



Peer-to-peer computing research a fad?

Frans Kaashoek

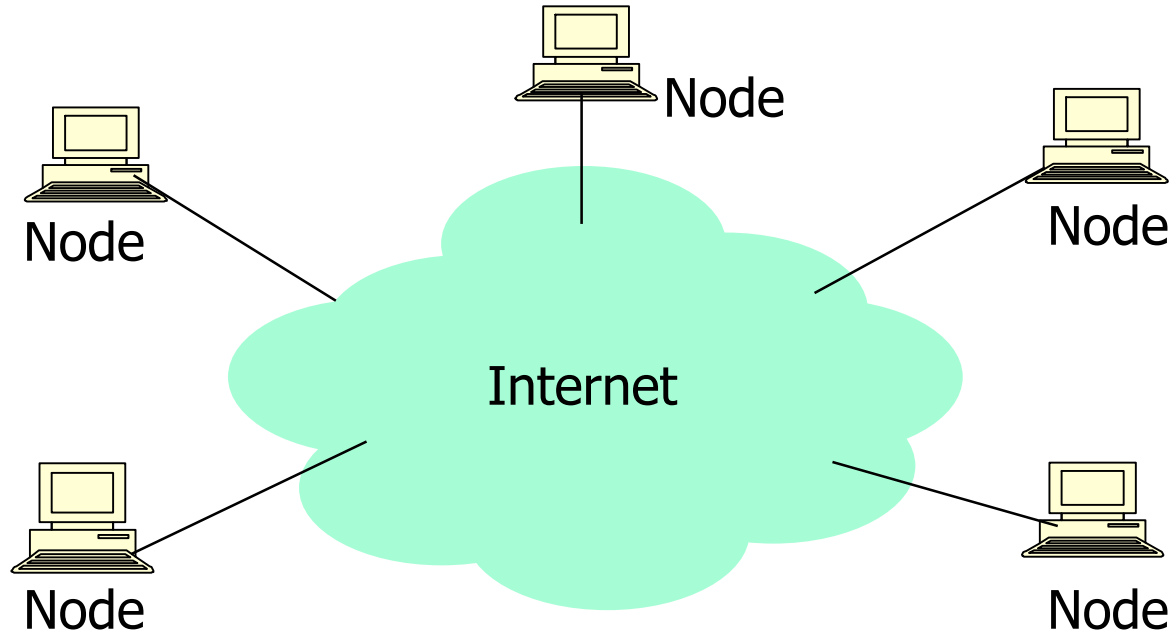
kaashoek@lcs.mit.edu

NSF Project IRIS

<http://www.project-iris.net>

Berkeley, ICSI, MIT, NYU, Rice

What is a P2P system?



- A distributed system architecture:
 - No centralized control
 - Nodes are symmetric in function
- Large number of unreliable nodes
- Enabled by technology improvements

P2P: an exciting social development

- Internet users cooperating to share, for example, music files
 - Napster, Gnutella, Morpheus, KaZaA, etc.
- Lots of attention from the popular press
 - “The ultimate form of democracy on the Internet”
 - “The ultimate threat to copy-right protection on the Internet”

How to build robust services?

- Many critical services use Internet
 - Hospitals, government agencies, etc.
- These services need to be robust
 - Node and communication failures
 - Load fluctuations (e.g., flash crowds)
 - Attacks (including DDoS)

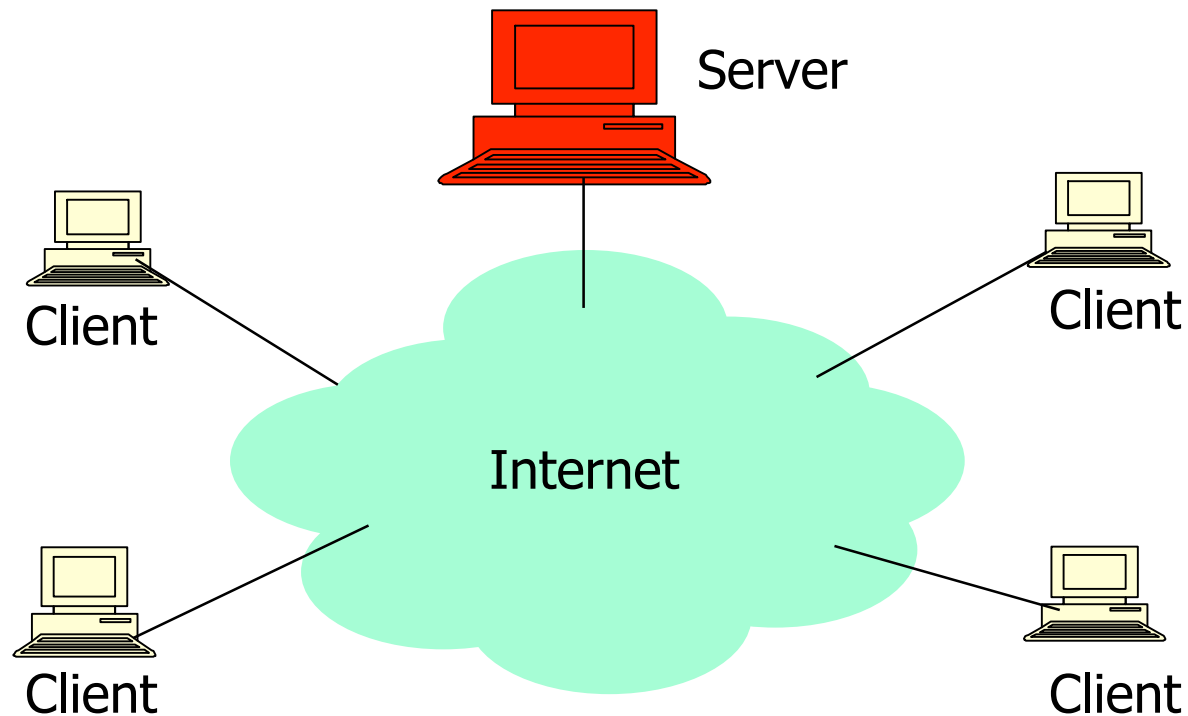
Example: peer-to-peer data archiver

- Back up hard disk to other users' machines
- Why?
 - Backup is usually difficult
 - Many machines have lots of spare disk space
- Requirements for cooperative archiver:
 - Divide data evenly among many computers
 - Find data
 - Don't lose any data
 - High performance: backups are big
- More challenging than sharing music!

The promise of P2P computing

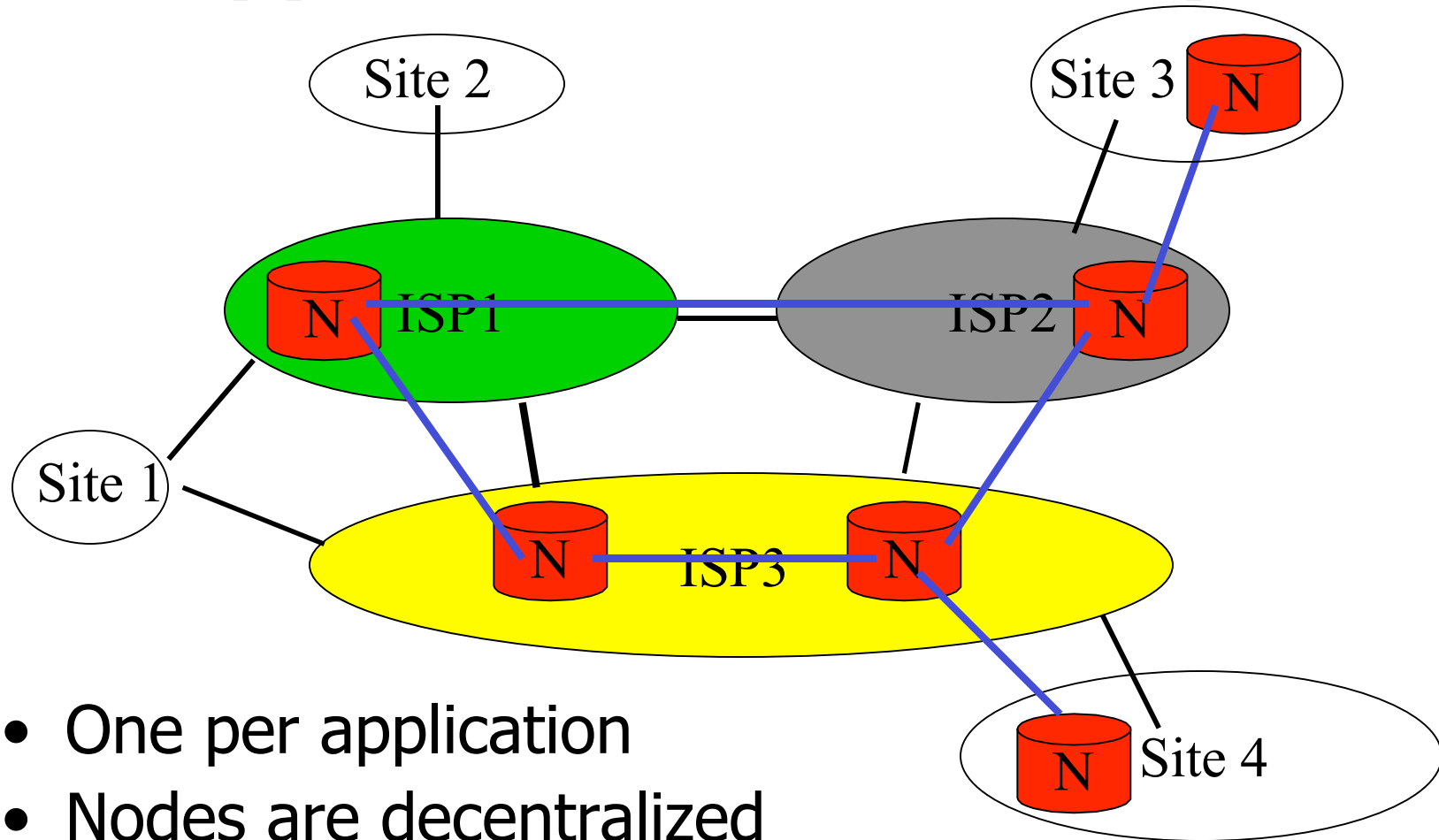
- Reliability: no central point of failure
 - Many replicas
 - Geographic distribution
- High capacity through parallelism:
 - Many disks
 - Many network connections
 - Many CPUs
- Automatic configuration
- Useful in public and proprietary settings

Traditional distributed computing: client/server



- Successful architecture, and will continue to be so
- Tremendous engineering necessary to make server farms scalable and robust

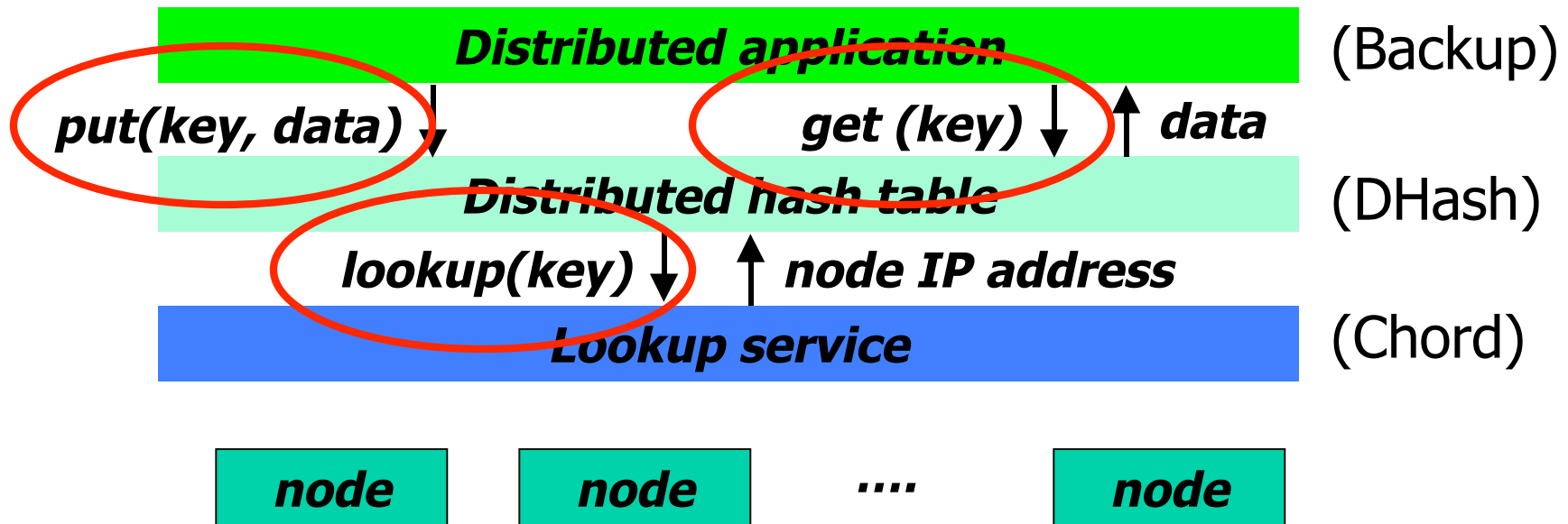
Application-level overlays



- One per application
- Nodes are decentralized
- NOC is centralized

P2P systems are overlay networks without central control

Distributed hash table (DHT)



- DHT distributes data storage over perhaps millions of nodes
- Many applications can use the same DHT infrastructure

A DHT has a good interface

- Put(key, value) and get(key) \rightarrow value
 - Simple interface!
- API supports a wide range of applications
 - DHT imposes no structure/meaning on keys
- Key/value pairs are persistent and global
 - Can store keys in other DHT values
 - And thus build complex data structures

A DHT makes a good *shared* infrastructure

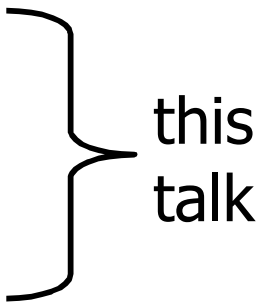
- Many applications can share one DHT service
 - Much as applications share the Internet
- Eases deployment of new applications
- Pools resources from many participants
 - Efficient due to statistical multiplexing
 - Fault-tolerant due to geographic distribution

Many applications for DHTs

- Streaming [SplitStream, ...]
- Content distribution networks [Coral, Squirrel, ..]
- File systems [CFS, OceanStore, PAST, Ivy, ...]
- Archival/Backup store [HiveNet, Mojo, Pastiche]
- Censor-resistant stores [Eternity, FreeNet, ..]
- DB query and indexing [PIER, ...]
- Event notification [Scribe]
- Naming systems [ChordDNS, Twine, ..]
- Communication primitives [I3, ...]

Common thread: data is location-independent

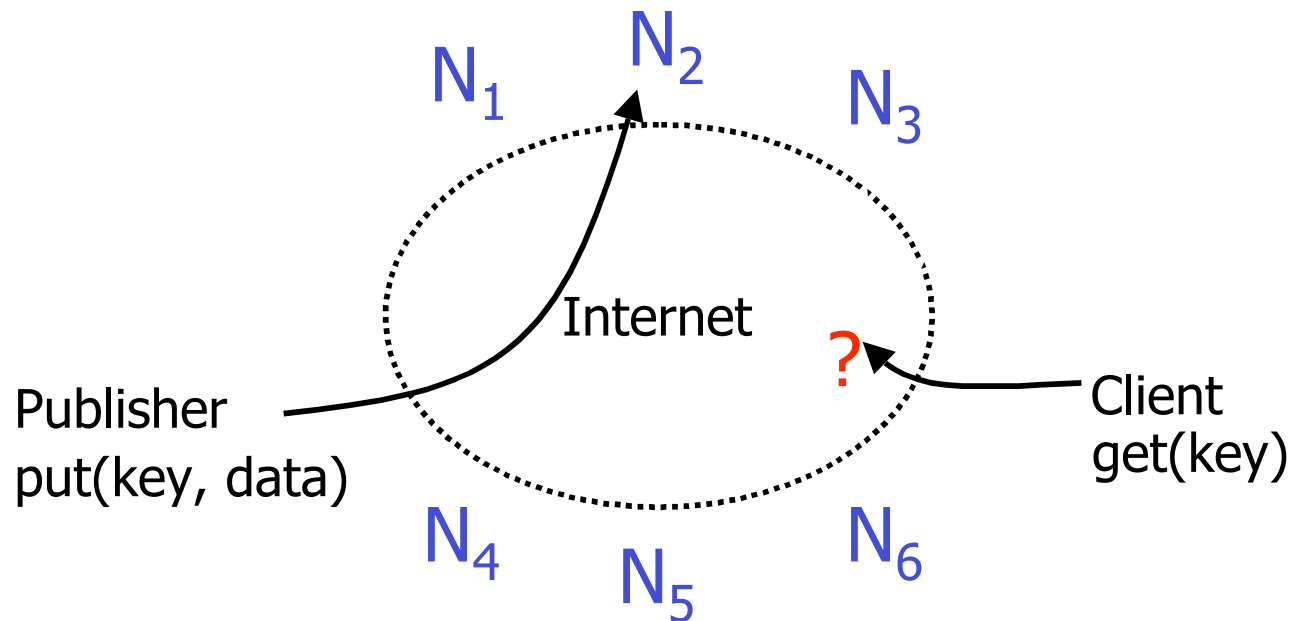
DHT implementation challenges

1. Scalable lookup
 2. Handling failures
 3. Network-awareness for performance
 4. Data integrity
 5. Coping with systems in flux
 6. Balance load (flash crowds)
 7. Robustness with untrusted participants
 8. Heterogeneity
 9. Anonymity
 10. Indexing
- 
- this
talk

Goal: simple, provably-good algorithms

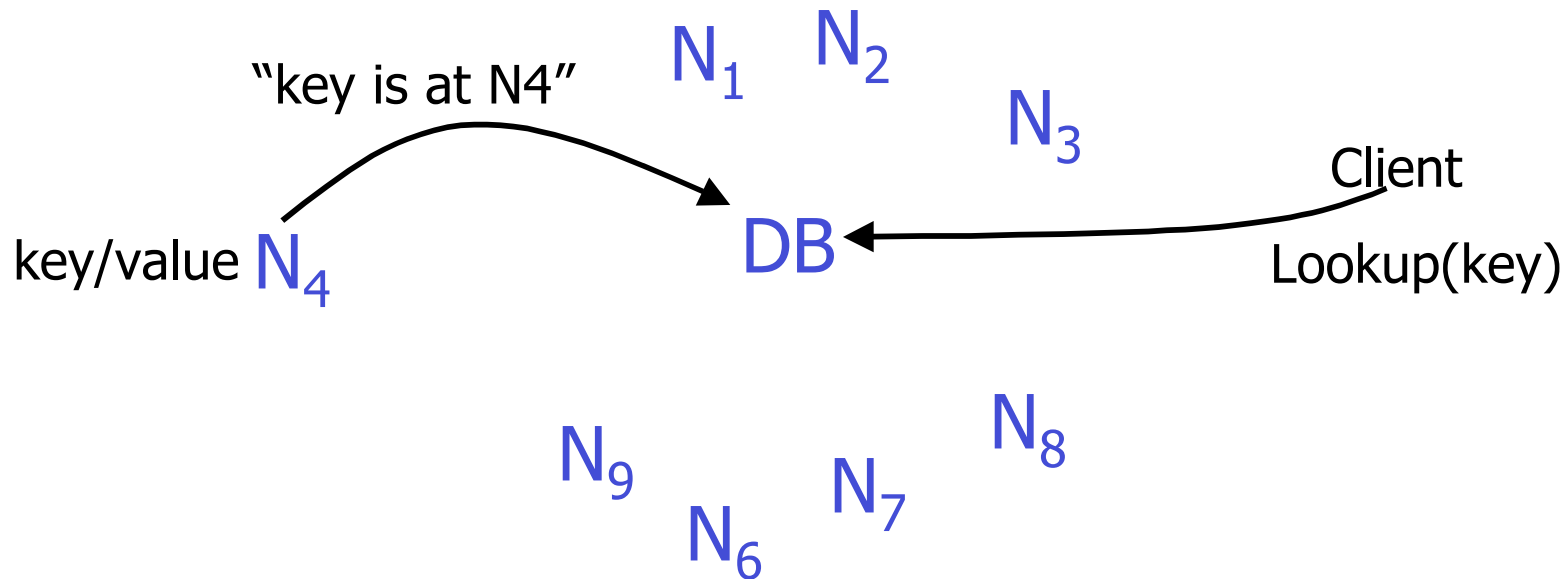
1. The lookup problem

How do you find the node responsible for a key?



Centralized lookup (Napster)

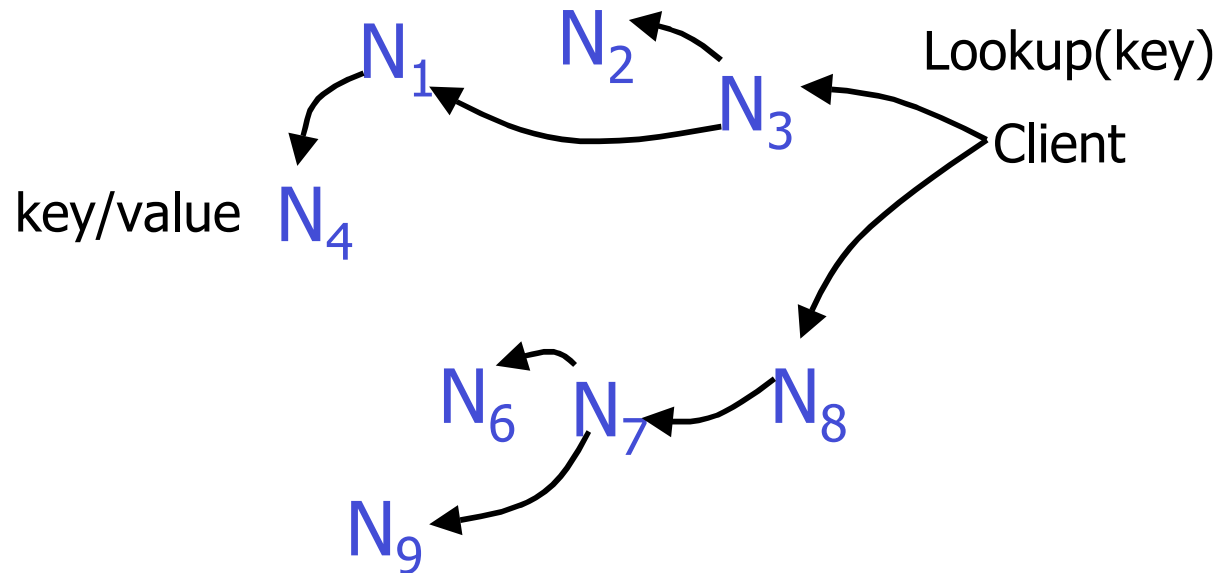
- Central server knows where all keys are



- Simple, but $O(N)$ state for server
- Server can be attacked (lawsuit killed Napster)

Flooding queries (Gnutella)

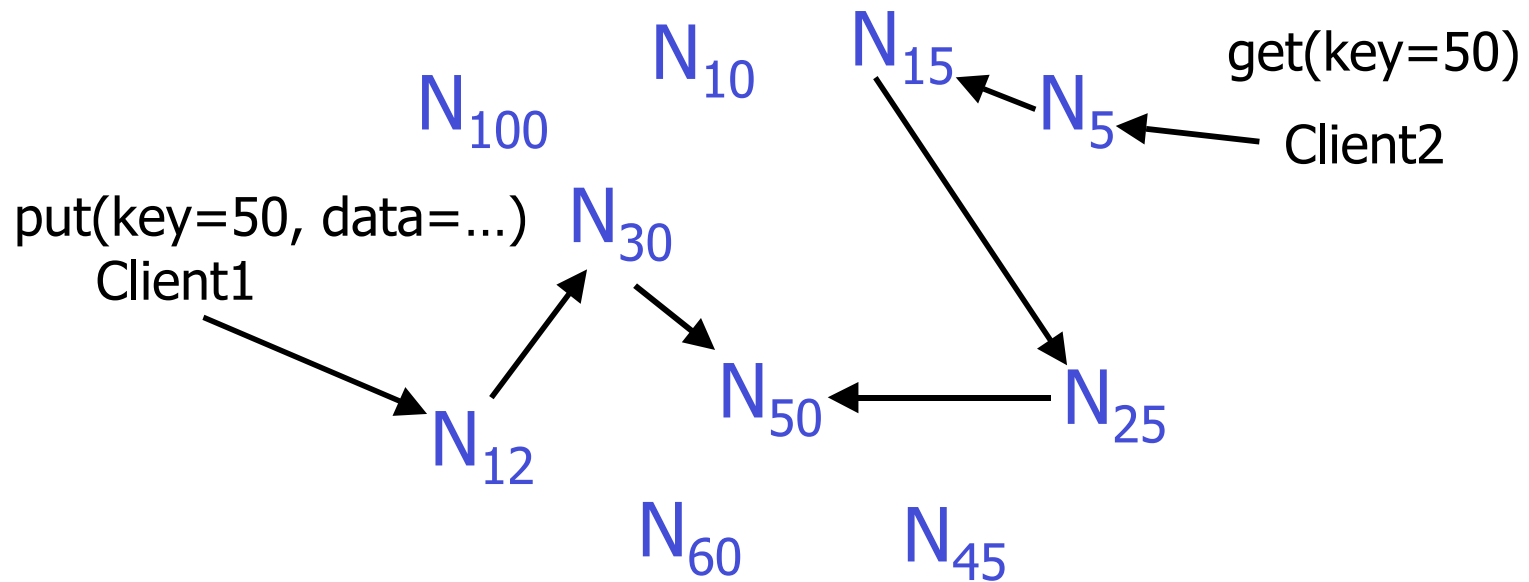
- Lookup by asking every node



- Robust but expensive: $O(N)$ messages per lookup

Routed lookups

- Each node has a numeric identifier (ID)
- Route lookup(key) in ID space

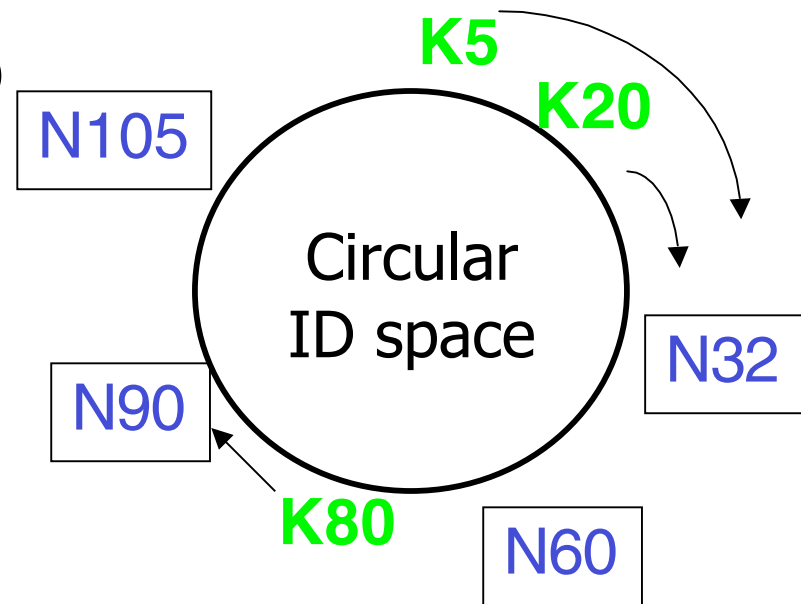


Routing algorithm goals

- Fair (balanced) key range assignments
- Easy to maintain routing table
 - Dead nodes lead to time outs
- Small number of hops to route message
 - Low stretch
- Stay robust despite rapid change
- Solutions:
 - Small table and $\text{Log}(N)$ hops: CAN, Chord, Kademlia, Pastry, Tapestry, Koorde, etc.
 - Big table and One/two hops: Kellips, EpiChord, etc.

Chord key assignments

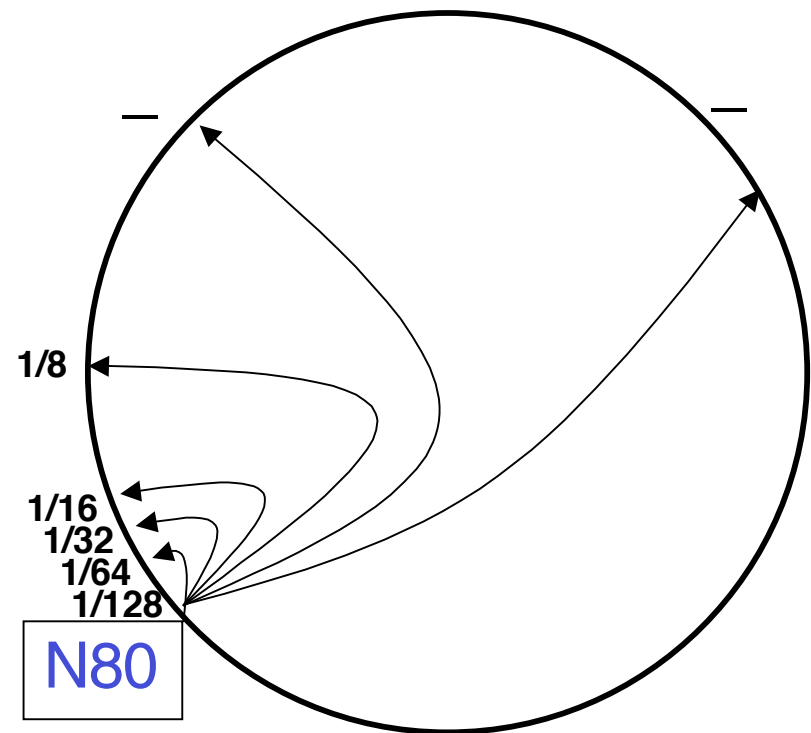
- Each node has 160-bit ID
- ID space is circular
- Data keys are also IDs
- A key is stored on the next higher node
- Good load balance
- Easy to find keys slowly



(N90 is responsible for keys K61 through K90)

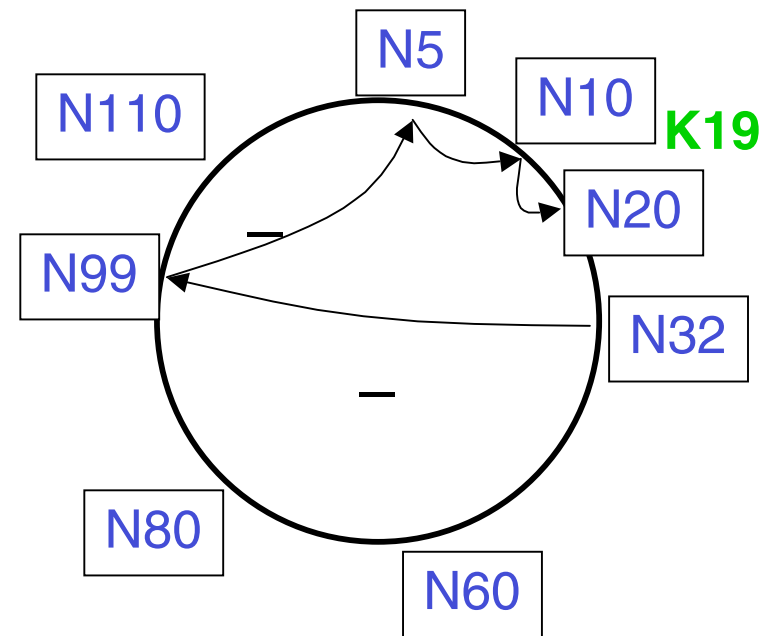
Chord's routing table

- Routing table lists nodes:
 - $_$ way around circle
 - $_$ way around circle
 - $1/8$ way around circle
 - ...
 - next around circle
- The table is small:
 - $\log N$ entries



Chord lookups take $O(\log N)$ hops

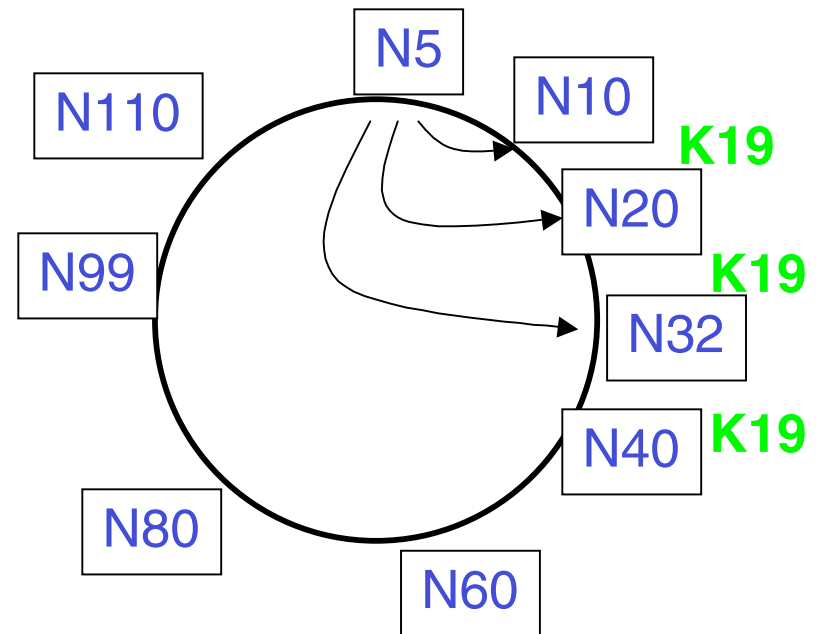
- Each step goes at least halfway to the destination
- Lookups are fast:
 - $\log N$ steps



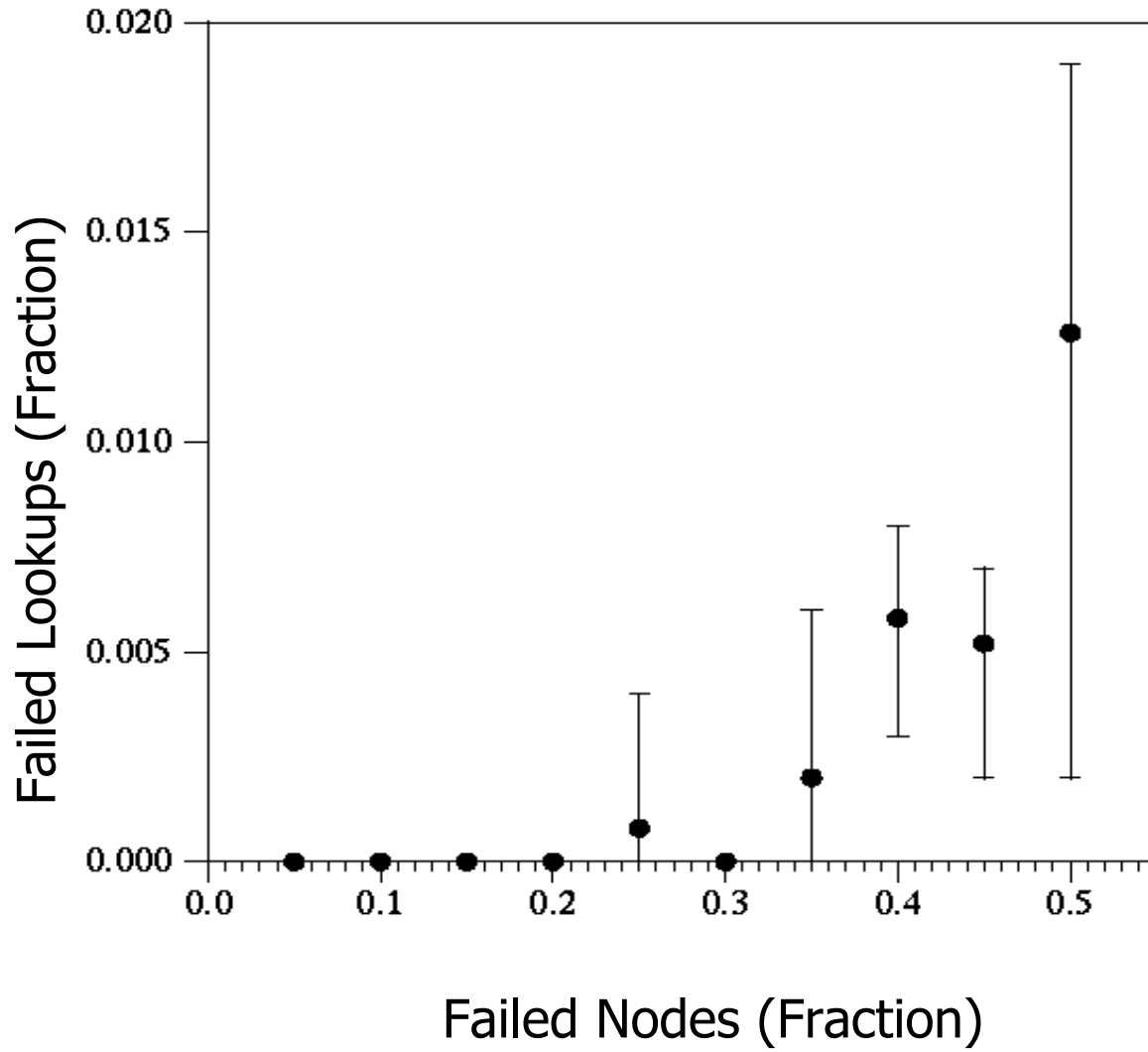
Node **N32** looks up key **K19**

2. Handling failures: redundancy

- Each node knows about next r nodes on circle
- Each key is stored by the r nodes after it on the circle
- To save space, each node stores only a piece of the block
- Collecting half the pieces is enough to reconstruct the block



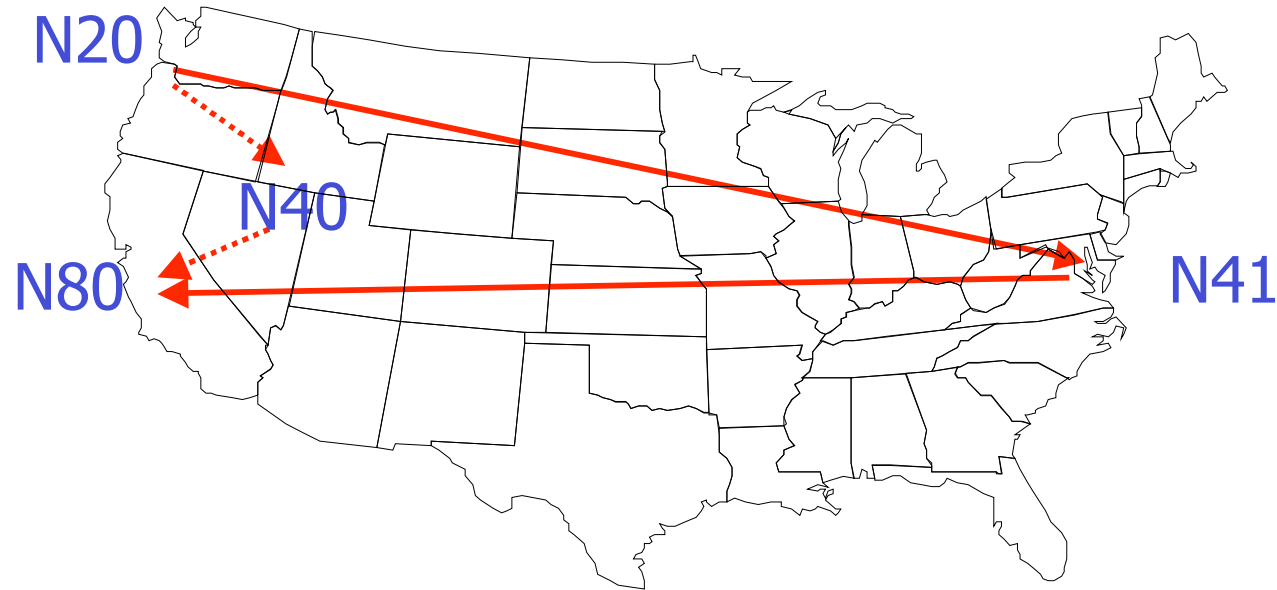
Redundancy handles failures



- 1000 DHT nodes
- Average of 5 runs
- 6 replicas for each key

- Kill fraction of nodes
- Then measure how many lookups fail
- All replicas must be killed for lookup to fail

3. Reducing Chord lookup delay



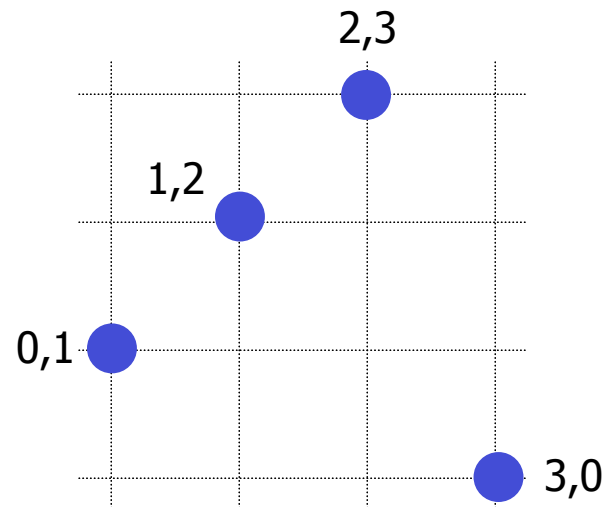
- N20's routing table may point to distant nodes
- Any node with a closer ID could be used in route
- Knowing about proximity could help performance
 - Key challenge: avoid $O(N^2)$ pings

Estimate latency using synthetic coordinate system

- Each node estimates its position
- Position = (x,y) : "synthetic coordinates"
- x and y units are time (milliseconds)
- Distance between two nodes' coordinates predicts network latency
 - Challenges: triangle equality, etc.

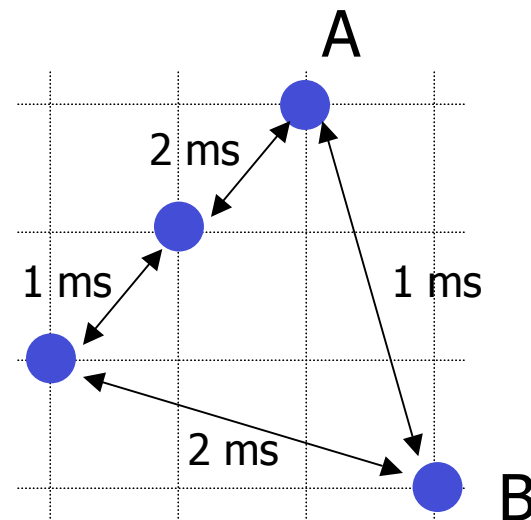
Vivaldi synthetic coordinates

- Each node starts with a random incorrect position



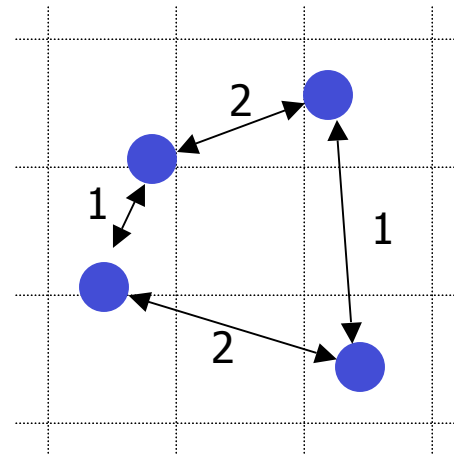
Vivaldi synthetic coordinates

- Each node starts with a random incorrect position
- Each node "pings" a few other nodes to measure network latency (distance)



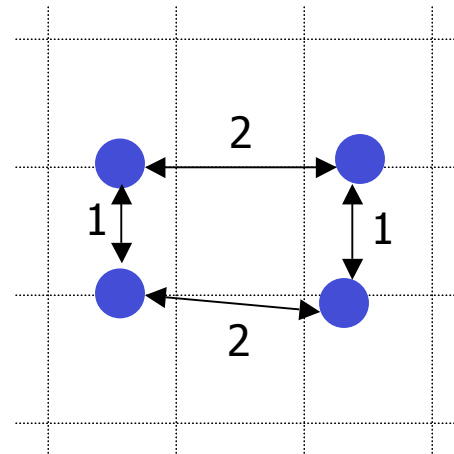
Vivaldi synthetic coordinates

- Each node starts with a random incorrect position
- Each node "pings" a few other nodes to measure network latency (distance)
- Each nodes "moves" to cause measured distances to match coordinates



Vivaldi synthetic coordinates

- Each node starts with a random incorrect position
- Each node "pings" a few other nodes to measure network latency (distance)
- Each nodes "moves" to cause measured distance to match coordinates
- Minimize force in spring network

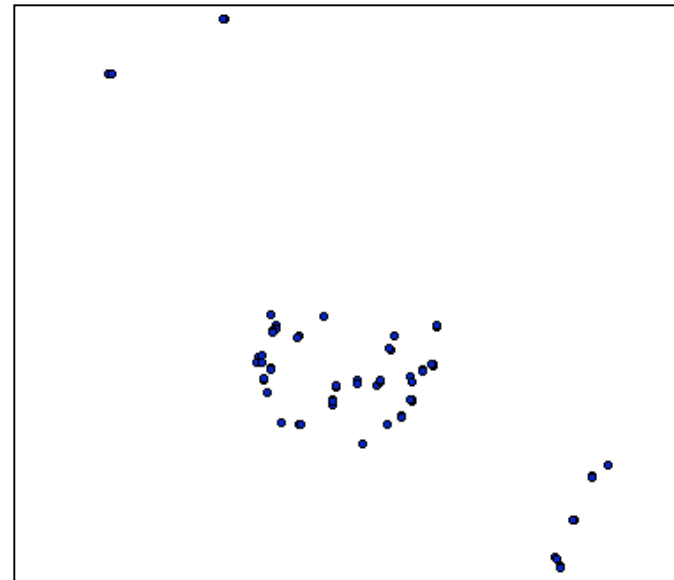
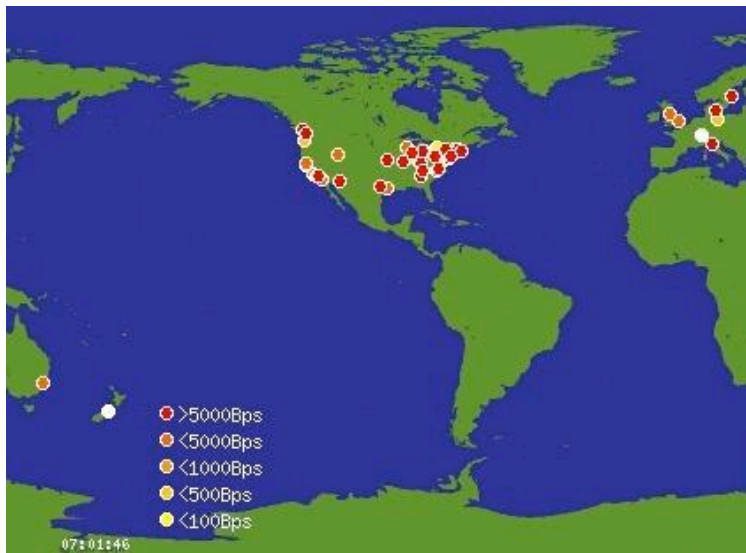


Vivaldi in action

- Execution on 86 PlanetLab Internet hosts
- Each host only pings a few other random hosts
- Most hosts find useful coordinates after a few dozen pings
- Use 3D coordinates

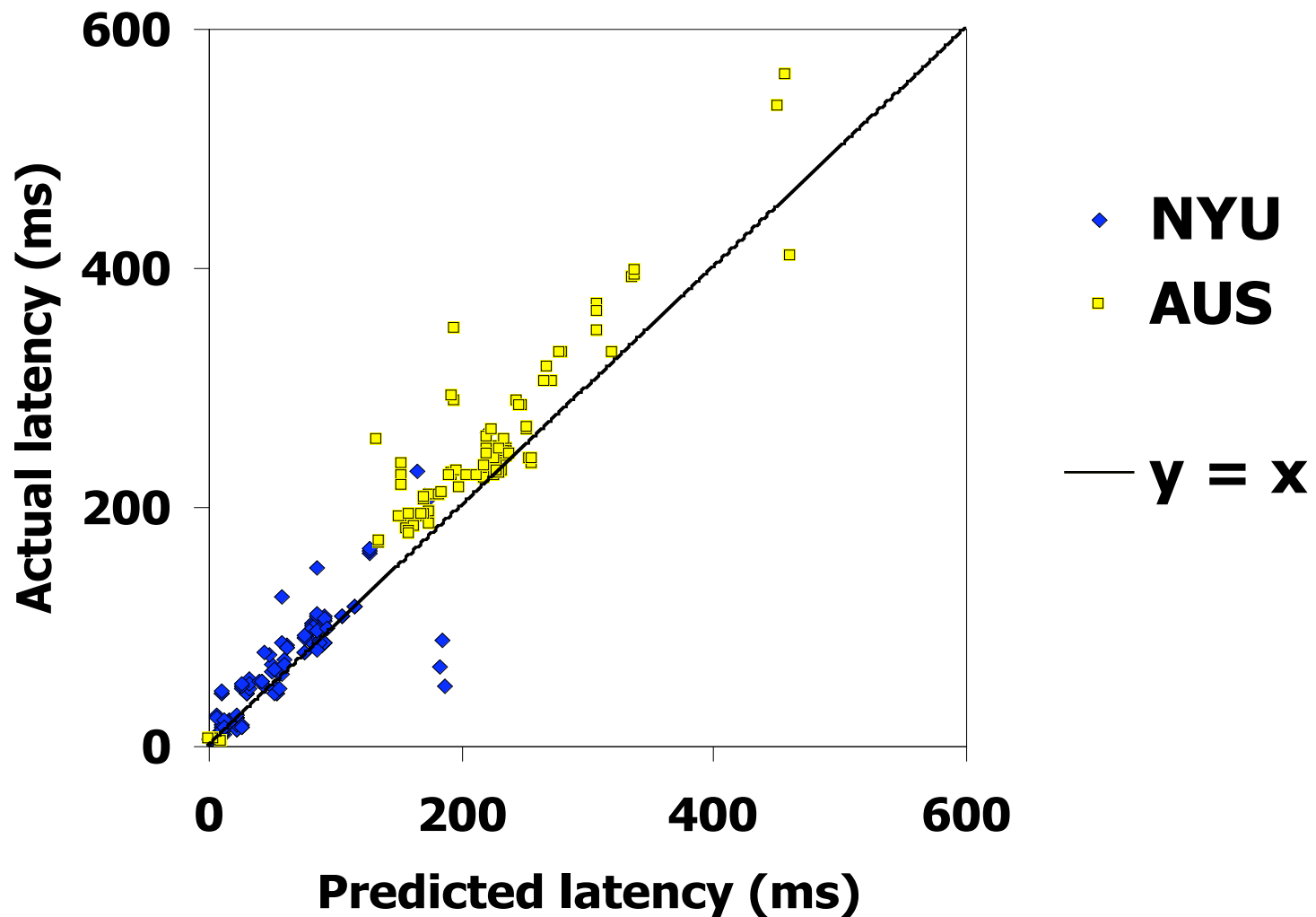


Vivaldi vs. network coordinates



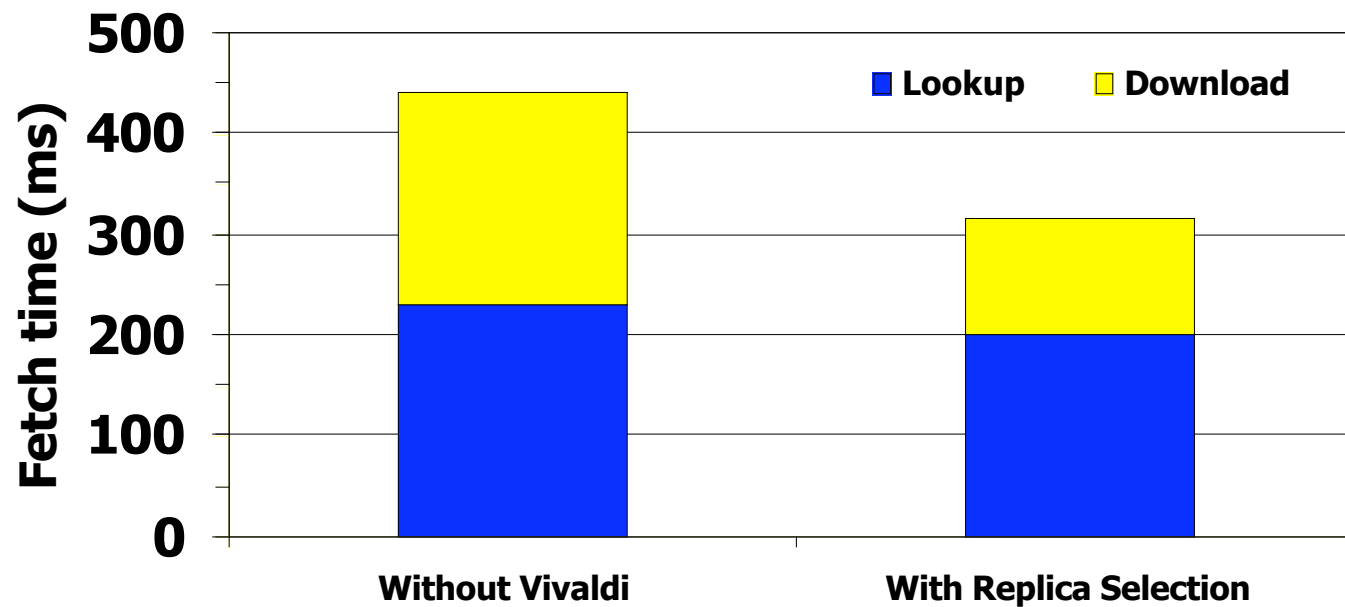
- Vivaldi's coordinates match geography well
 - over-sea distances shrink (faster than over-land)
 - orientation of Australia and Europe is "wrong"
- Simulations confirm results for larger networks

Vivaldi predicts latency well



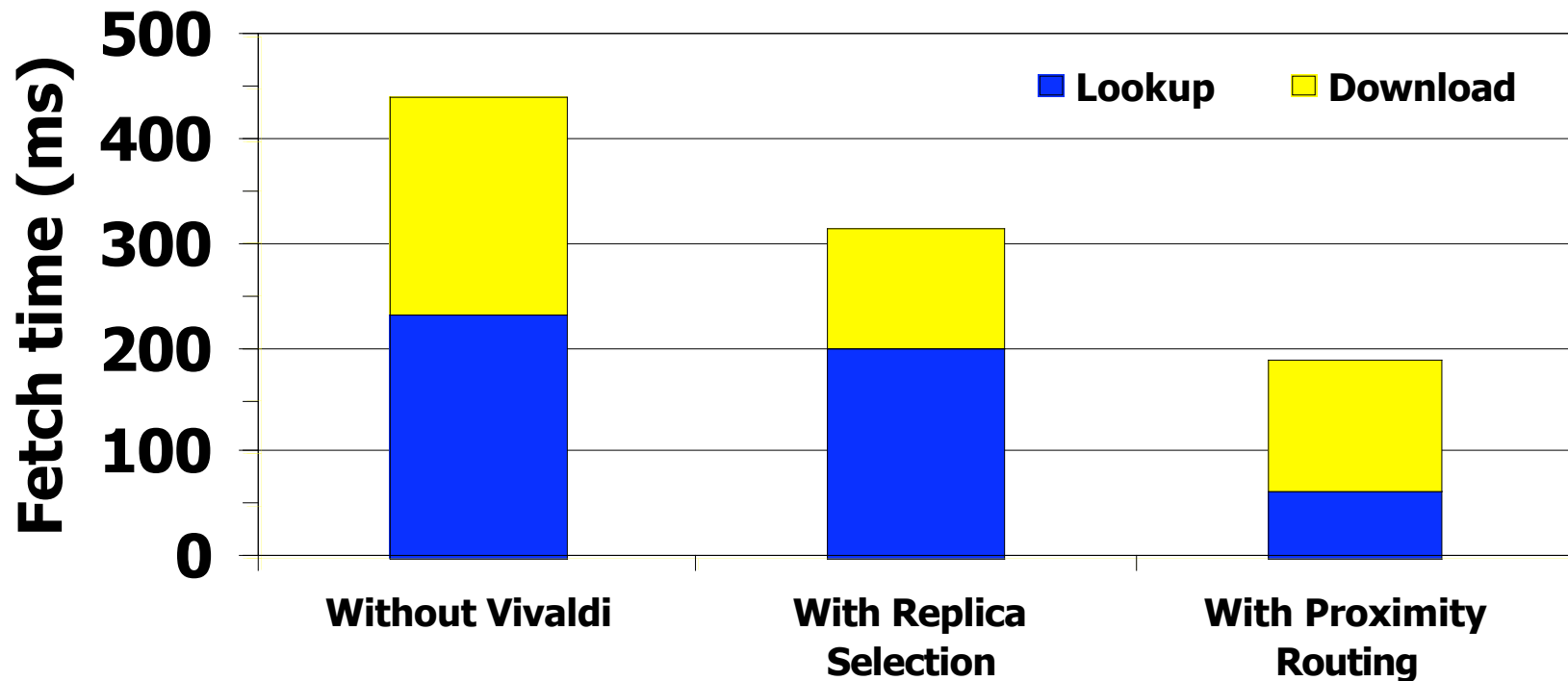
DHT fetches with Vivaldi (1)

- Choose the lowest-latency copy of data



DHT fetches with Vivaldi (2)

- Choose the lowest-latency copy of data
- Route Chord lookup through nearby nodes



DHT implementation summary

- Chord for looking up keys
- Replication at successors for fault tolerance
- Fragmentation and erasure coding to reduce storage space
- Vivaldi network coordinate system for
 - Server selection
 - Proximity routing

Backup system on DHT

- Store file system image snapshots as hash trees
 - Can access daily images directly
 - Yet images share storage for common blocks
 - Only incremental storage cost
 - Encrypt data
- User-level NFS server parses file system images to present dump hierarchy
- Application is ignorant of DHT challenges
 - DHT is just a reliable block store

Future work

DHTs

- Improve performance
- Handle untrusted nodes

Vivaldi

- Does it scale to larger and more diverse networks?

Apps

- Need lots of interesting applications

Philosophical questions

- How decentralized should systems be?
 - Gnutella versus content distribution network
 - Have a bit of both? (e.g., CDNs)
- Why does the distributed systems community have more problems with decentralized systems than the networking community?
 - “A distributed system is a system in which a computer you don’t know about renders your own computer unusable”
 - Internet (BGP, NetNews)

Related Work

Lookup algs

- CAN, Kademlia, Koorde, Pastry, Tapestry, Viceroy, ...

DHTs

- DOLR, Past, OpenHash...

Network coordinates and springs

- GNP, Hoppe's mesh relaxation

Applications

- Ivy, OceanStore, Pastiche, Twine, ...

Conclusions

- Peer-to-peer promises some great properties
- DHTs are a good way to build peer-to-peer applications:
 - Easy to program
 - Single, shared infrastructure for many applications
 - Robust in the face of failures
 - Scalable to large number of servers
 - Self configuring
 - Can provide high performance

<http://www.project-iris.net>