Cannot ensure all clients are running algorithm X, parameter Y
- 10,000 users regularly (and automatically) update to CVS
 - » >50% within one day; >75% within one month.

Outline

Introduction

A Network Coordinate Refresher

Methodology

Two Problems and our Solutions

- Limited Horizon
- Slow DHT Lookups

Conclusion

Zero Maintenance

Limited Horizon

Azureus maintains DHT routing table (RT)

- Heartbeat messages
 - » Frequent: several per minute
- Tests new nodes for entry into RT
 - » Less frequent: one per 3-5 minutes

NC maintenance “piggybacks” on these messages

- Good: Zero additional messages!

Problem: Limited Horizon

Limited Horizon

Bad: Limits view of network to routing table

- RT (+ neighbor set) biased to nearby nodes
- Local bias damages accuracy [Dabek '04, Ledlie '05]
- Creates islands: poor estimation beyond horizon

Strawman

Limited Horizon

Make tug proportional to distance

- Boost impact of (occasional) long range contacts

Problems

- What's the right weight?
- Produces instability

Insight: Expand Horizon over time

Limited Horizon

Optimize coordinate against (recent) whole network

- Don't boost all at once; extend period of effect

Scale push/pull per neighbor by age

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

In Effect

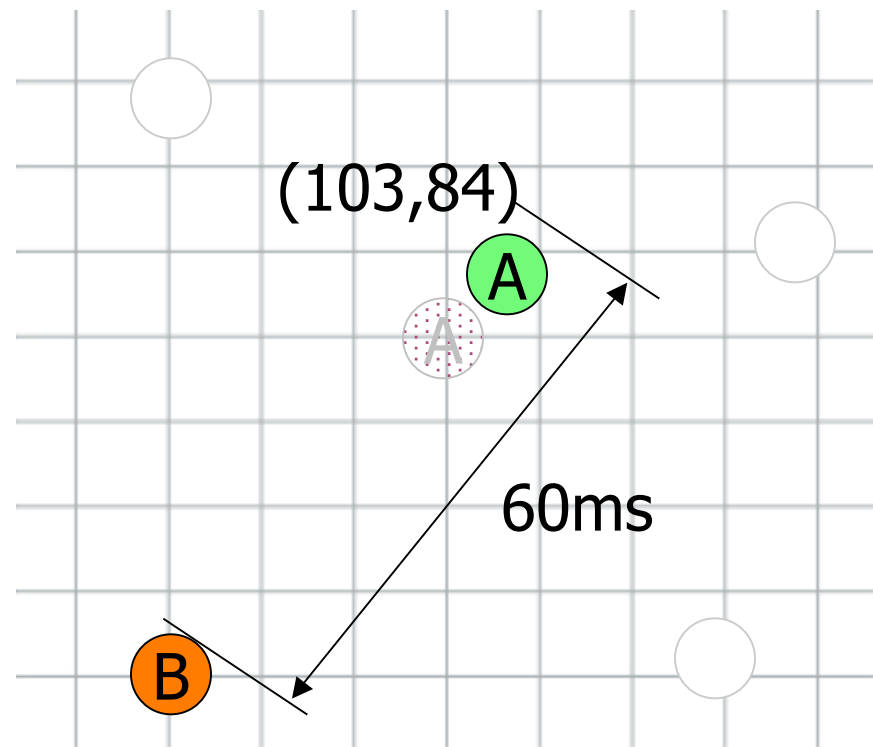
- Limits impact of high frequency (nearby) neighbors
- Extends impact of low frequency (longer-distance) ones

Original Adjustment Step

Limited Horizon

A optimizes with respect to B

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.



Neighbor Decay Adjustment Step

Limited Horizon

As springs age, they get looser

Force $F = 0^d$

for (n : neighbors) {

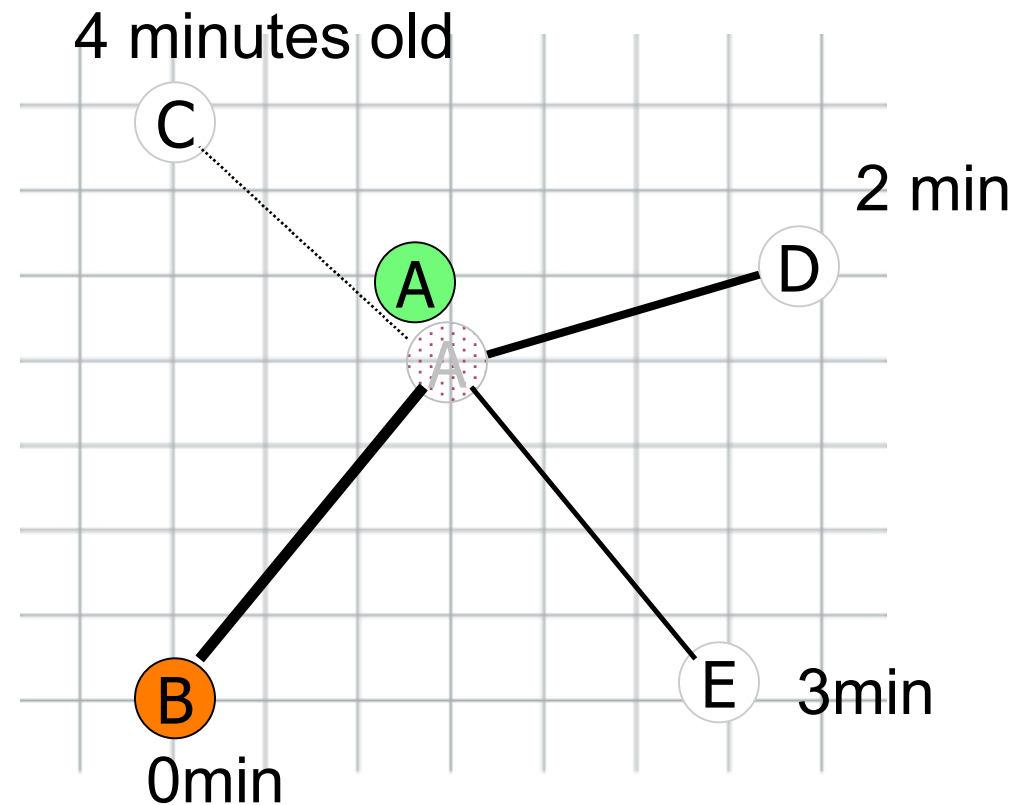
 Compute force F_n

 Norm F_n by age_n

$F += F_n$

}

$A += F$



Older information gets less weight

Reading the Neighbor Decay Video

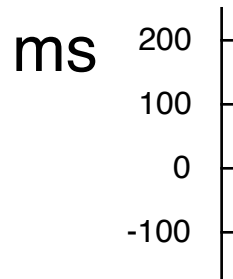
Limited Horizon

Neighbor Decay Comparison

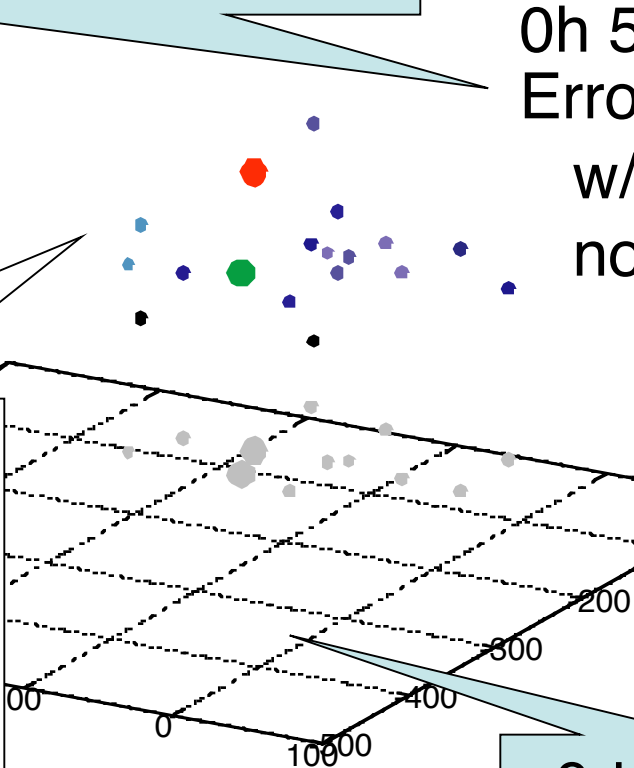
Pct. improvement in accuracy (30-50%)

Elapsed Time

0h 51m
Error: 0.472%
w/ND 0.303
no ND 0.574



Two coords - same node
With ND: Green
No ND: Red
Neighbors: Blue



Accuracy
ms

3d Euclidean
ms x ms x ms

Neighbor Decay Video

Limited Horizon

[First half hour in the life of two coordinates on Azureus
with and without neighbor decay]

Outline

Introduction

A Network Coordinate Refresher

Methodology

Two Problems and our Solutions

- Limited Horizon
- Slow DHT Lookups

Conclusion

Improved Accuracy, but...

End-to-end benefit

That's nice that accuracy
improves, but...

Do NCs actually improve app-level performance?

Aid in decisions involving node locality?

DHT Traversal: Problem

End-to-end benefit

Problem: Azureus frequently performs DHT lookups.

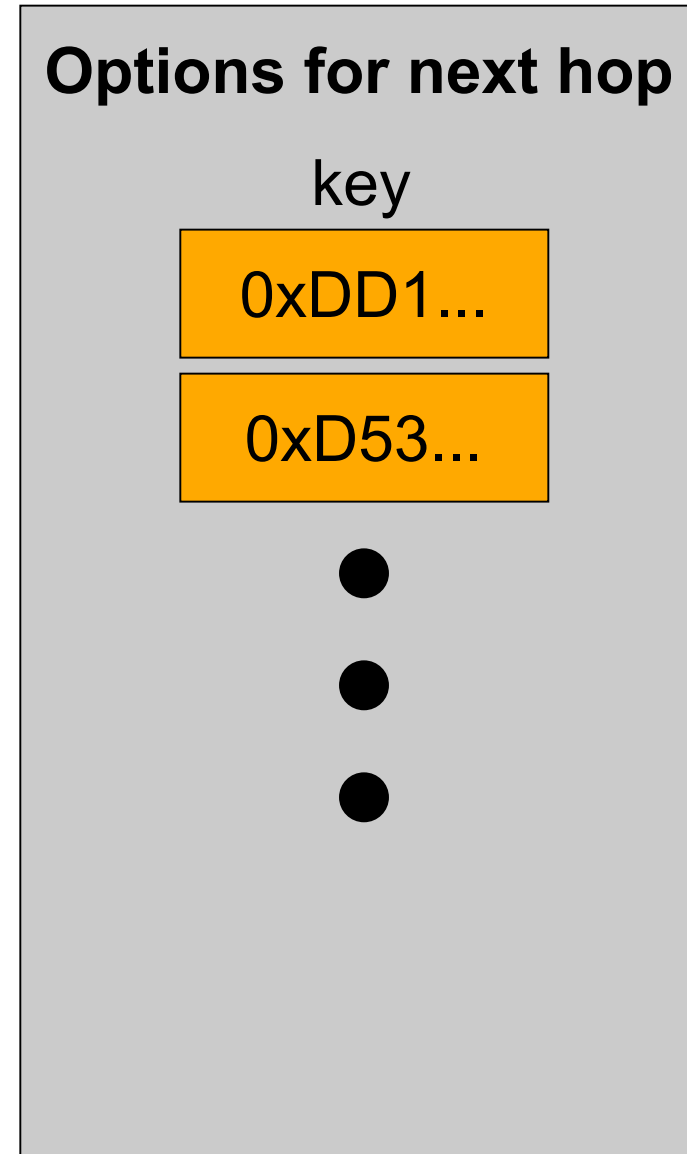
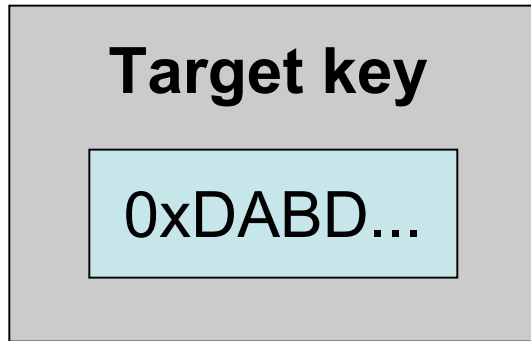
Need to be as fast as possible! E.g.:

- » Distributed tracker run within DHT
- » Want to find these trackers faster

**Can we use latency prediction
to speed up its lookups?**

Kademlia's Iterative Lookups

End-to-end benefit

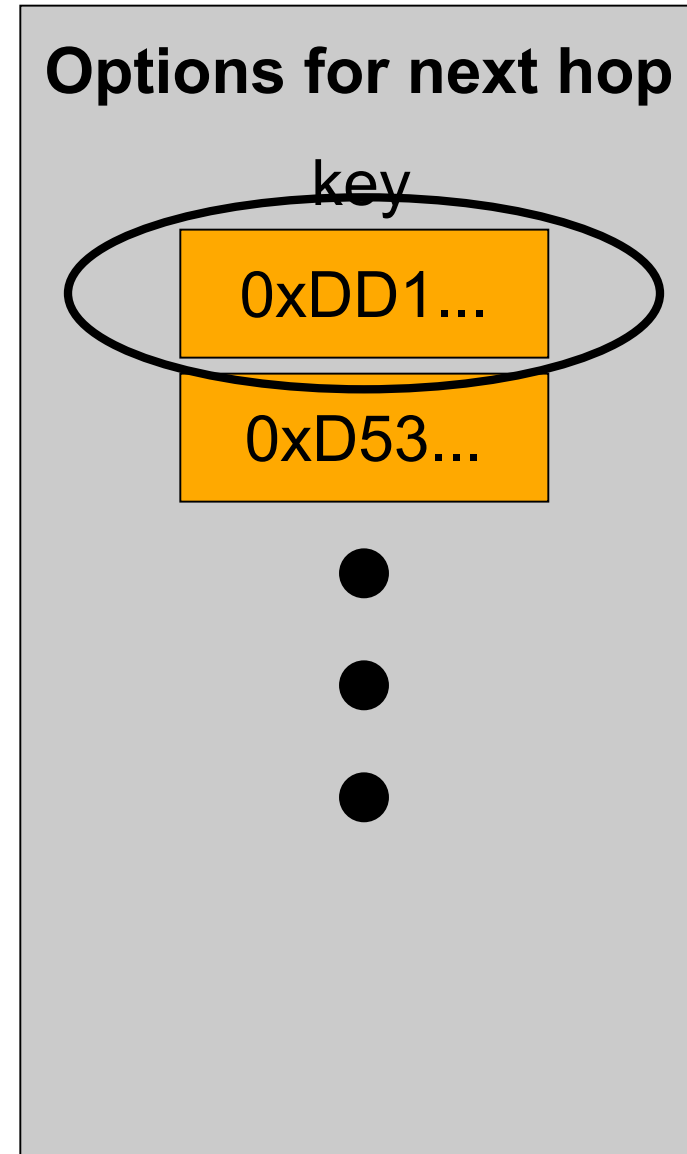
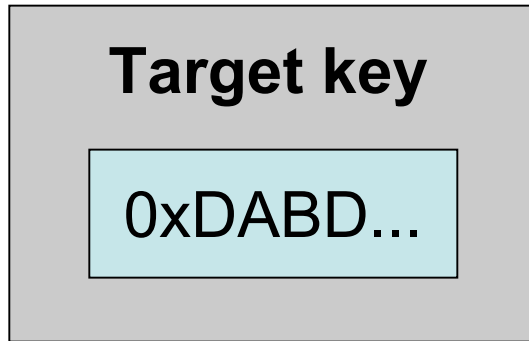


1. Sort routing table by distance to T
 - » distance $(a,b) = a \oplus b$

Kademlia [Maymounkov '02]

Kademlia's Iterative Lookups

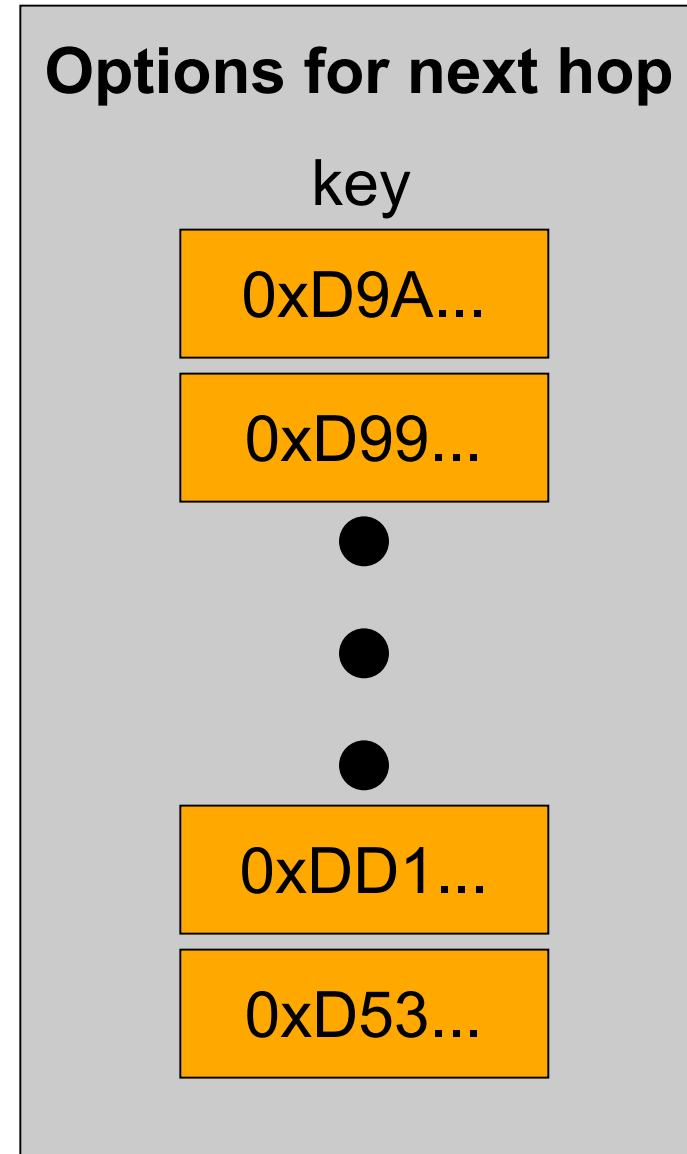
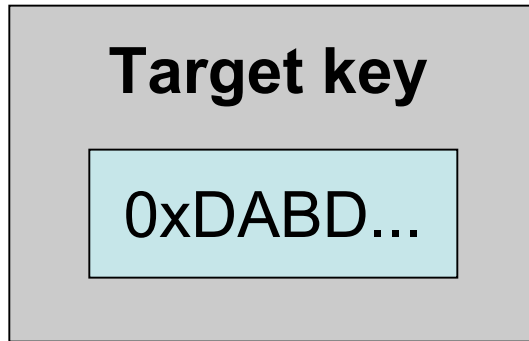
End-to-end benefit



1. Sort RT by distance to T
 - » distance $(a,b) = a \oplus b$
2. Query logically nearest for nodes near T

Kademlia's Iterative Lookups

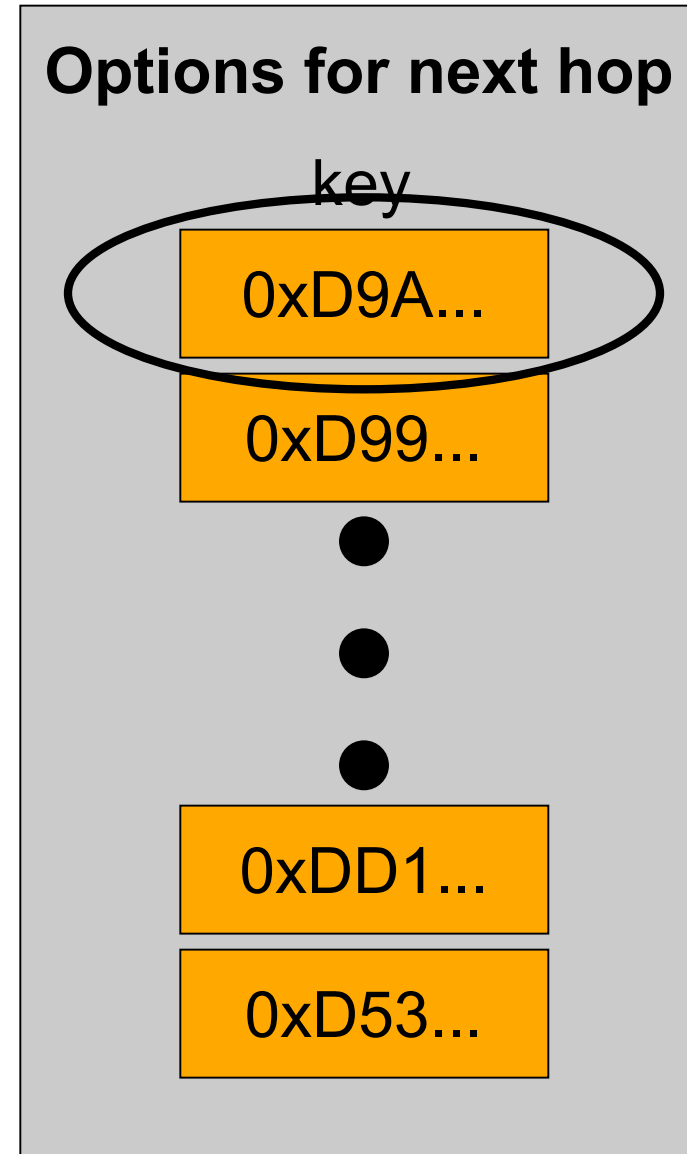
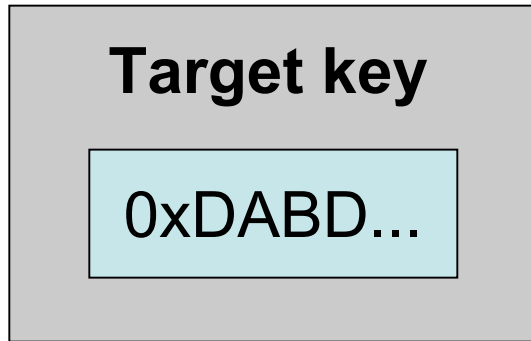
End-to-end benefit



1. Sort RT by distance to T
» distance (a,b) = $a \oplus b$
2. Query logically nearest for nodes near T
3. Add responses to list

Kademlia's Iterative Lookups

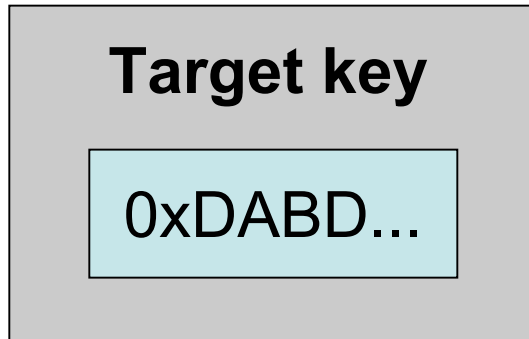
End-to-end benefit



1. Sort RT by distance to T
» distance (a,b) = $a \oplus b$
2. Query logically nearest for nodes near T
3. Add responses to list
4. Repeat

DHT Traversal: Solution

End-to-end benefit



Find good trade-off between
logical progress and delay

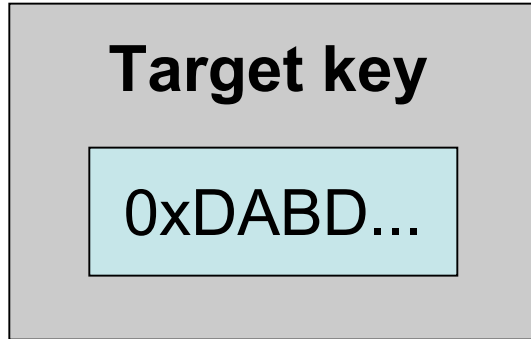
Similar opt. in Chord [Dabek '04].

Options for next hop

key	NC dist.
0xD9A...	1904
0xD99...	2216
0xD98...	90
0xD9F...	301
0xDD1...	595
0xD53...	120

DHT Traversal: Solution

End-to-end benefit



Find good trade-off between logical progress and delay

1. Eliminate logically distant hops
 - » Make similar progress to target

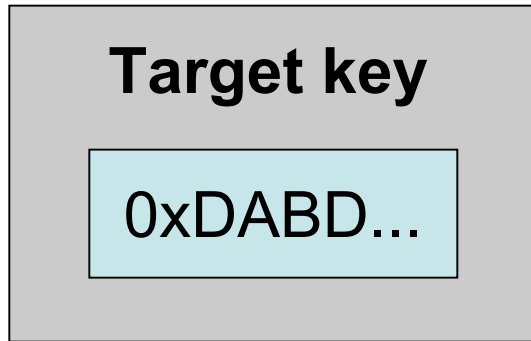
≈ Same number of expected hops if fixing same number of bits

Options for next hop

key	NC dist.
0xD9A...	1904
0xD99...	2216
0xD98...	90
0xD9F...	301
0xDD1...	595
0xD53...	120

DHT Traversal: Solution

End-to-end benefit



Find good trade-off between logical progress and delay

1. Eliminate logically distant hops
 - » Make similar progress to target
2. Break ties in favor of lowest predicted latency

Options for next hop

key	NC dist.
0xD9A...	1904
0xD99...	2216
0xD98...	90
0xD9F...	301
0xDD1...	595
0xD53...	120

DHT Traversal: Evaluation

End-to-end benefit

Compared four methods:

	Options for next hop		
	key	OptNC	OrigNC
XOR →	0xD9A...	1904	1756
	0xD99...	2216	4103
Optimized →	0xD98...	90	170
Original →	0xD9F...	301	152
	0xDD1...	595	702
and Random	0xD53...	120	183

DHT Traversal: Evaluation

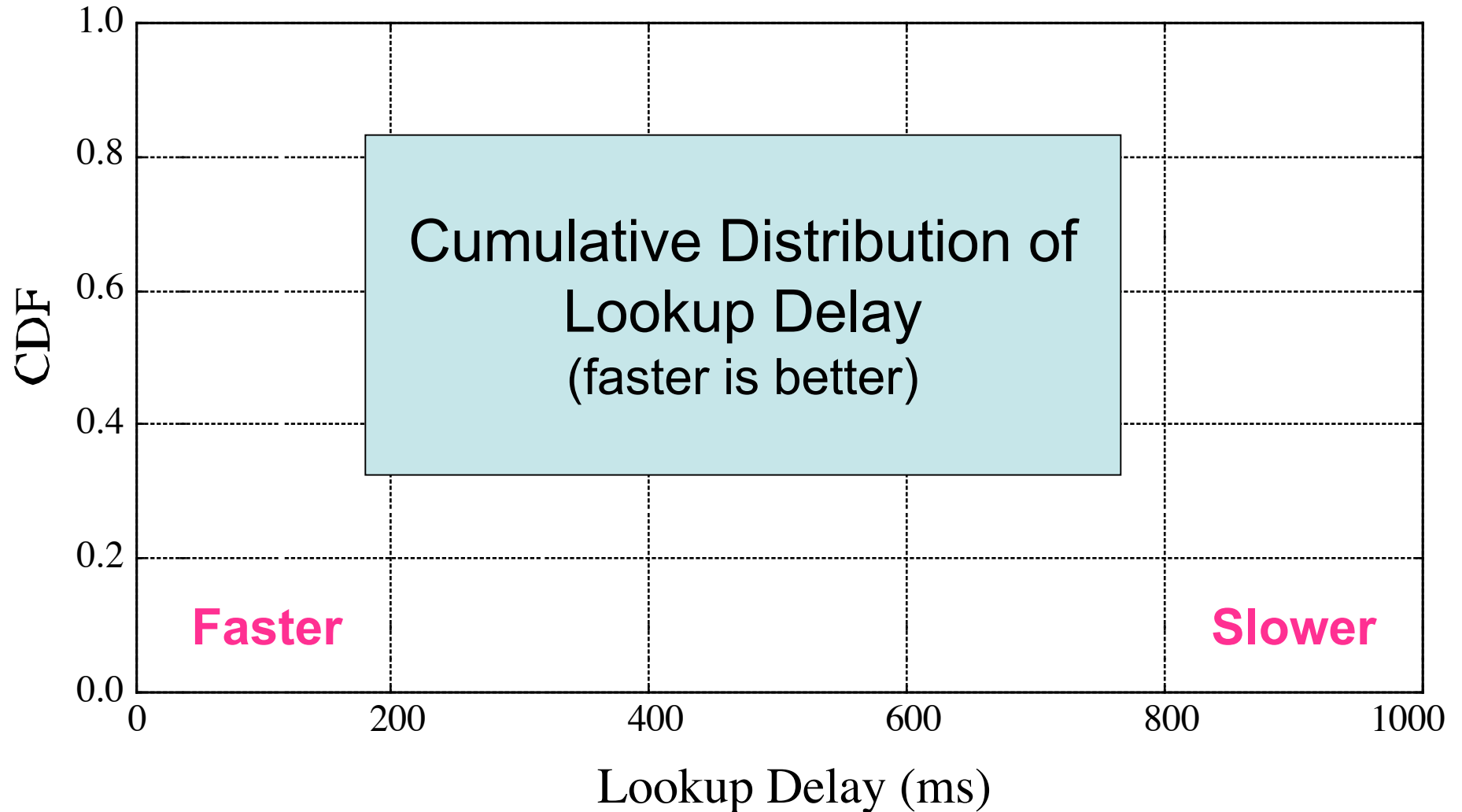
End-to-end benefit

Experiment

- Put 250 random keys into Azureus' DHT
- For each key, performed four lookups (once per method)
 - » XOR, OptNC, OrigNC, Random
 - » Permuted method ordering
- Repeated 10x

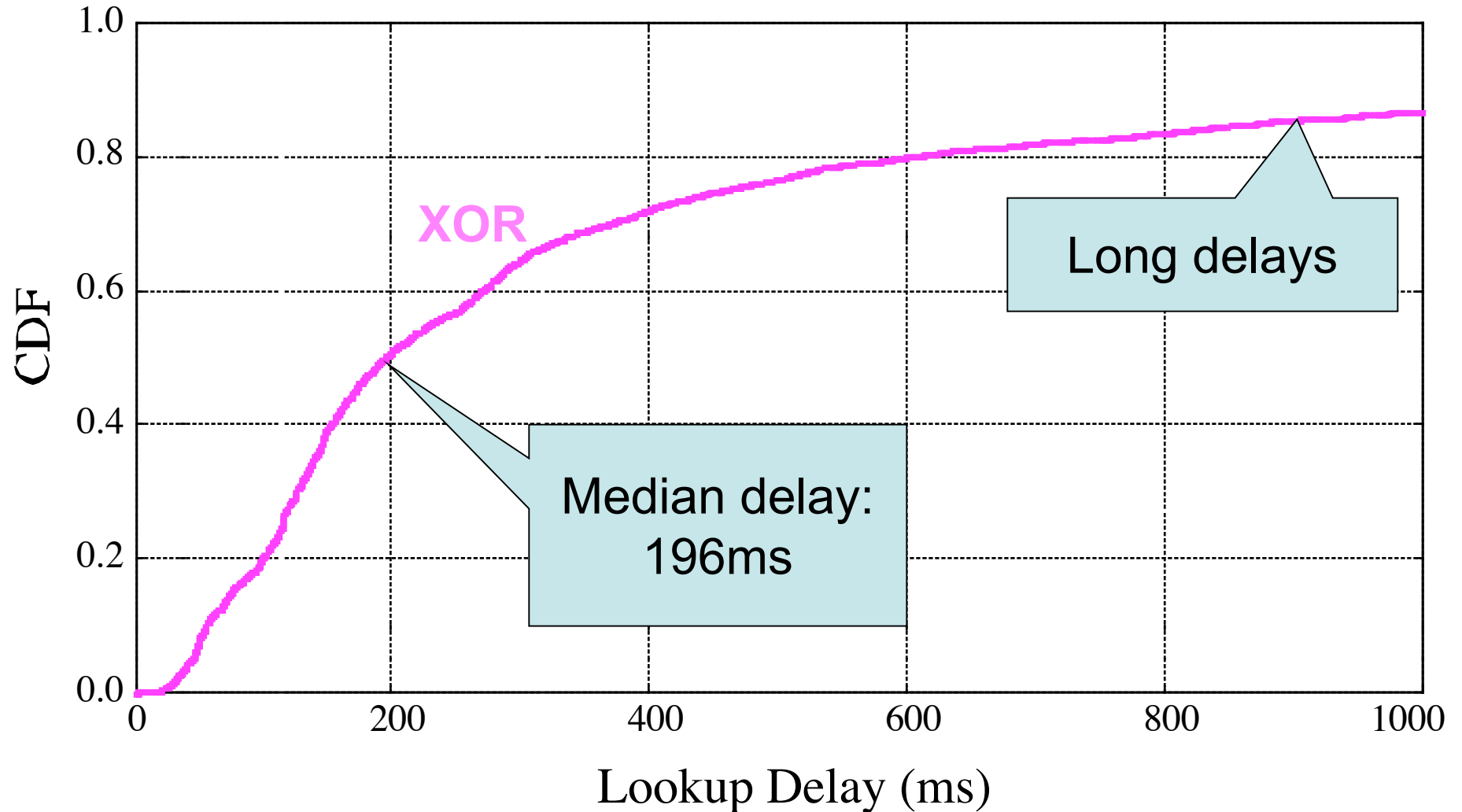
DHT Traversal: Evaluation

End-to-end benefit



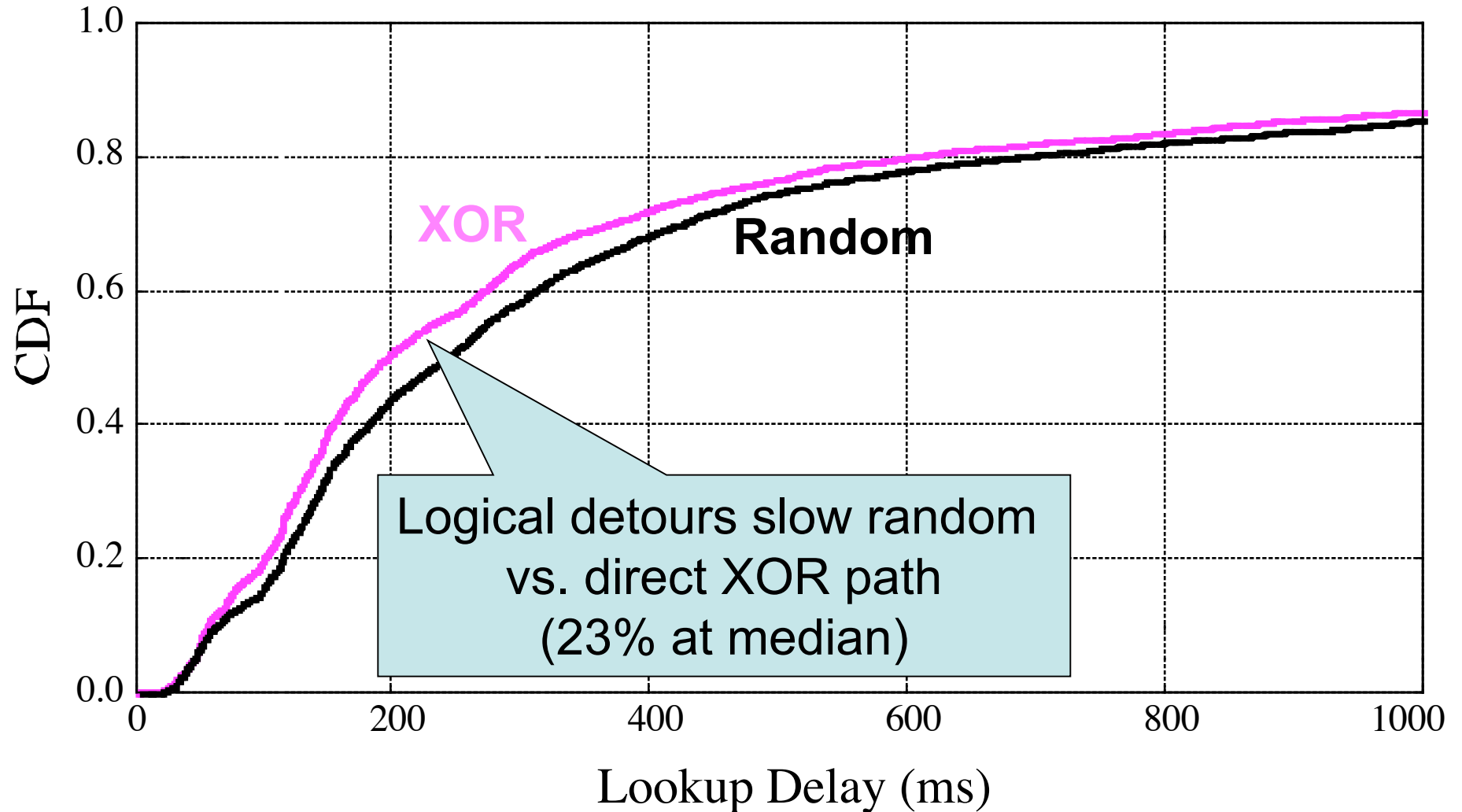
DHT Traversal: Evaluation

End-to-end benefit



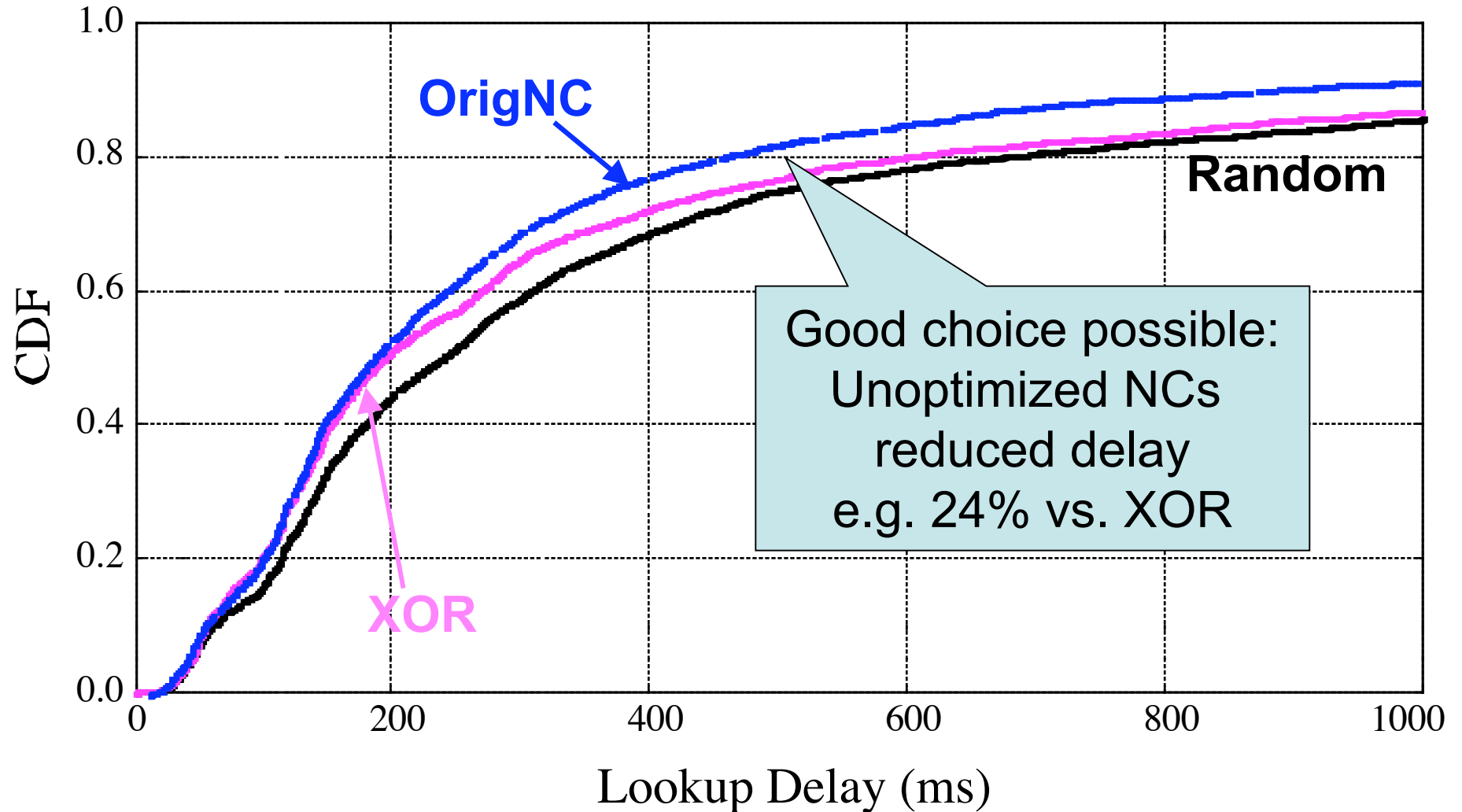
DHT Traversal: Evaluation

End-to-end benefit



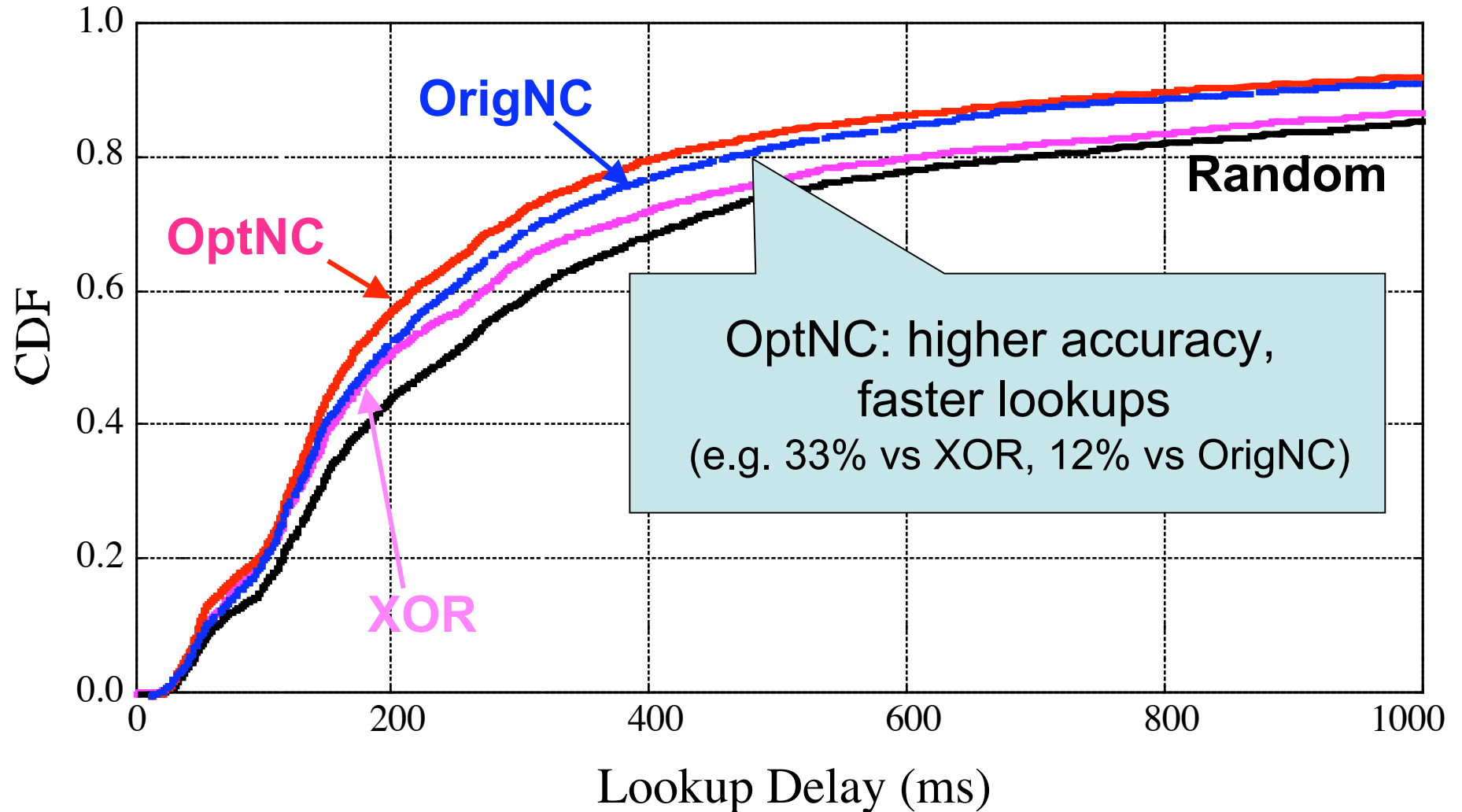
DHT Traversal: Evaluation

End-to-end benefit



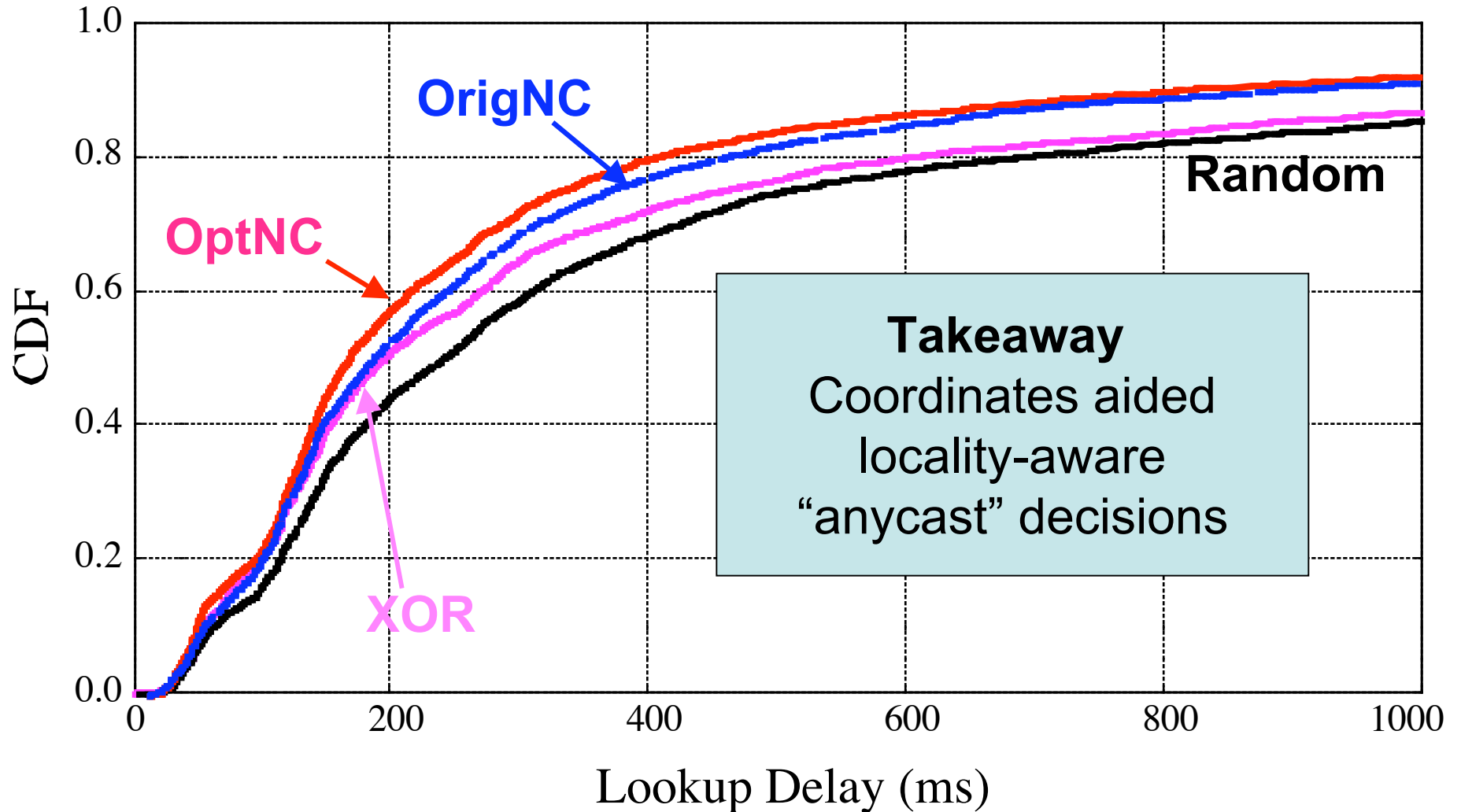
DHT Traversal: Evaluation

End-to-end benefit



DHT Traversal: Evaluation

End-to-end benefit



Conclusion

Methodology

- Coarse and fine statistics

Limited Horizon Problem

- Solution: **neighbor decay**

Locality-aware “anycast”-style decisions

- **DHT traversal**

Thanks & Pyxida



Check out **Pyxida**: <http://pyxida.sourceforge.net>

- Same code base
- Network Coordinate Service for PlanetLab
- Library Implementation of NC techniques
- Links to data

Thanks. Any Questions?

Jonathan Ledlie

jonathan@eecs.harvard.edu