5 Lectures on Peer-peer and Application-level Networking Advanced System Topics 2003

Presented by Jon Crowcroft

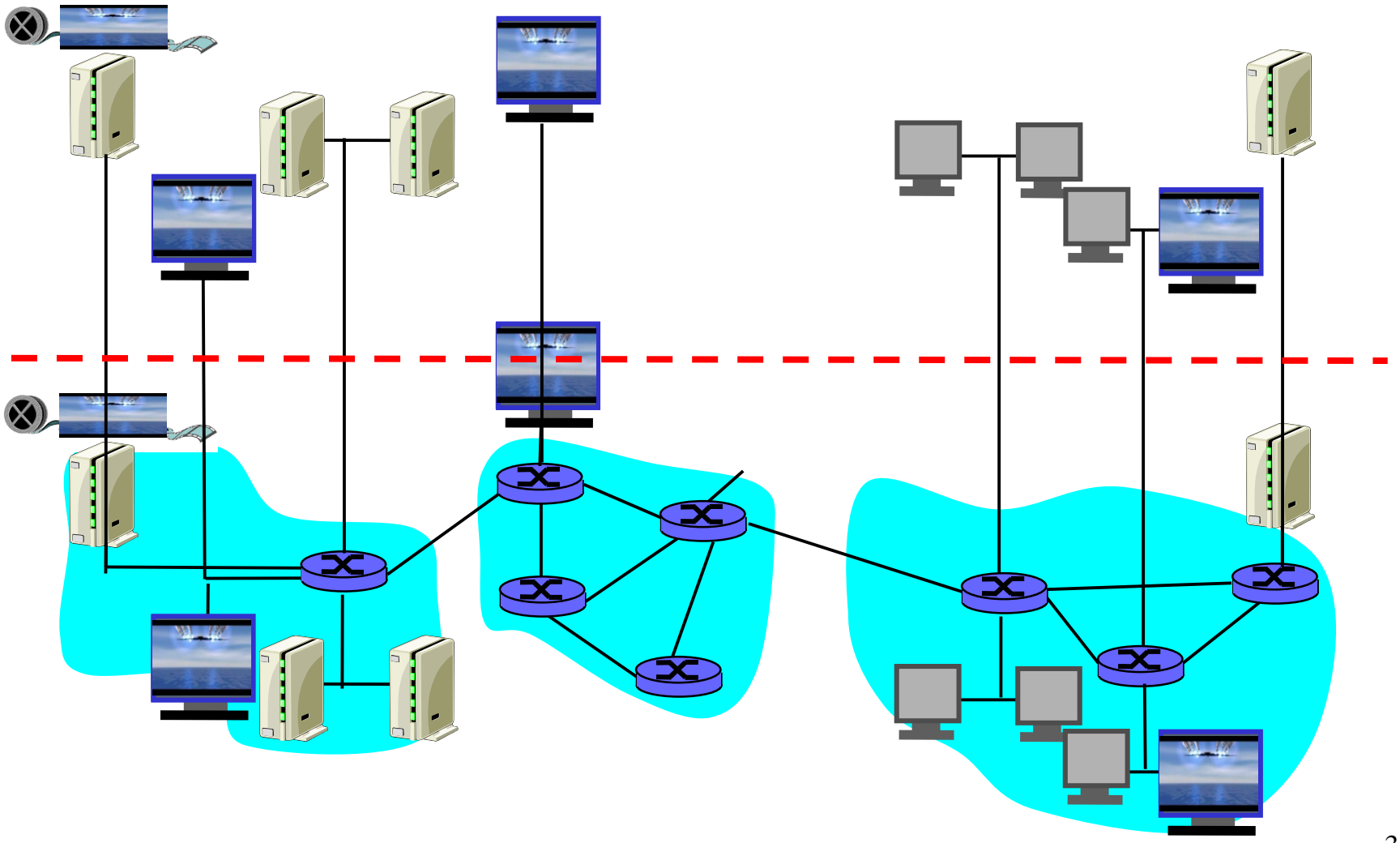
Based ***strongly*** on material by

Jim Kurose,Brian Levine,Don Towsley, and the class of 2001 for the Umass Comp Sci 791N course..

0.Introduction

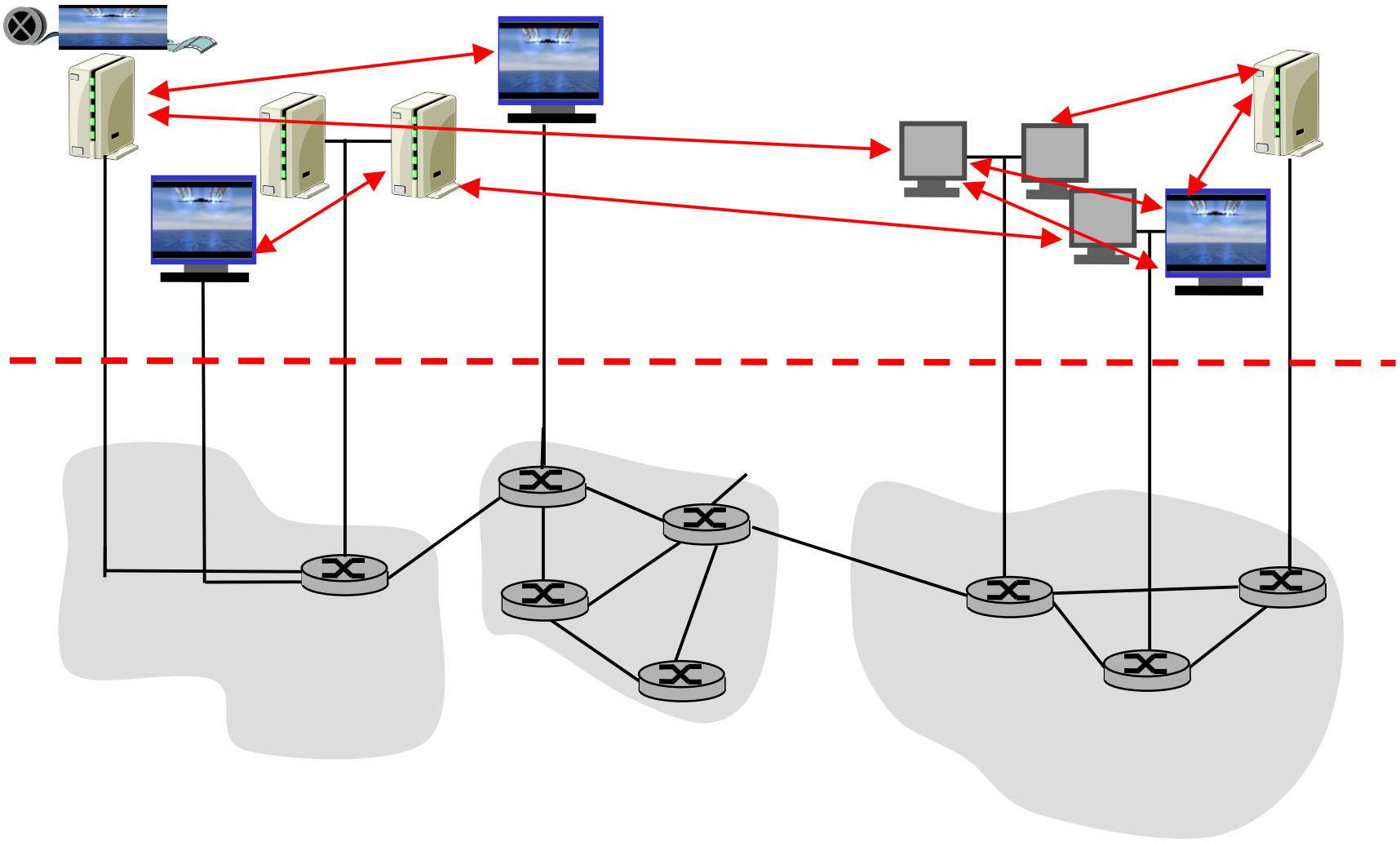
- Background
- Motivation
- outline of the lectures

Peer-peer networking



Peer-peer networking

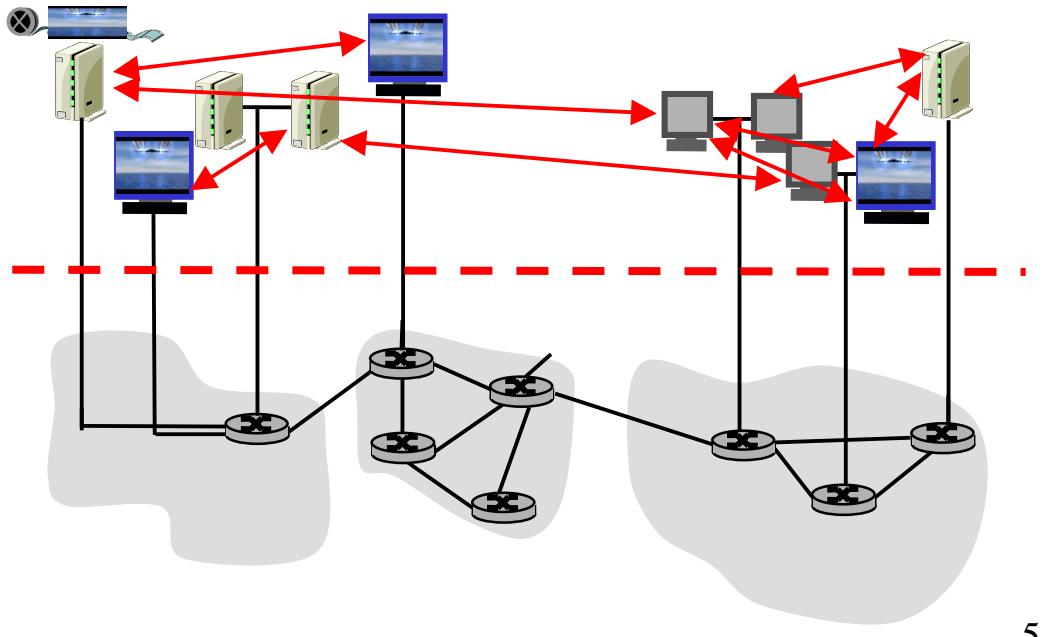
Focus at the application level



Peer-peer networking

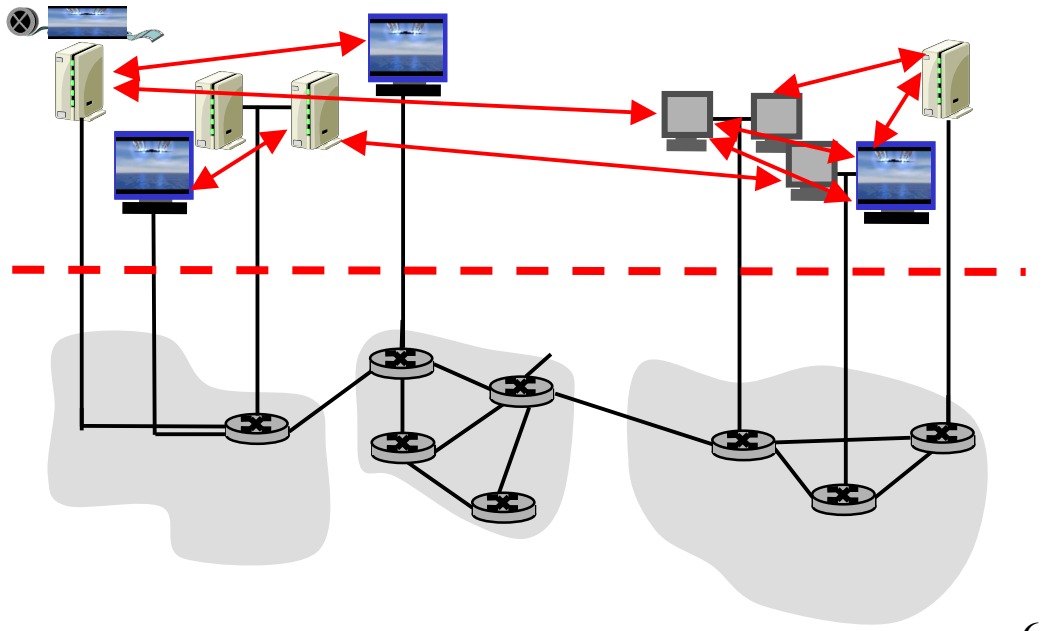
Peer-peer applications

- Napster, Gnutella, Freenet: file sharing
- ad hoc networks
- multicast overlays (e.g., video distribution)



Peer-peer networking

- Q: What are the new technical challenges?
- Q: What new services/applications enabled?
- Q: Is it just "networking at the application-level"?
  - "There is nothing new under the Sun" (William Shakespeare)



Course Lecture Contents

- Introduction
  - Client-Server v. P2P
  - File Sharing/Finding
  - Napster
  - Gnutella
- Overlays and Query Routing
  - RON
  - Freenet
  - Publius
- Distributed Hash Tables
  - Chord
  - CAN
  - Tapestry
- Middleware
  - JXTA
  - ESM
  - Overcast
- Applications
  - Storage
  - Conferencing

Client Server v. Peer to Peer(1)

- RPC/RMI
- Synchronous
- Assymmetric
- Emphasis on language integration and binding models (stub *IDL/XDR* compilers etc)
- Kerberos style security – access control, crypto

- Messages
- Asynchronous
- Symmetric
- Emphasis on service location, content addressing, application layer routing.
- Anonymity, high availability, integrity.
- Harder to get right☺

Client Server v. Peer to Peer(2) RPC

```
Cli_call(args)

Srv_main_loop()
{
  while(true) {
   deque(call)
   switch(call.procid)
   case 0:
   call.ret=proc1(call.args)
   case 1:
   call.ret=proc2(call.args)
   … … …
   default:
   call.ret = exception
   }
}
```

Client Server v. Peer to Peer(3) P2P

```
Peer_main_loop()
{
   while(true) {
   await(event)
   switch(event.type) {
   case timer_expire: do some p2p work()
        randomize timers
        break;
   case inbound message: handle it
        respond
        break;
   default: do some book keeping
        break;
   }
}
```

Peer to peer systems actually old

- IP routers are peer to peer.
- Routers discover topology, and maintain it
- Routers are neither client nor server
- Routers continually chatter to each other
- Routers are fault tolerant, inherently
- Routers are autonomous

Peer to peer systems

- Have no distinguished role
- So no single point of bottleneck or failure.
- However, this means they need distributed algorithms for
  - Service discovery (name, address, route, metric, etc)
  - Neighbour status tracking
  - Application layer routing (based possibly on content, interest, etc)
  - Resilience, handing link and node failures
  - Etc etc etc

Ad hoc networks and peer2peer

- Wireless ad hoc networks have many similarities to peer to peer systems
- No *a priori knowledge*
- No given infrastructure
- Have to construct it from "thin air"!
- Note for later – wireless☺

Overlays and peer 2 peer systems

- P2p technology is often used to create overlays which offer services that could be offered in the IP level
- Useful **deployment** strategy
- Often economically a way around other barriers to deployment
- IP Itself was an overlay (on telephone core infrastructure)
- Evolutionary path!!!

Rest of lectures oriented 14 case studies from literature

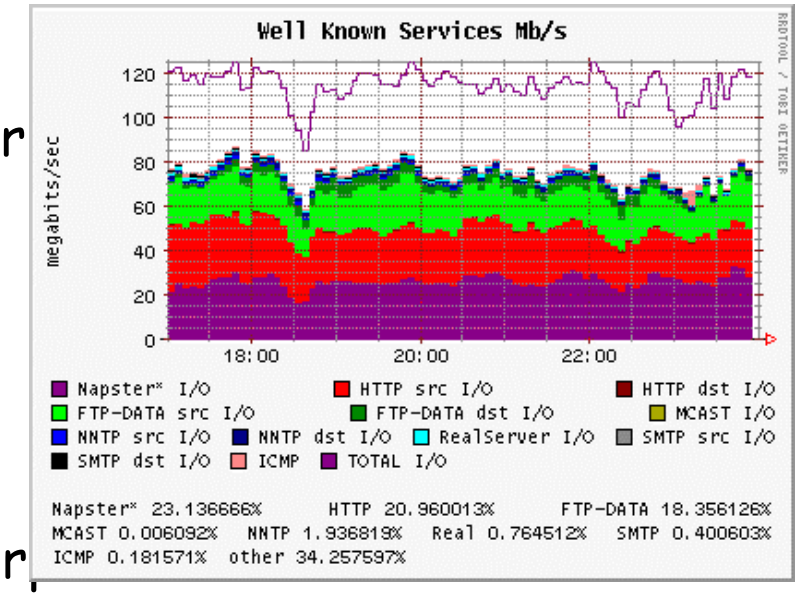
- Piracy^H^H^H^H^H^content sharing ☺
- Napster
- Gnutella
- Freenet
- Publius
- etc

1. NAPSTER

- The most (in)famous
- Not the first (c.f. probably Eternity, from Ross Anderson in Cambridge)
- But instructive for what it gets right, and
- Also wrong…
- Also has a political message…and economic and legal…etc etc etc

Napster

- program for sharing files over the Internet
- a "disruptive" application/technology?
- history:
  - 5/99: Shawn Fanning (freshman, Northeasten U.) founds Napster Online music service
  - 12/99: first lawsuit
  - 3/00: 25% UWisc traffic Napster
  - 2000: est. 60M users
  - 2/01: US Circuit Court of Appeals: Napster knew users violating copyright laws
  - 7/01: # simultaneous online users: Napster 160K, Gnutella: 40K, Mor



Napster: how does it work

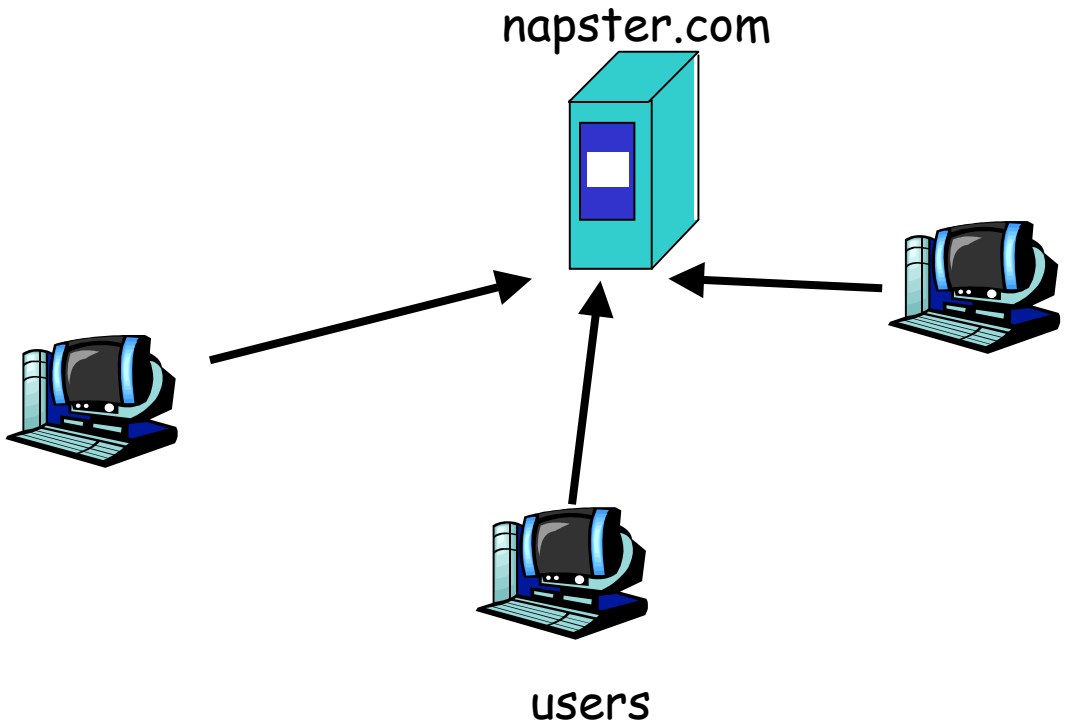
Application-level, client-server protocol over point-to-point TCP

Four steps:

- Connect to Napster server
- Upload your list of files (push) to server.
- Give server keywords to search the full list with.
- Select "best" of correct answers. (pings)

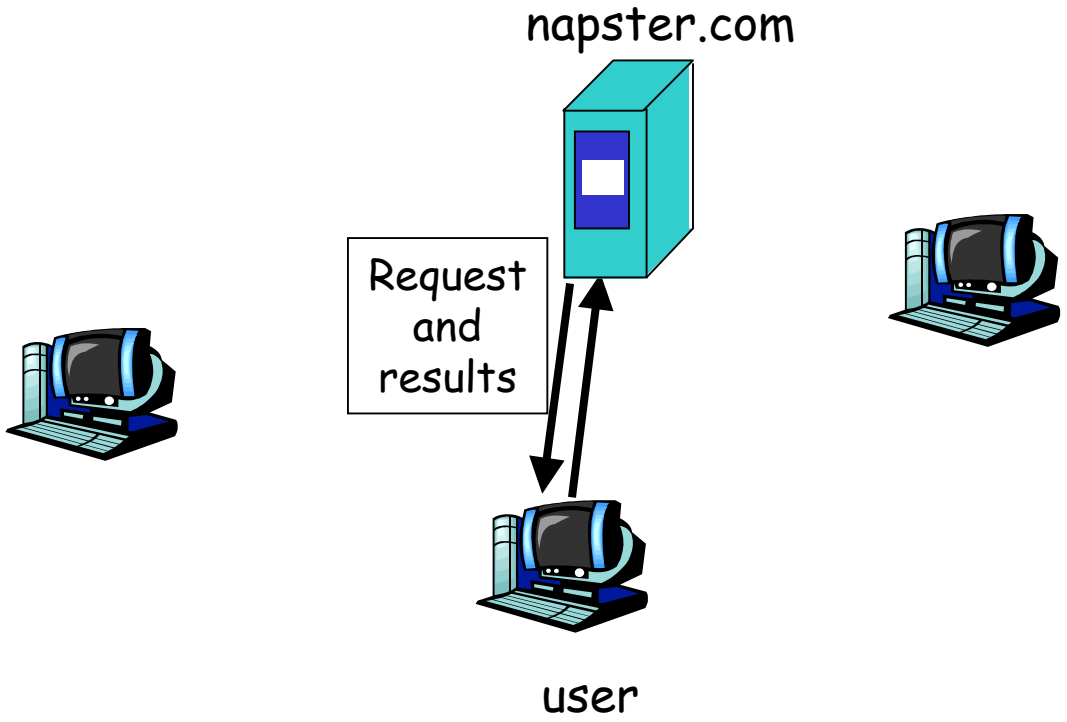
Napster

1. File list is uploaded



Napster

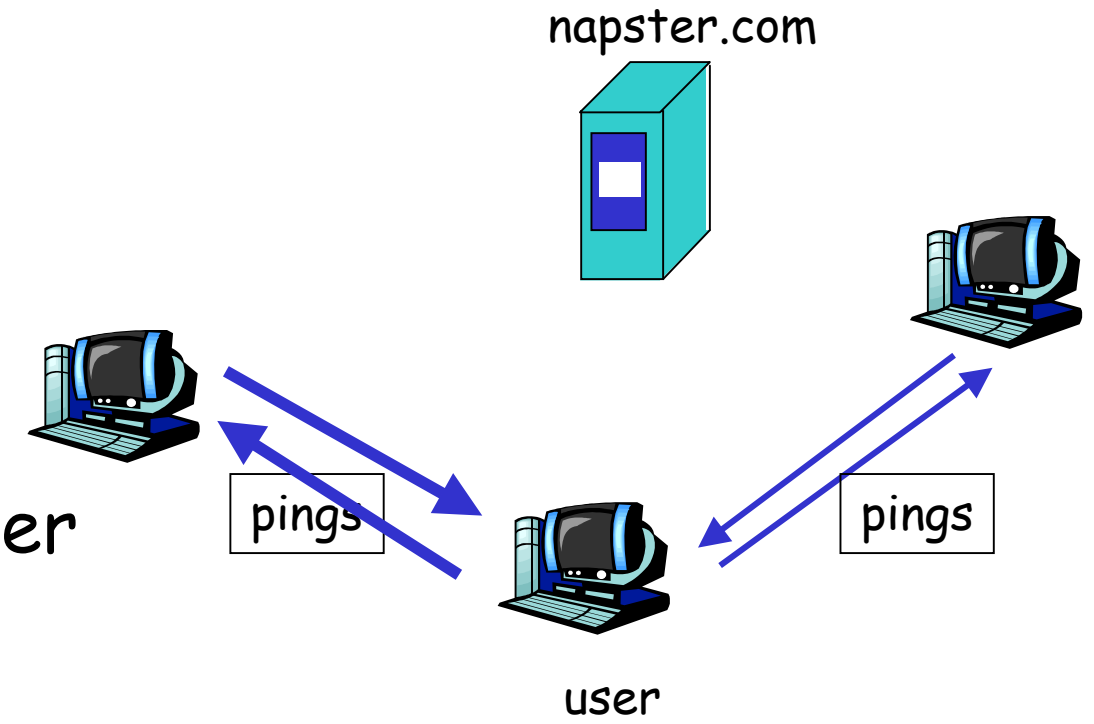
2. User requests search at server.



Napster

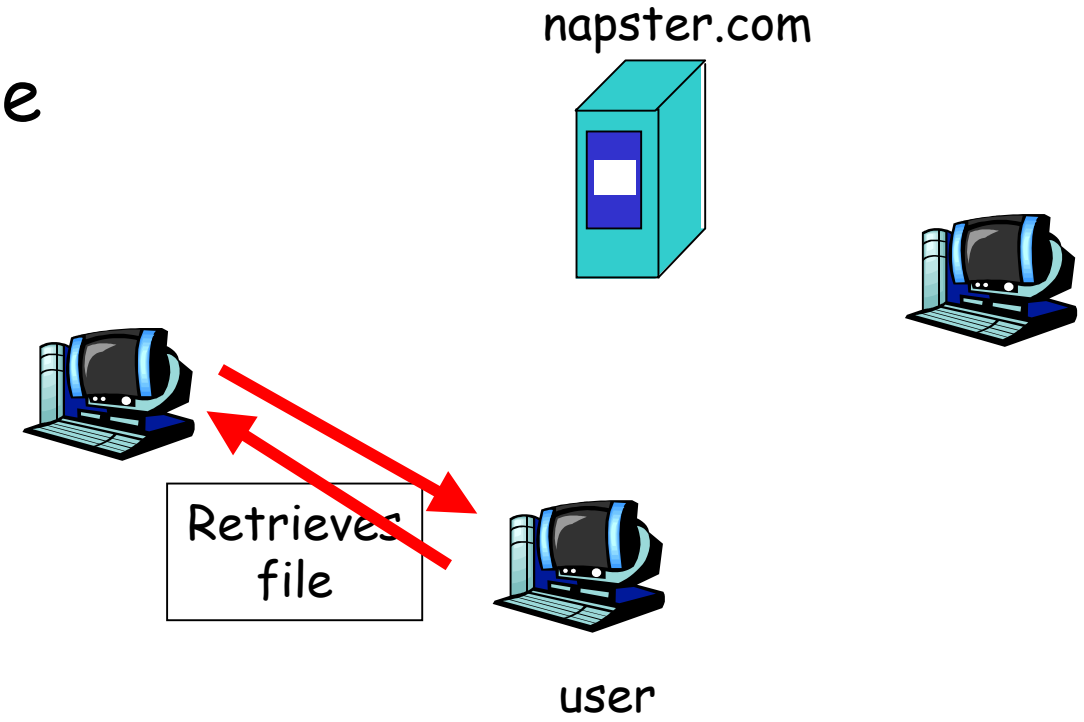
3. User pings hosts that apparently have data.

   Looks for ***best*** transfer rate.



Napster

4. User retrieves file



Napster messages

General Packet Format

[chunksize] [chunkinfo] [data...]

CHUNKSIZE:
Intel-endian 16-bit integer
size of [data...] in bytes

CHUNKINFO: (hex)
Intel-endian 16-bit integer.

00 - login rejected
02 - login requested
03 - login accepted
0D - challenge? (nuprin1715)
2D - added to hotlist
2E - browse error (user isn't online!)
2F - user offline

5B - whois query
5C - whois result
5D - whois: user is offline!
69 - list all channels
6A - channel info
90 - join channel
91 - leave channel
.....

From: http://david.weekly.org/code/napster.php3

Napster: requesting a file

SENT to server (after logging in to server)
    2A 00 CB 00 username
        "C:\MP3\REM - Everybody Hurts.mp3"
RECEIVED
    5D 00 CC 00 username
        2965119704 (IP-address backward-form = A.B.C.D)
        6699 (port)
        "C:\MP3\REM - Everybody Hurts.mp3" (song)
        (32-byte checksum)
        (line speed)
[connect to A.B.C.D:6699]
RECEIVED from client
    31 00 00 00 00 00
SENT to client
    GET
RECEIVED from client
    00 00 00 00 00 00

SENT to client
    Myusername
    "C:\MP3\REM - Everybody Hurts.mp3"
    0 (port to connect to)
RECEIVED from client
    (size in bytes)
SENT to server
    00 00 DD 00 (give go-ahead thru server)
RECEIVED from client
    [DATA]

Napster: architecture notes

- centralized server:
  - single logical point of failure
  - can load balance among servers using DNS rotation
  - potential for congestion
  - Napster "in control" (freedom is an illusion)
- no security:
  - passwords in plain text
  - no authentication
  - no anonymity

2 Gnutella

- Napster fixed
- Open Source
- Distributed
- Still very political…

Gnutella

- peer-to-peer networking: applications connect to peer applications
- focus: decentralized method of searching for files
- each application instance serves to:
  - store selected files
  - route queries (file searches) from and to its neighboring peers
  - respond to queries (serve file) if file stored locally
- Gnutella history:
  - 3/14/00: release by AOL, almost immediately withdrawn
  - too late: 23K users on Gnutella at 8 am this AM
  - many iterations to fix poor initial design (poor design turned many people off)

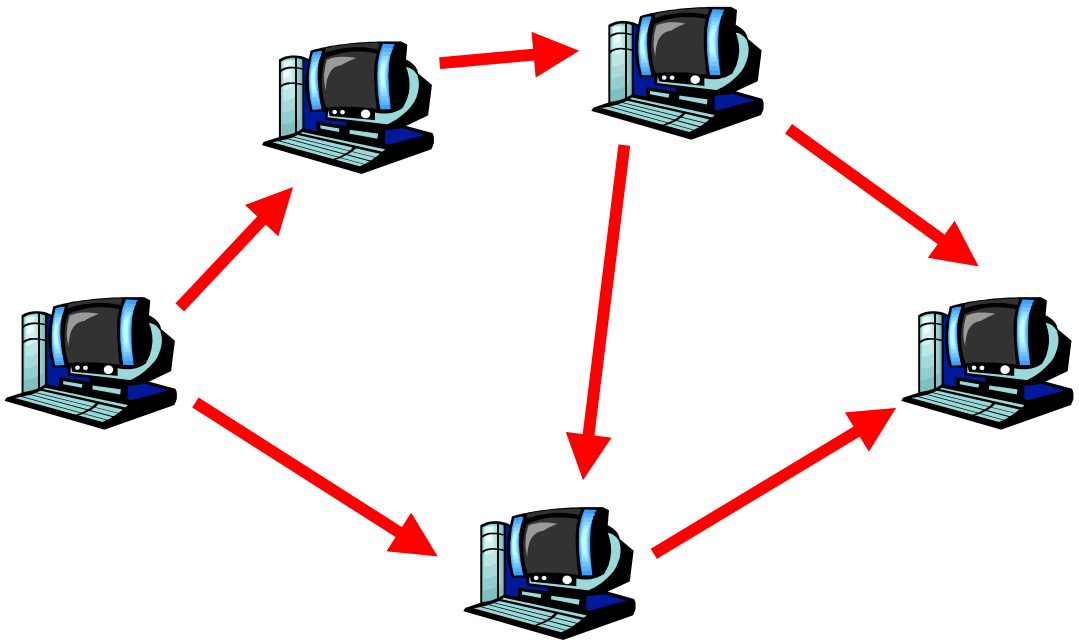
Gnutella: how it works

Searching by flooding:

- If you don't have the file you want, query 7 of your partners.
- If they don't have it, they contact 7 of their partners, for a maximum hop count of 10.
- Requests are flooded, but there is no tree structure.
- No looping but packets may be received twice.
- Reverse path forwarding(?)

Note: Play gnutella animation at:
http://www.limewire.com/index.jsp/p2p

Flooding in Gnutella: loop prevention



Seen already list: "A"

Gnutella message format

- Message ID: 16 bytes (yes bytes)
- FunctionID: 1 byte indicating
  - 00 ping: used to probe gnutella network for hosts
  - 01 pong: used to reply to ping, return # files shared
  - 80 query: search string, and desired minimum bandwidth
  - 81: query hit: indicating matches to 80:query, my IP address/port, available bandwidth
- RemainingTTL: decremented at each peer to prevent TTL-scoped flooding
- HopsTaken: number of peer visited so far by this message
- DataLength: length of data field

Gnutella: initial problems and fixes

- Freeloading: WWW sites offering search/retrieval from Gnutella network without providing file sharing or query routing.
  - Block file-serving to browser-based non-file-sharing users
- Prematurely terminated downloads:
  - long download times over modems
  - modem users run gnutella peer only briefly (Napster problem also!) or any users becomes overloaded
  - fix: peer can reply "I have it, but I am busy. Try again later"
  - late 2000: only 10% of downloads succeed
  - 2001: more than 25% downloads successful (is this success or failure?)

Gnutella: initial problems and fixes (more)

- 2000: avg size of reachable network ony 400-800 hosts. Why so smalll?
  - modem users: not enough bandwidth to provide search routing capabilities: routing black holes
- Fix: create peer hierarchy based on capabilities
  - previously: all peers identical, most modem blackholes
  - connection preferencing:
    - favors routing to well-connected peers
    - favors reply to clients that themselves serve large number of files: prevent freeloading
  - Limewire gateway functions as Napster-like central server on behalf of other peers (for searching purposes)

Anonymous?

- Not anymore than it's scalable.
- The person you are getting the file from knows who you are. That's not anonymous.

- Other protocols exist where the owner of the files doesn't know the requester.

- Peer-to-peer anonymity exists.
- See Eternity and, next, Freenet!

Gnutella Discussion:

- Architectural lessons learned?
- Do Gnutella's goals seem familiar? Does it work better than say squid or summary cache? Or multicast with carousel?
- anonymity and security?
- Other?
- Good source for technical info/open questions:

  http://www.limewire.com/index.jsp/tech_papers

Lecture 2 – (Query) Routing

- How can we route queries (or anything, packets, calls) better?…
- …While retaining the p2p advtanges (alleged)

3. Overlays

- Next, we need to look at overlays in general, and more specically, at
- Routing…
- RON is a good example…

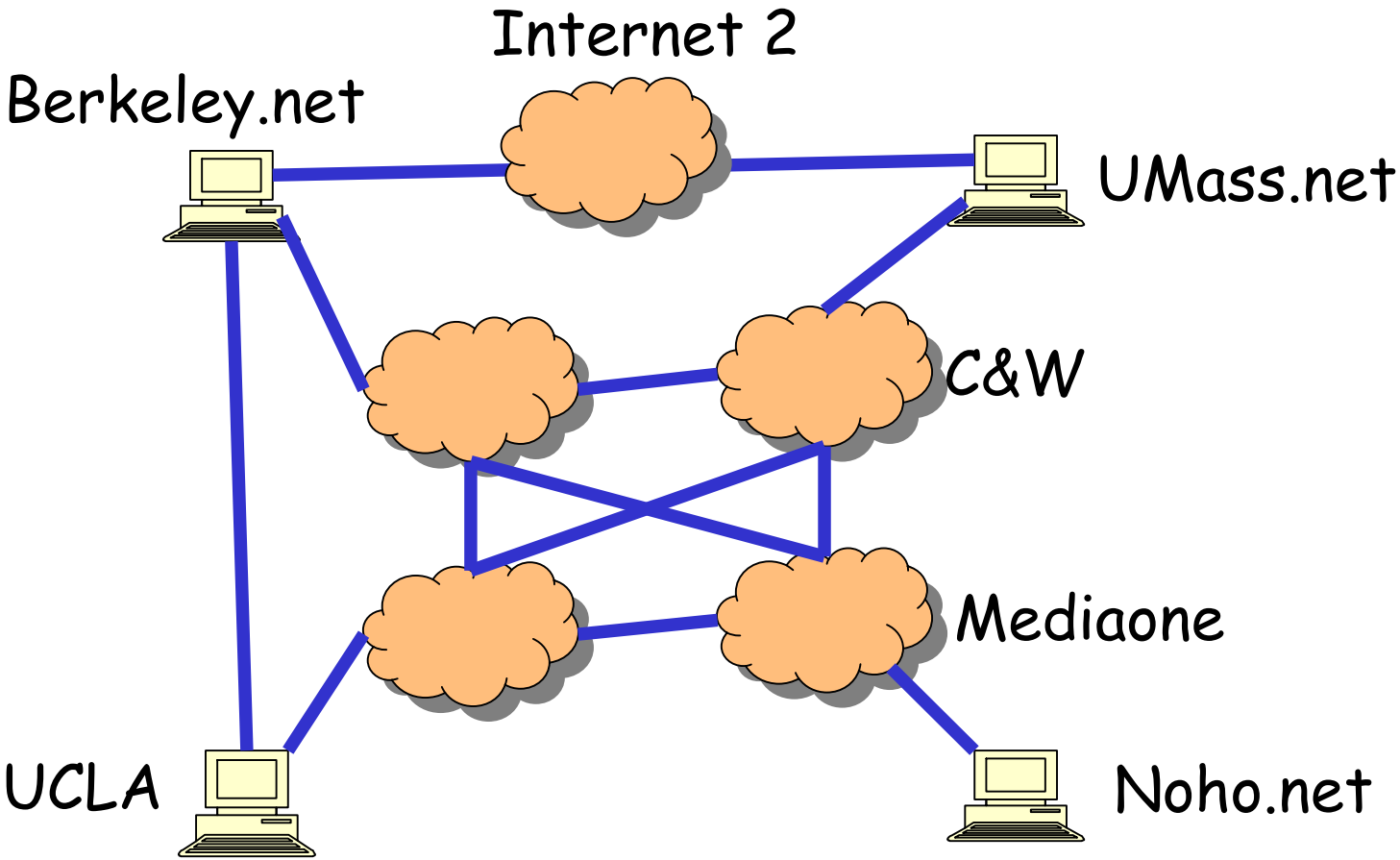
Resilient Overlay Networks

Overlay network:

- applications, running at various sites
- create "logical" links (e.g., TCP or UDP connections) pairwise between each other
- each logical link: multiple physical links, routing defined by native Internet routing
- let's look at an example, taken from:
  - D. Anderson, H. Balakrishnan, F. Kaashoek, R. Morris, "The case for reslient overlay networks," Proc. HotOS VIII, May 2001, http://nms.lcs.mit.edu/papers/ron-hotos2001.html.

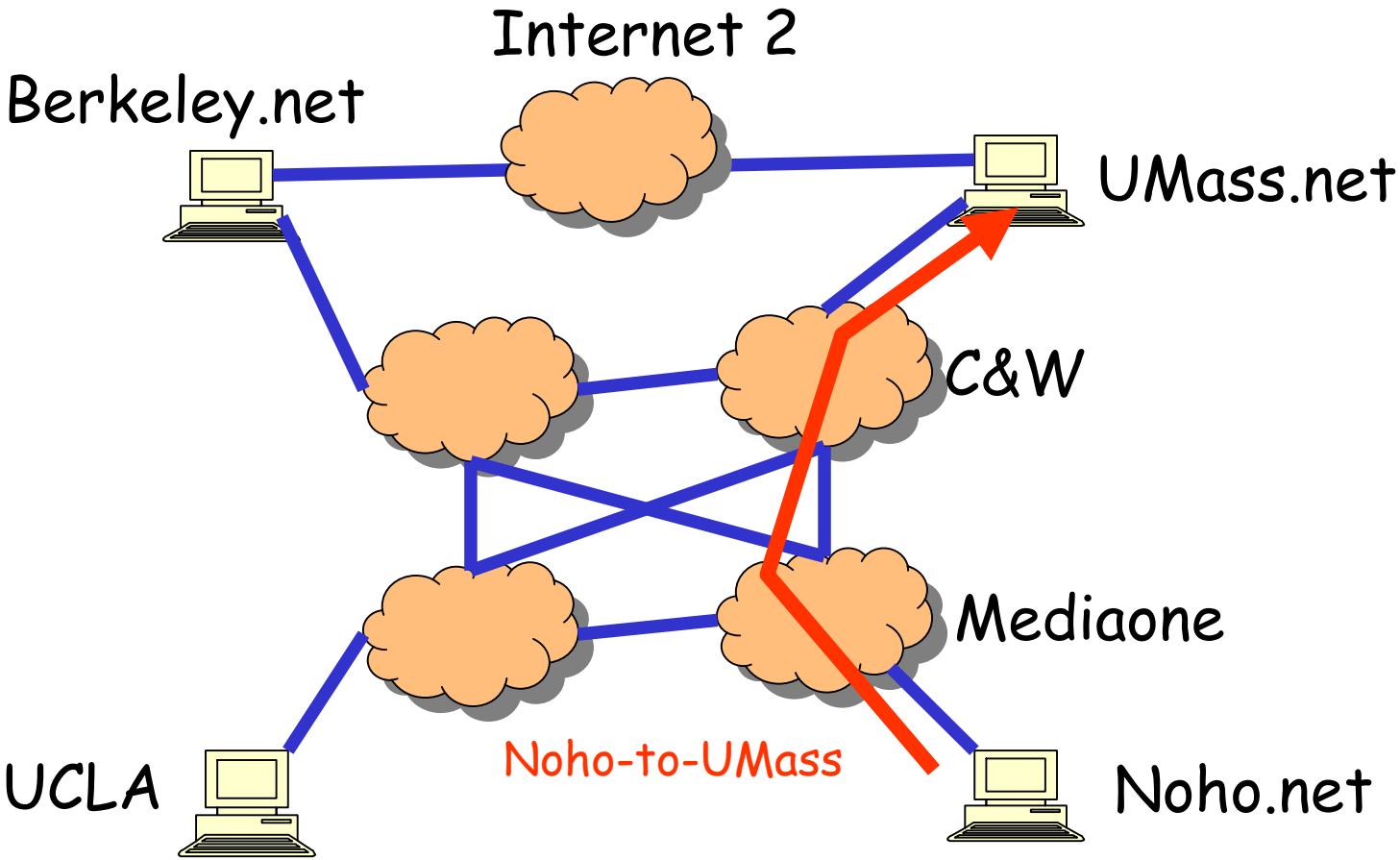
Internet Routing

- BGP defines routes between stub networks



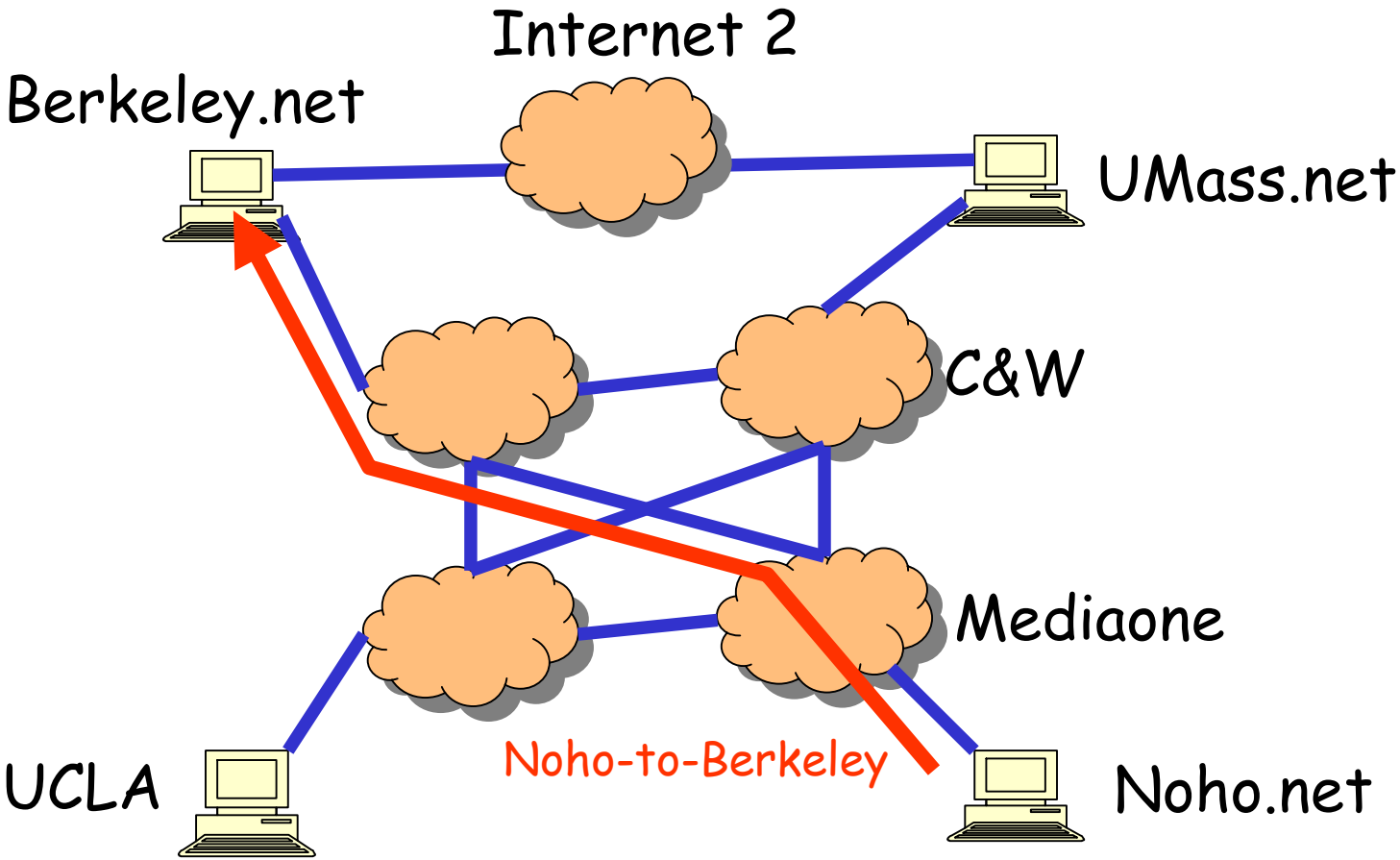
Internet Routing

- BGP defines routes between stub networks

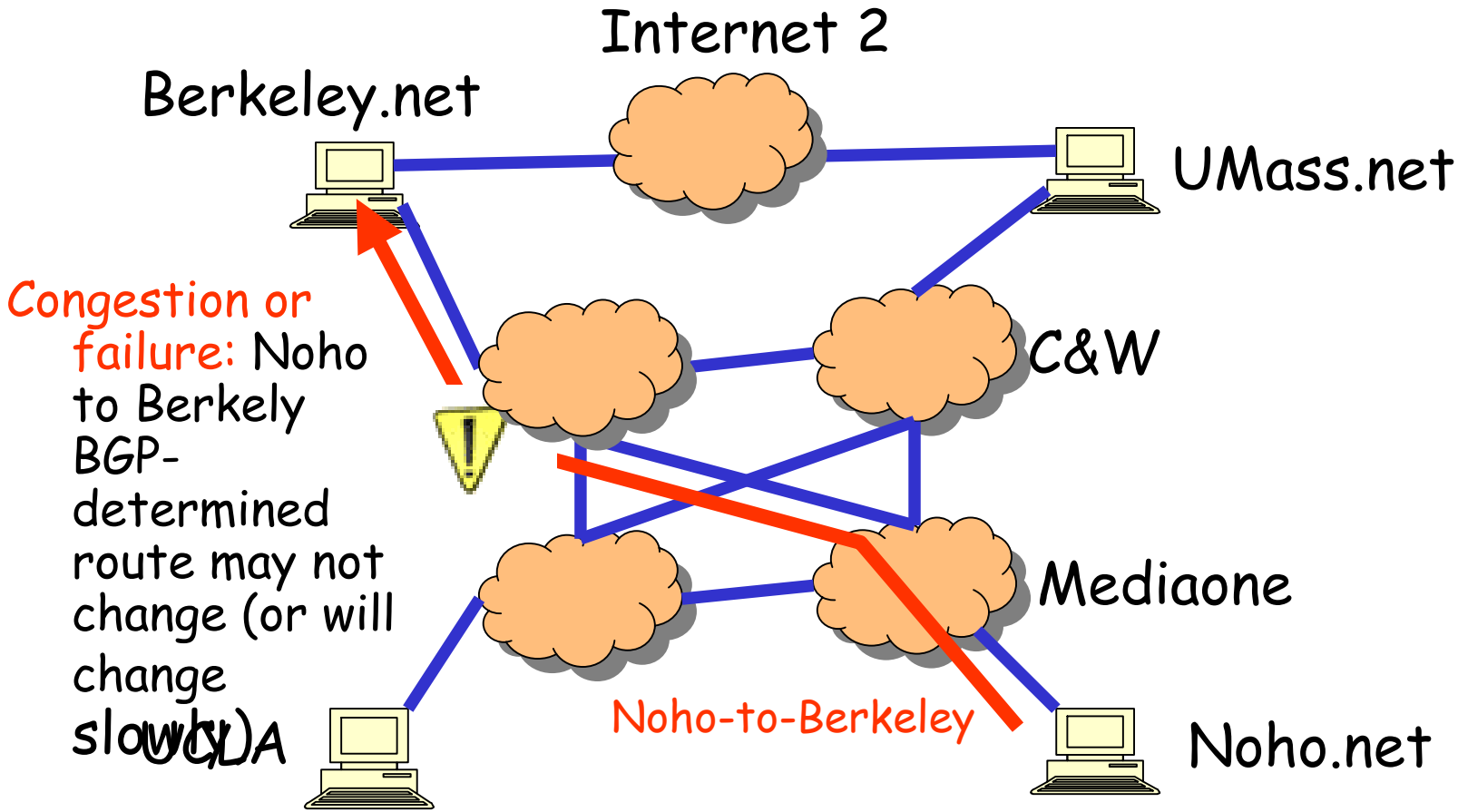


Internet Routing

- BGP defines routes between stub networks



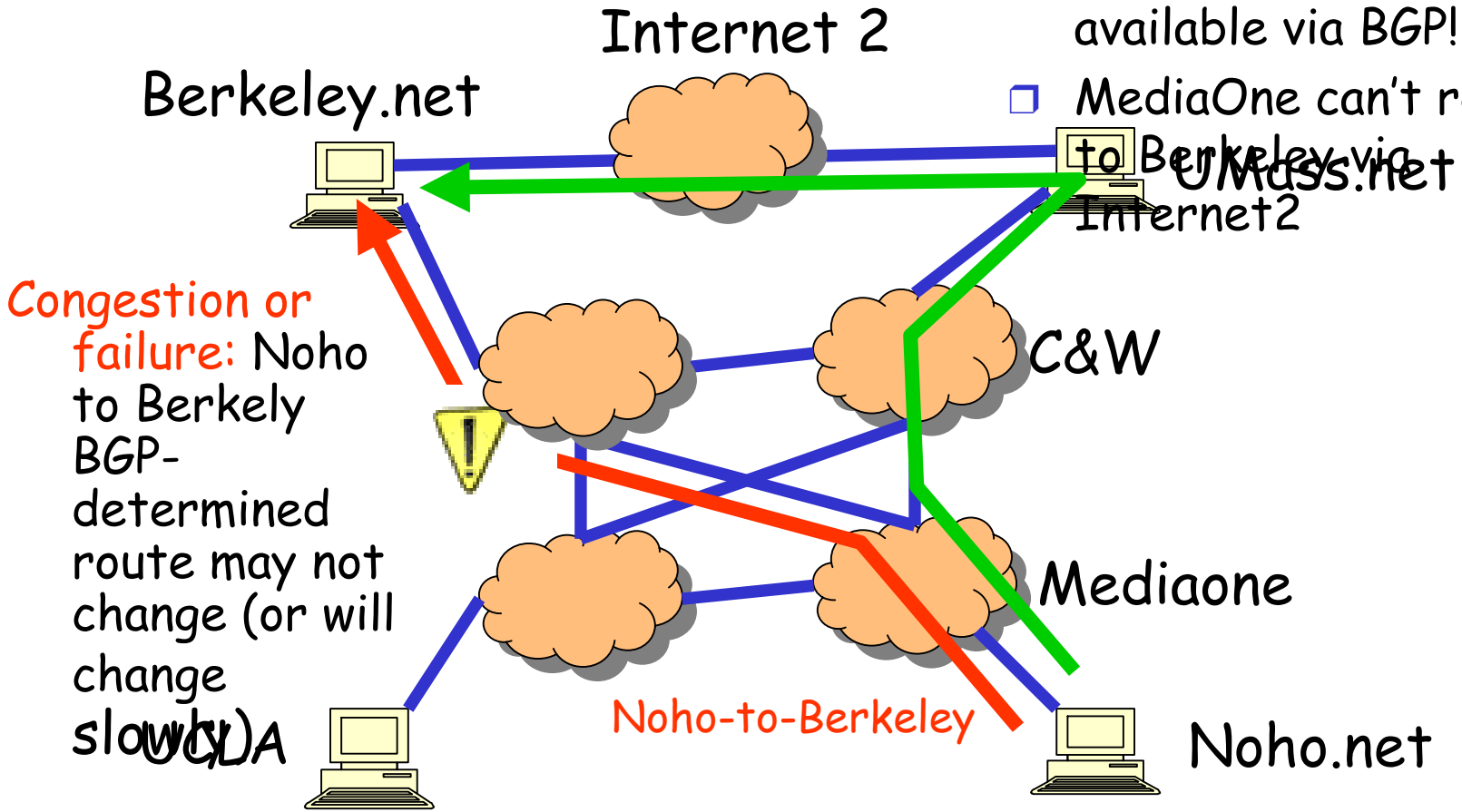
Internet Routing



Internet Routing

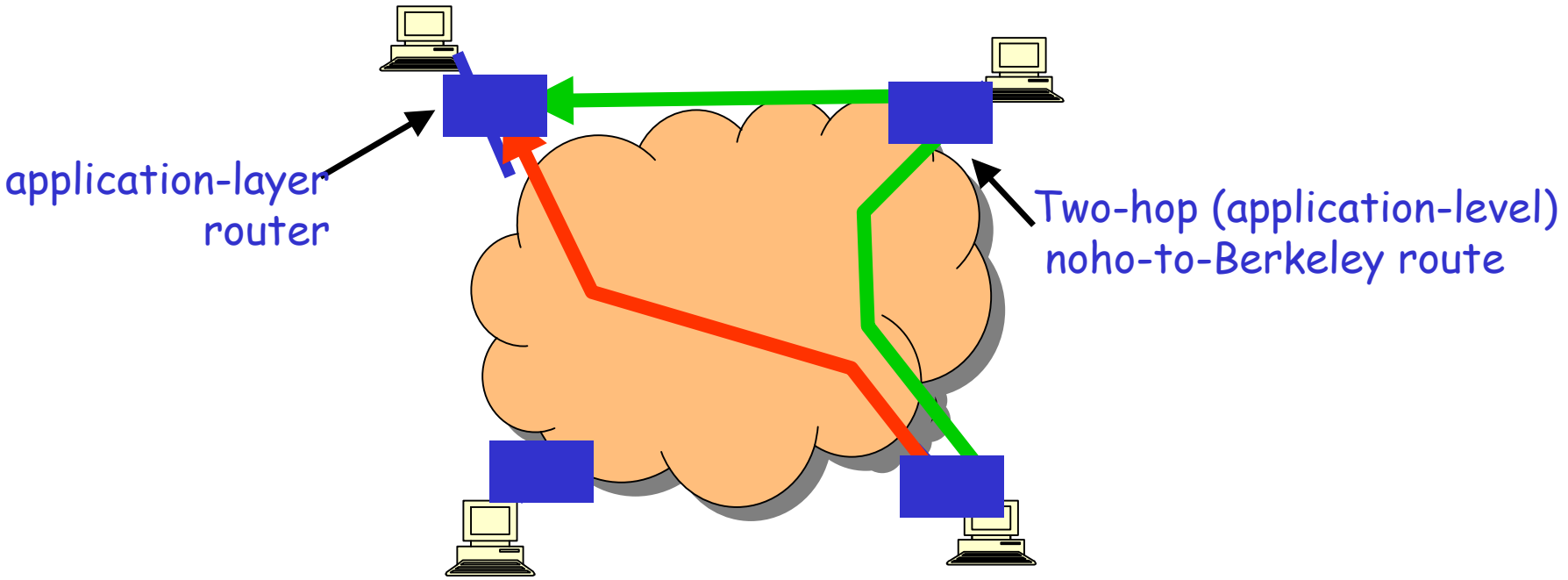
Noho to UMass to Berkeley

- route not visible or available via BGP!
- MediaOne can't route to Berkeley via Internet2



RON: Resilient Overlay Networks

Premise: by building application overlay network, can increase performance, reliability of routing



RON Experiments

- Measure loss, latency, and throughput with and without RON
- 13 hosts in the US and Europe
- 3 days of measurements from data collected in March 2001
- 30-minute average loss rates
  - A 30 minute outage is very serious!
- Note: Experiments done with "No-Internet2-for-commercial-use" policy

An order-of-magnitude fewer failures

30-minute average loss rates

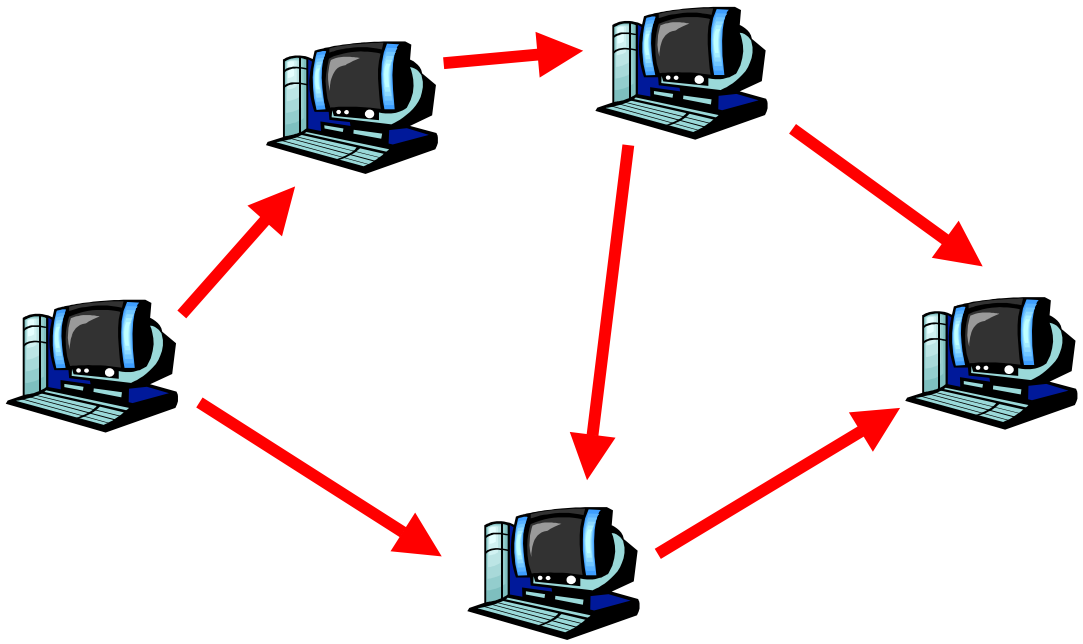
Loss Rate	RON Better	No Change	RON Worse
10%	479	57	47
20%	127	4	15
30%	32	0	0
50%	20	0	0
80%	14	0	0
100%	10	0	0

6,825 “path hours” represented here
12 “path hours” of essentially complete outage
76 “path hours” of TCP outage
RON routed around all of these!
One indirection hop provides almost all the benefit!

RON Research Issues

- How to design overlay networks?
  - Measurement and self-configuration
  - Understanding performance of underlying net.
  - Fast fail-over.
  - Sophisticated metrics.
  - application-sensitive (e.g., delay versus throughput) path selection.
- Effect of RON on underlying network
  - If everyone does RON, are we better off?

4. Freenet:- Small Worlds



P2P Application

- Centralized model
  - e.g. Napster
  - global index held by central authority
  - direct contact between requestors and providers
- Decentralized model
  - e.g. Freenet, Gnutella, Chord
  - no global index – local knowledge only (approximate answers)
  - contact mediated by chain of intermediaries

Freenet history

- Final Year project Ian Clarke , Edinburgh University, Scotland, June, 1999
- Sourceforge Project, most active
- V.0.1 (released March 2000)
- Latest version(Sept, 2001): 0.4

What is Freenet and Why?

- Distributed, Peer to Peer, file sharing system
- Completely anonymous, for producers or consumers of information
- Resistance to attempts by third parties to deny access to information

Freenet: How it works

- Data structure
- Key Management
- Problems
  - How can one node know about others
  - How can it get data from remote nodes
  - How to add new nodes to Freenet
  - How does freenet mangage its data
- Protocol Details
  - Header information

Data structure

- Routing Table
  - Pair: node address: ip, tcp; corresponding key value
- Data Store
  - Requirement:
    - rapidly find the document given a certain key
    - rapidly find the closest key to a given key
    - keep track the popularity of documents and know which document to delete when under pressure

Key Management(1)

- A way to locate a document anywhere
- Keys are used to form a URI
- Two similar keys don't mean the subjects of the file are similar!
- Keyword-signed Key*(KSK)*
  - *Based on a short descriptive string, usually a set of keywords that can describe the document*
  - *Example: University/umass/cs/hzhang*
  - *Uniquely identify a document*
  - *Potential problem – global namespace*

Key Management (2)

- Signed-subspace Key*(SSK)*
  - *Add sender information to avoid namespace conflict*
  - *Private key to sign/ public key to varify*
- Content-hash Key*(CHK)*
  - *Message digest algorithm, Basically a hash of the document*

Freenet: Routing Algorithm: search or insert




Darling,
Tell me
the truth!

Believe me, I
don't have it.

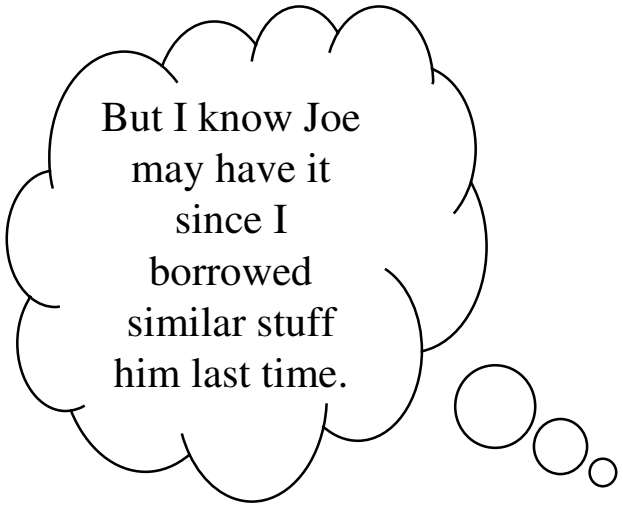
Freenet: Routing Algorithm: search or insert

B

C

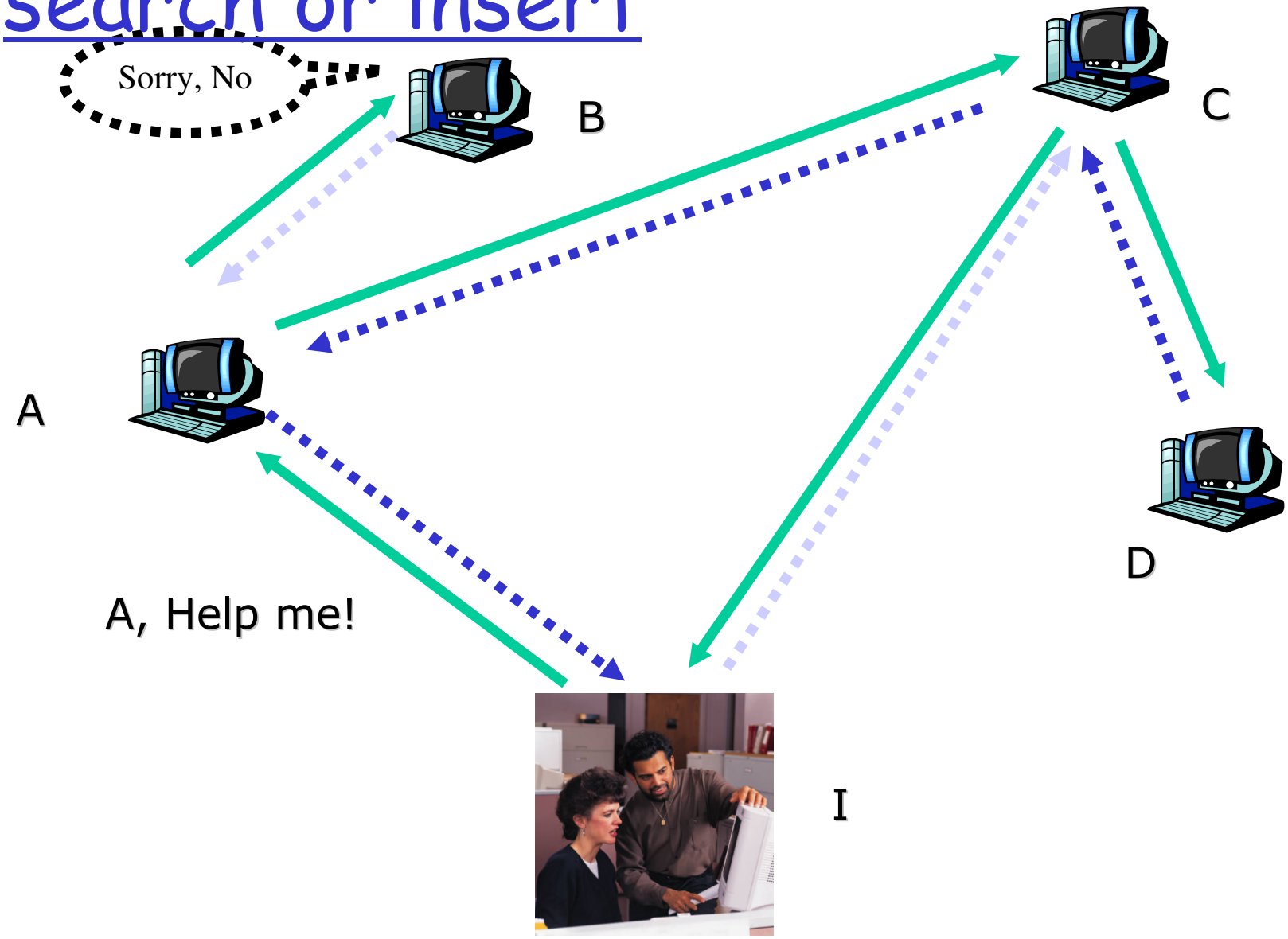
A

D



I

Freenet: Routing Algorithm: search or insert

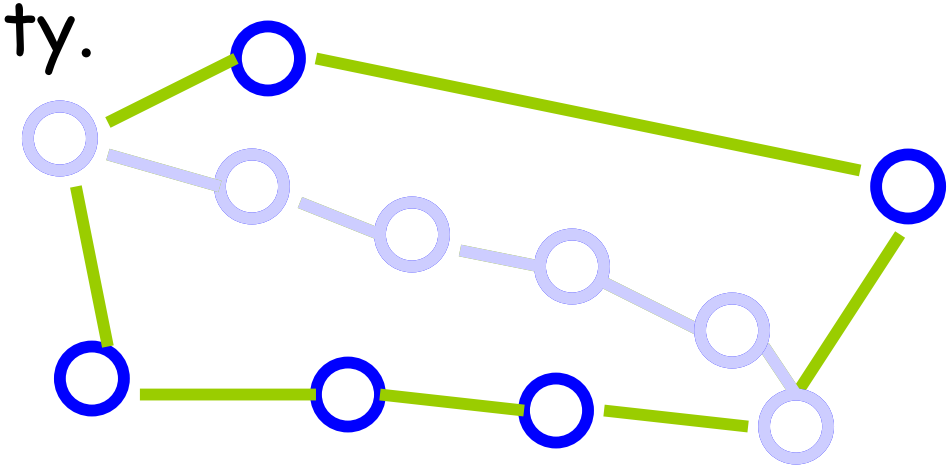


Strength of routing algorithm(1)

- Replication of Data Clustering (1)

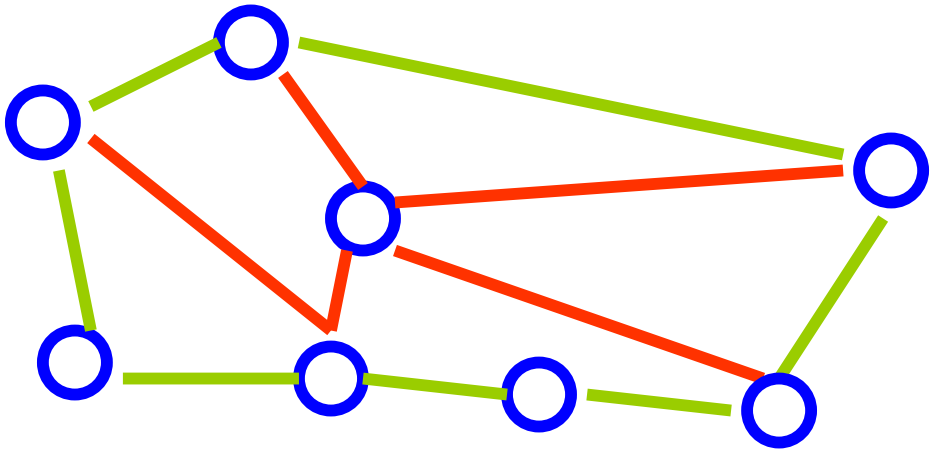
  (Note: Not subject-clustering but key-clustering!)

- Reasonable Redundancy: improve data availability.



Strength of routing algorithm(2)

- New Entry in the Routing Table: the graph will be more and more connected. --- Node discovery



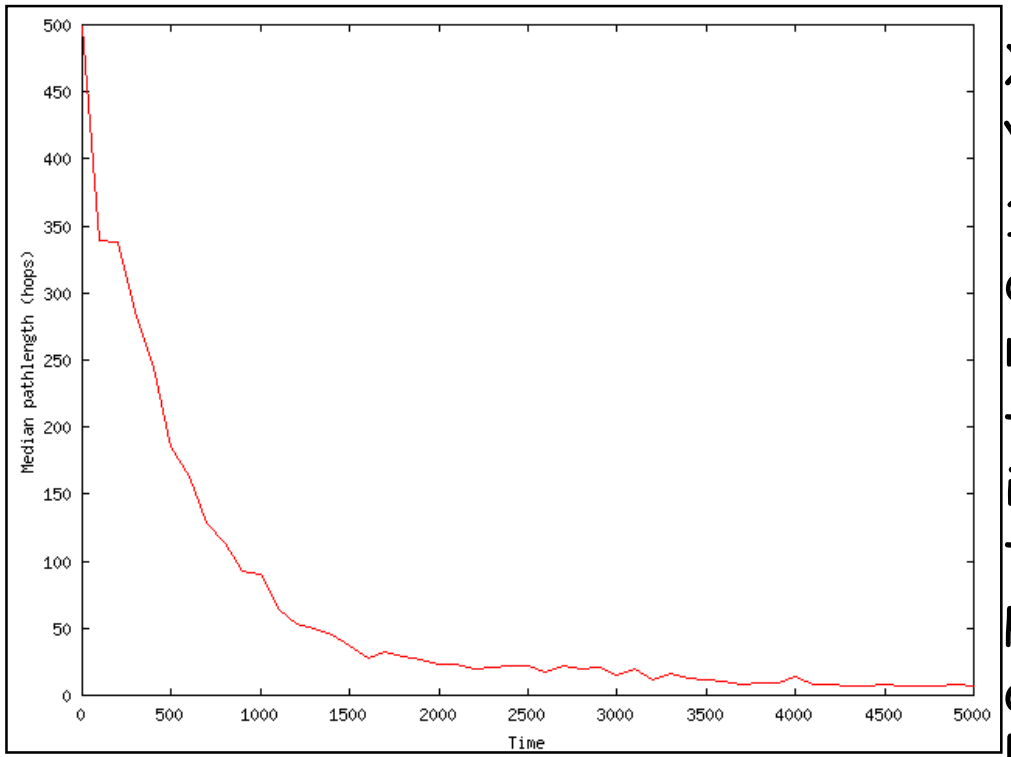
Protocol Details

- Header information
  - DataReply
    UniqueID=C24300FB7BEA06E3
    Depth=a
    * HopsToLive=2c
    Source=tcp/127.0.0.1:2386
    DataLength=131
    Data 'Twas brillig, and the slithy toves Did gyre and gimble in the wabe: All mimsy were the borogoves And the mome raths outgrabe

Some security and authentication issues

- How to ensure anonymity:
  - Nodes can lie randomly about the requests and claim to be the origin or the destination of a request
  - Hop-To-Live values are fuzzy
  - Then it's impossible to trace back a document to its original node
  - Similarly, it's impossible to discover which node inserted a given document.

Network convergence



X-axis: time

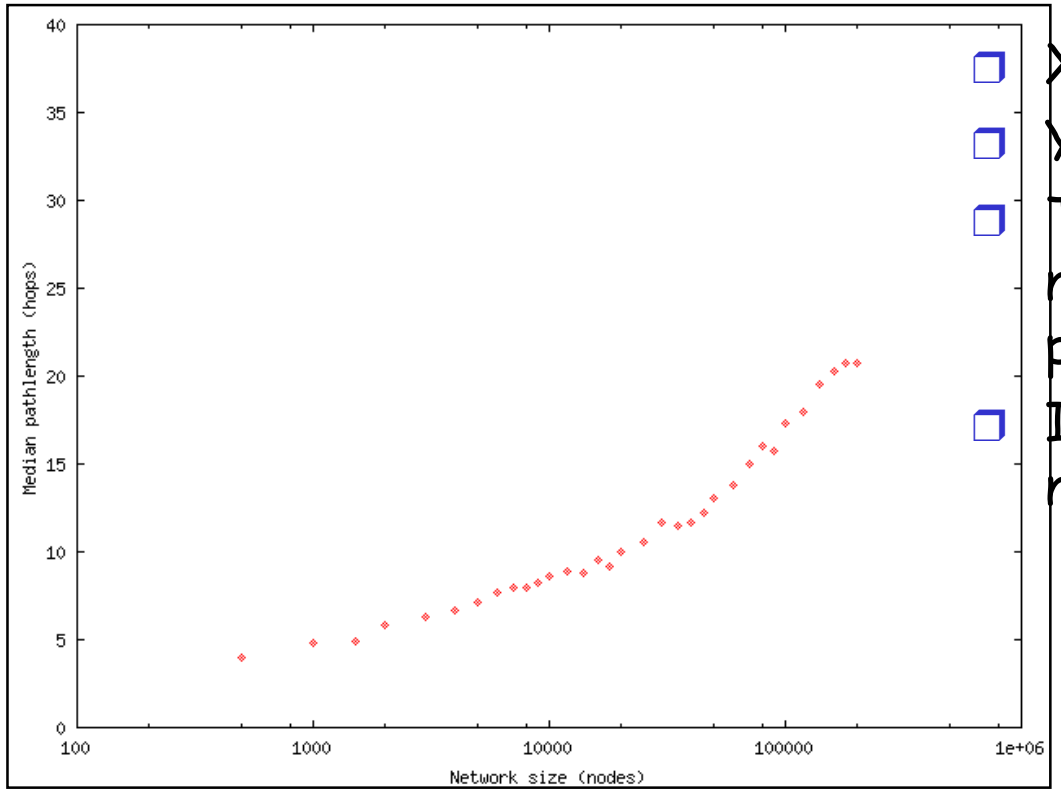
Y-axis: # of pathlength

1000 Nodes, 50 items datastore, 250 entries routing table

the routing tables were initialized to ring-lattice topology

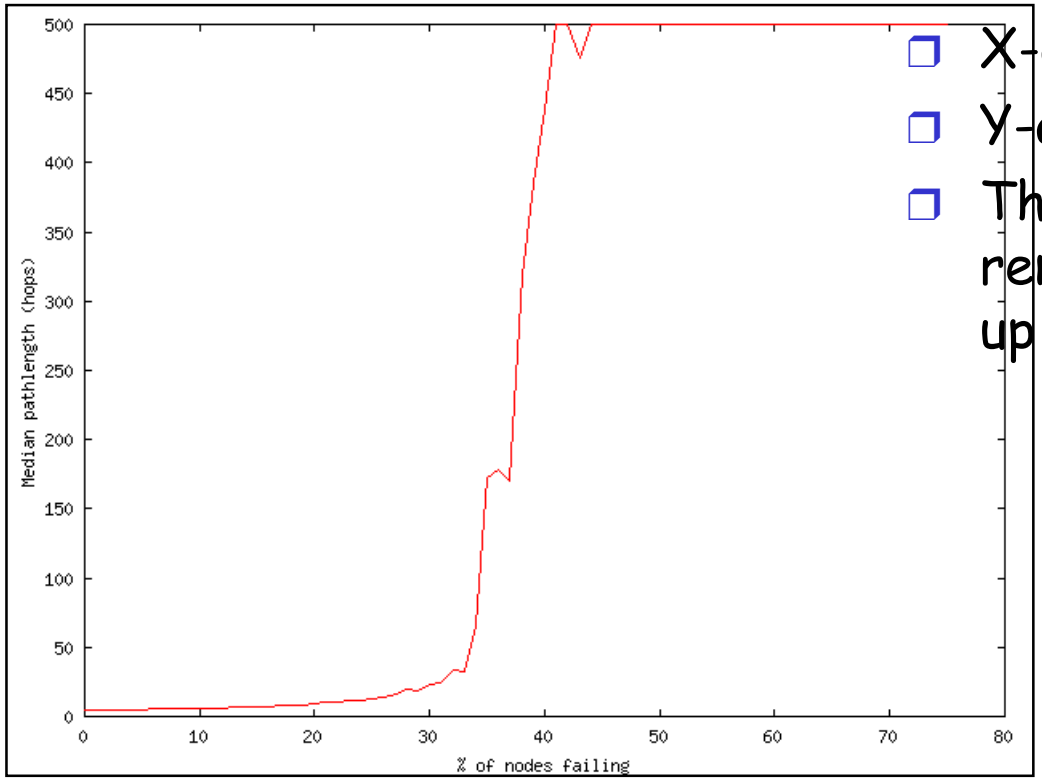
Pathlength: the number of hops actually taken before finding the data.

Scalability



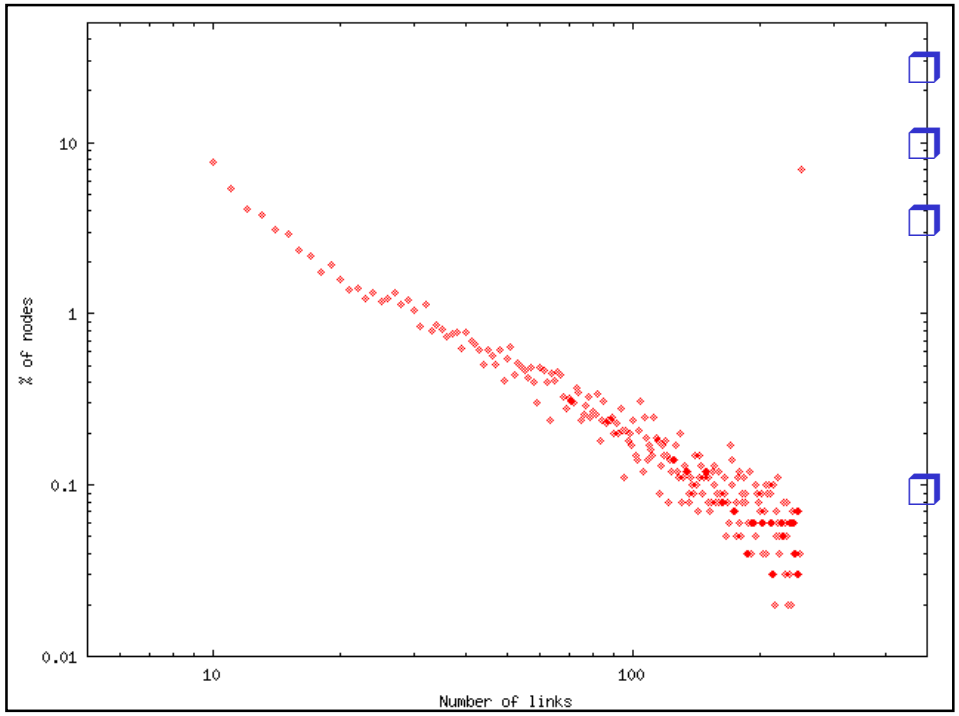
- X-axis: # of nodes
- Y-axis: # of pathlength
- The relation between network size and average pathlenth.
- Initially, 20 nodes. Add nodes regularly.

Fault Tolerance



- X-axis: # of nodes failing
- Y-axis: # of pathlength
- The median pathlength remains below 20 even when up to 30% nodes fails.

Small world Model



- X-axis: # of nodes failing
- Y-axis: # of pathlength
- Most of nodes have only few connections while a small number of news have large set of connections.
- The authors claim it follows power law.

So far, it's really a good model

- Keep anonymity
- Distributed model; data available
- Converge fast
- Adaptive

Is it Perfect?

- How long will it take to search or insert?
  - Trade off between anonymity and searching efforts: Chord vs Freenet
  - Can we come up a better algorithm? A good try: "Search in Power-Law Networks"
- Have no idea about if search fails due to no such document or just didn't find it.
- File lifetime. Freenet doesn't guarantee a document you submit today will exist tomorrow!!

Question??

- Anonymity? Security?
- Better search algorithm? Power law?
- …

5. Publius: A robust, tamper-evident, censorship-resistant web publishing system – see also Eternity…

Marc Waldman **Aviel Rubin** **Lorrie Faith Cranor**

Outline

- Design Goals
- Kinds of Anonymity
- Publius Features
- Publius Limitations and Threats
- Questions

Design Goals

- Censorship resistant
  - Difficult for a third party to modify or delete content
- Tamper evident
  - Unauthorized changes should be detectable
- Source anonymous
  - No way to tell who published the content
- Updateable
  - Changes to or deletion of content should be possible for publishers

Design Goals

- Deniable
  - Involved third parties should be able to deny knowledge of what is published
- Fault Tolerant
  - System remains functional, even if some third parties are faulty or malicious
- Persistent
  - No expiration date on published materials

Web Anonymity

- Connection Based
  - Hides the identity of the individual requesting a page Examples:
    - Anonymizing proxies, such as The Anonymizer or Proxymate
    - Proxies utilizing Onion Routing, such as Freedom
    - Crowds, where users in the Crowd probabilistically route or retrieve for other users in the Crowd

Web Anonymity

- Author Based
  - Hides the location or author of a particular document Examples:
    - Rewebber, which proxies requests for encrypted URLs
    - The Eternity Service, which for a fee inserts a document into a random subset of servers, and guarantees its future existence
    - Freenet
  - Publius provides this sort of anonymity

Publius System Overview

- Publishers
  - Post Publius content to the web
- Servers
  - A static set which host random-looking content
- Retrievers
  - Browse Publius content on web

Publius System Overview

- Publish
  - A publisher posts content across multiple servers in a source anonymous fashion
- Retrieve
  - A retriever gets content from multiple servers
- Delete
  - The original publisher of a document removes it from the Publius servers
- Update
  - The original publisher modifies a document

Publius Publishing

- Alice generates a random symmetric key K
- She encrypts message M with key K, producing $\{M\}_K$
- She splits K into n shares, using Shamir secret sharing, such that any k can reproduce K
- Each share is uniquely named:

$$name_i = wrap(H(M \cdot share_i))$$

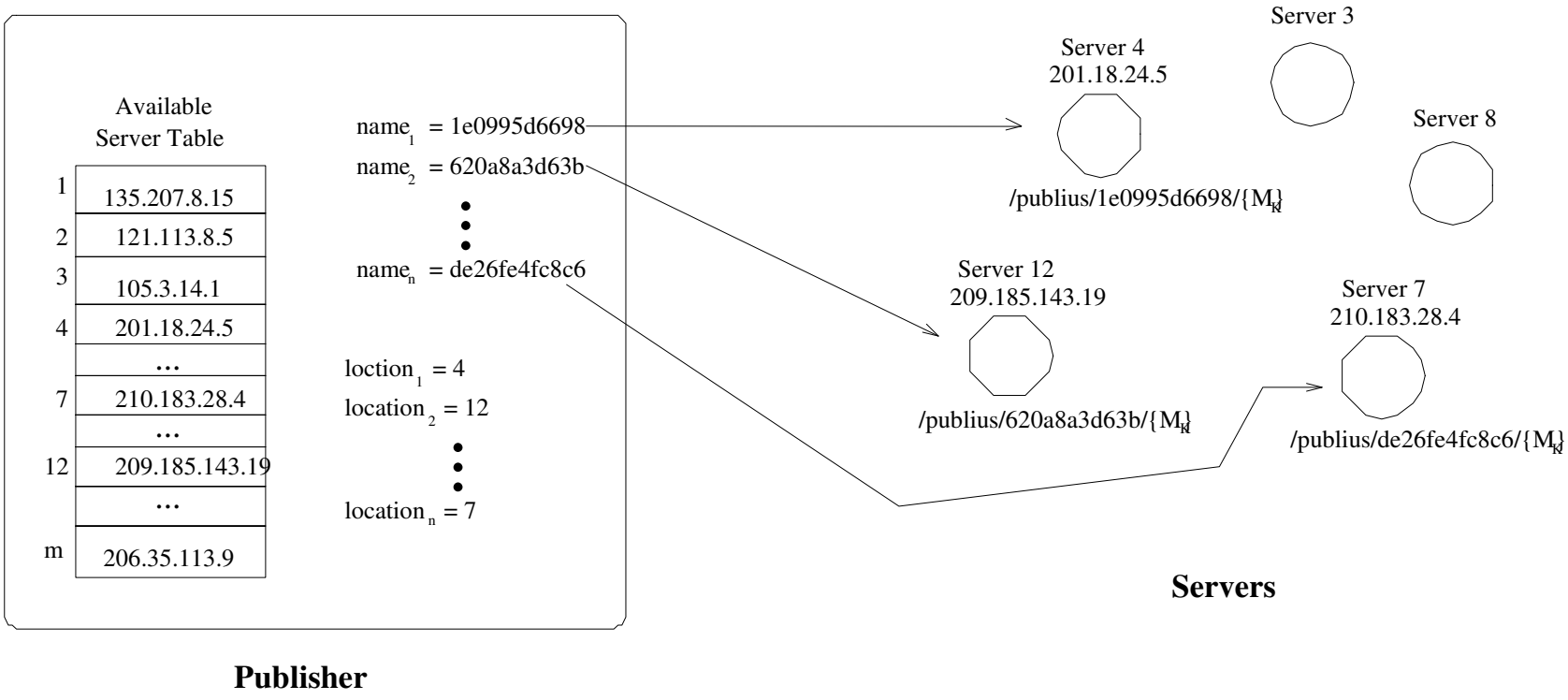
Publius Publishing

- A set of locations is chosen:

  $location_i = (name_i \text{ MOD } m) + 1$

- Each $location_i$ indexes into the list of m servers
- If $d >= k$ unique values are not obtained, start over
- Alice publishes $\{M\}_K$ and $share_i$ into a directory $name_i$ on the server at $location_i$
- A URL containing at least the d $name_i$ values is produced

Publius Publishing



Publius Retrieval

- Bob parses out each $name_i$ from URL, and for each, computes:

$$location_i = (name_i \ MOD \ m) + 1$$

- Bob chooses k of these, and retrieves the encrypted file $\{M\}_K$ and $share_i$ at each server
- Bob combines the shares to get K, and decrypts the file
- Bob verifies that each name value is correct:

$$name_i = wrap(H(M \cdot share_i))$$

Publius Delete

- Alice generates a password *PW* when publishing a file
- Alice includes *H(server_domain_name · PW)* in server directory when publishing
  - Note that each server has its own hash, to prevent a malicious server operator from deleting content on all servers
- Alice deletes by sending *H(server_domain_name · PW)* and $name_i$ to each of the *n* servers hosting content

Publius Update

- Idea: change content without changing original URL, as links to that URL may exist
- In addition to the file, the share, and the password, there may be an update file in the *name;* directory
- This update file will not exist if Alice has not updated the content

Publius Update

- To update, Alice specifies a new file, the original URL, the original password *PW*, and a new password
- First, the new content is published, and a new URL is generated
- Then, each of the *n* old files is deleted, and an update file, containing the new URL, is placed in each $name_i$ directory

Publius Update

- When Bob retrieves updated content, the server returns the update file instead
- Bob checks that all of the URLs are identical, then retrieves the content at the new URL

Linking Documents

- Simple case: file A links to file B
  - Solution: Publish B first, then rewrite URLs in A
- Harder: files C and D link to each other
  - Cannot use simple solution above
  - Alice publishes C and D in any order
  - She then rewrites the URLs in each file, and uses the Publius Update procedure on the new files

Other Features

- Entire directories can be published by exploiting the updateability of Publius
- Mechanism exists to encode MIME type into Publius content
- Publius URLs include option fields and other flags, the value of *k*, and other relevant values
  - Older broswers preclude URLs of length >255 characters
  - Once this limitation is removed, URLs can include server list, making this list non-static

Limitations and Threats

- Share deletion or corruption
  - If all *n* copies of a file, or *n*-*k*+1 copies of the shares, are deleted, then the file is unreadable
  - Increasing *n*, or decreasing *k*, makes this attack harder

Limitations and Threats

- Update file deletion or corruption 1
  - If there is no update file, malicious server operator Mallory could create one, pointing to bad content
  - This requires the assistance of at least *k* other server operator, and motivates a higher value of *k*
  - The Publius URL has several fields, among them a *no_update* flag, which will prevent this sort of attack

Limitations and Threats

- Update file deletion or corruption 2
  - If Publius content has already been updated, Mallory must corrupt update files on $n-k+1$ servers
  - Of course, if Mallory can do this, she can censor any document
  - Larger n and smaller k make this more difficult
- Deciding upon good values for n and k is difficult
  - No suggestions from Waldman et al.

Limitations and Threats

- Publius, like all internet services, is subject to DoS attacks
  - Flooding is less effective, as n-k+1 servers must be attacked
  - A malicious user could attempt to fill disk space on servers
    - Some mechanisms in place to prevent this

Limitations and Threats

- If the Publius content contains any identifying information, anonymity will be lost
- Publius does not provide any connection based anonymity
  - If you act as a publisher, you must anonymize your connections with the Publius servers

Questions

- How do you publish Publius URLs anonymously?
  - Freenet keys can be guessed at, but Publius URLs are entirely machine generated
  - The first person to publish a Publius URL must have some connection with the publisher of the content
  - If you have somewhere secure and anonymous to publish the Publius URLs, why do you need Publius?
    - One possible answer: censorship resistance
    - But server operators are then potentially liable

Questions

- How deniable is Publius?
  - Publius URLs are public
  - With minimal effort, a Publius server operator could determine the content being served

Questions

- How does Publius compare to Freenet?
  - Both provide publisher anonymity, deniability, and censorship resistance
  - Freenet provides anonymity for retrievers and servers, as well
    - Cost is high: data must be cached at many nodes
  - Publius provides persistence of data
    - Freenet does not
    - Can any p2p system provide persistence?

Questions

- Could Publius be made into a p2p service?
- Would it be appropriate to do so?

Lecture 3: Distributed Hash Tables

- Can we go from content to location in one go?
- Can we still retain locality?
- Can we keep any anonimity
- Look at Chord, Tapestry, CAN (pastry is similar… … …)
- Notice how networking people like silly names☺

6. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica, Robert Morris, David Karger,
M. Frans Kaashoek, Hari Balakrishnan

MIT and Berkeley

Now we see some CS in strength – Hash and Content based….for more scaleble (distributed) directory lookup

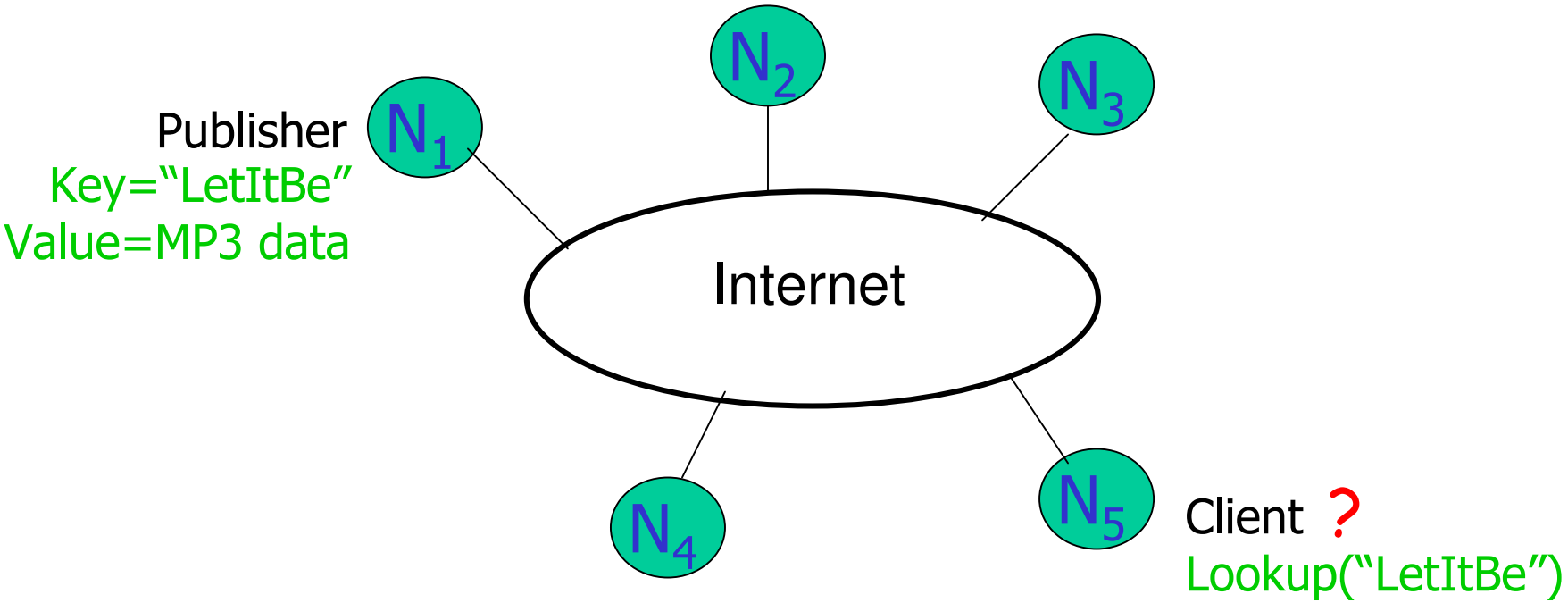
- presentation based on slides by Robert Morris (SIGCOMM'01)

Outline

- Motivation and background
- Consistency caching
- Chord
- Performance evaluation
- Conclusion and discussion

Motivation

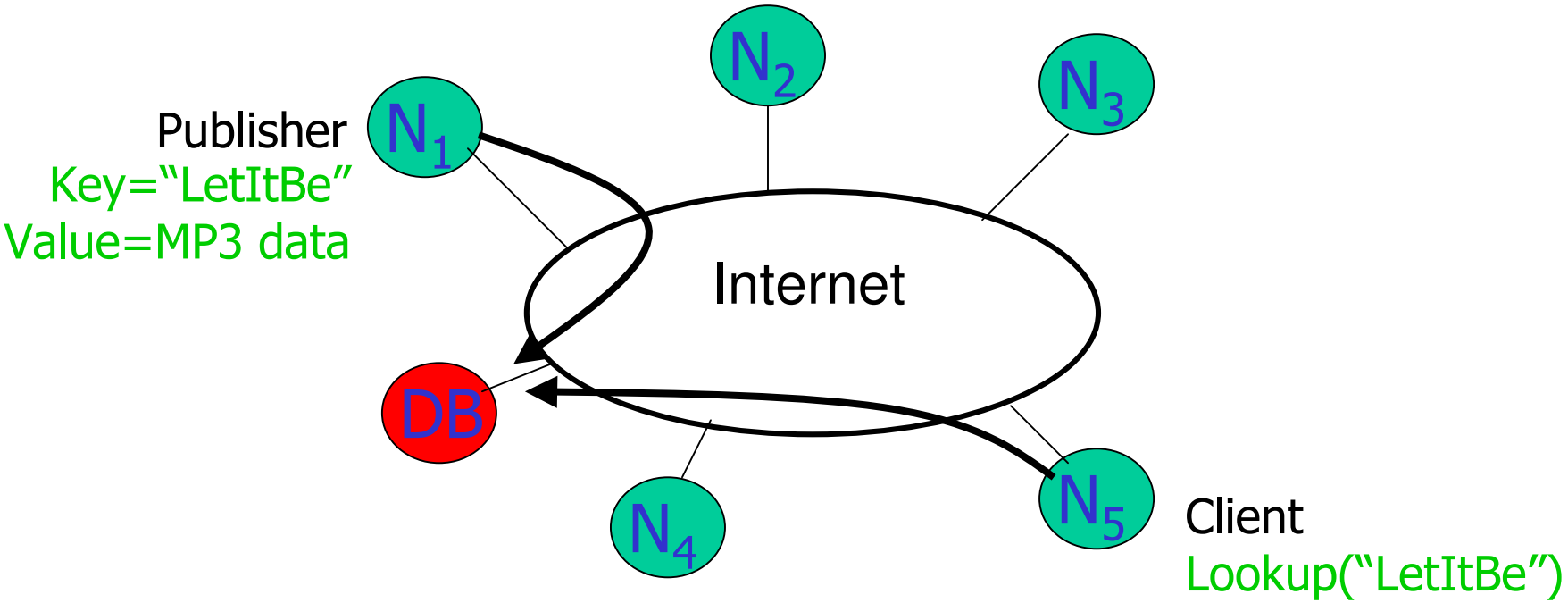
How to find data in a distributed file sharing system?



- Lookup is the key problem

Centralized Solution

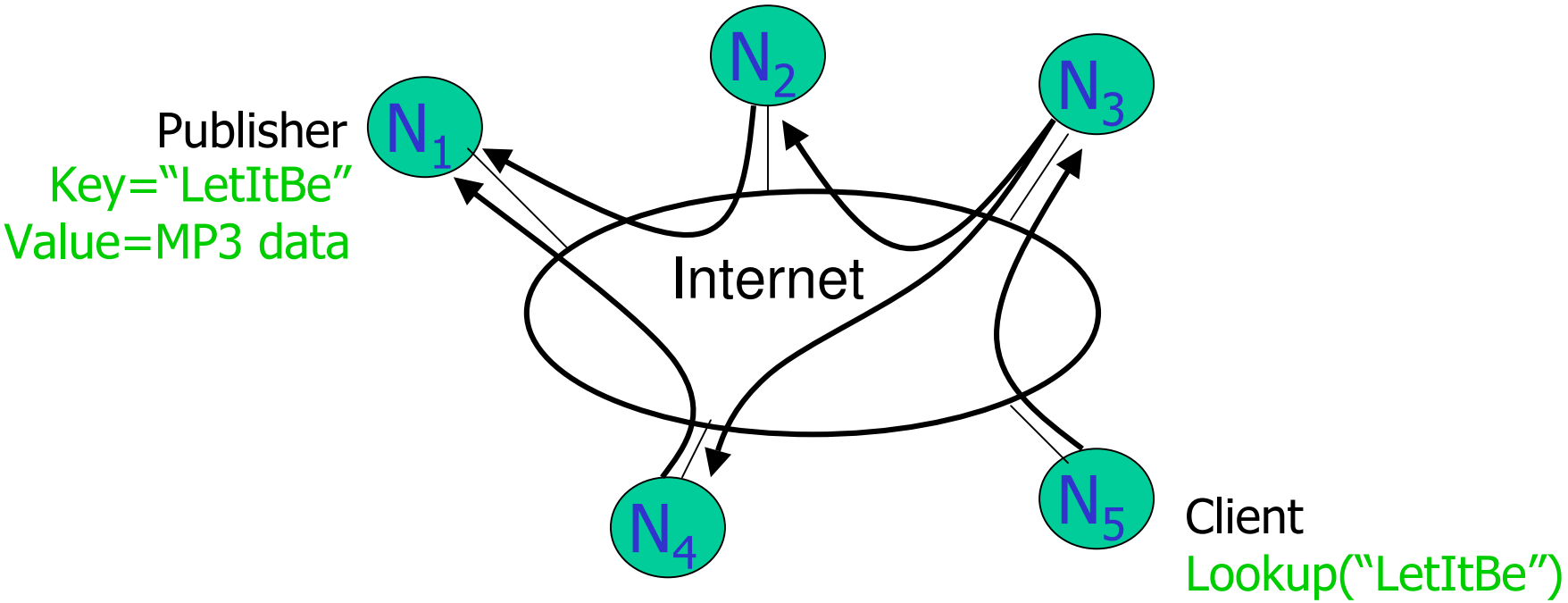
- Central server (Napster)



- Requires O(M) state
- Single point of failure

Distributed Solution (1)

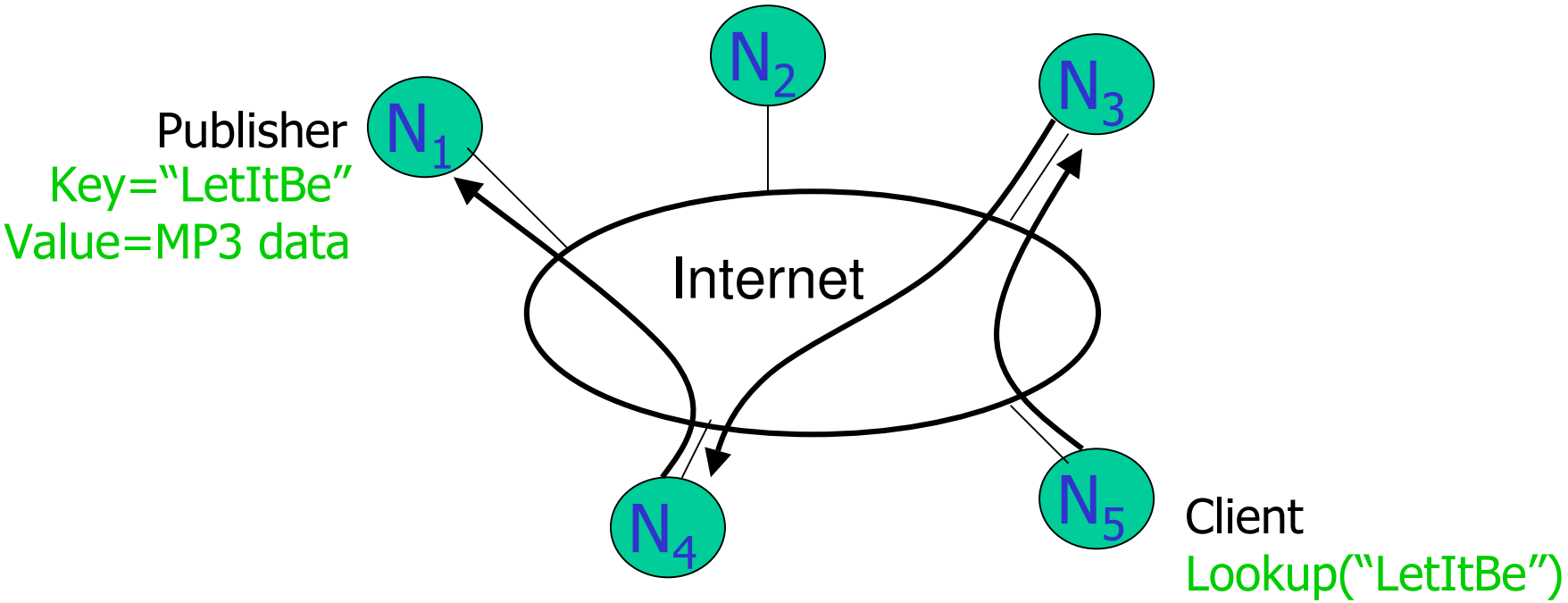
- Flooding (Gnutella, Morpheus, etc.)



- Worst case O(N) messages per lookup

Distributed Solution (2)

- Routed messages (Freenet, Tapestry, Chord, CAN, etc.)



- Only exact matches

Routing Challenges

- Define a useful key nearness metric
- Keep the hop count small
- Keep the routing tables "right size"
- Stay robust despite rapid changes in membership

Authors claim:

- Chord: emphasizes efficiency and simplicity

Chord Overview

- Provides peer-to-peer hash lookup service:
  - Lookup(key) → IP address
  - Chord does not store the data
- How does Chord locate a node?
- How does Chord maintain routing tables?
- How does Chord cope with changes in membership?

Chord properties

- Efficient: O(Log N) messages per lookup
  - N is the total number of servers
- Scalable: O(Log N) state per node
- Robust: survives massive changes in membership

- Proofs are in paper / tech report
  - Assuming no malicious participants

Chord IDs

- *m* bit identifier space for both keys and nodes
- Key identifier = SHA-1(key)

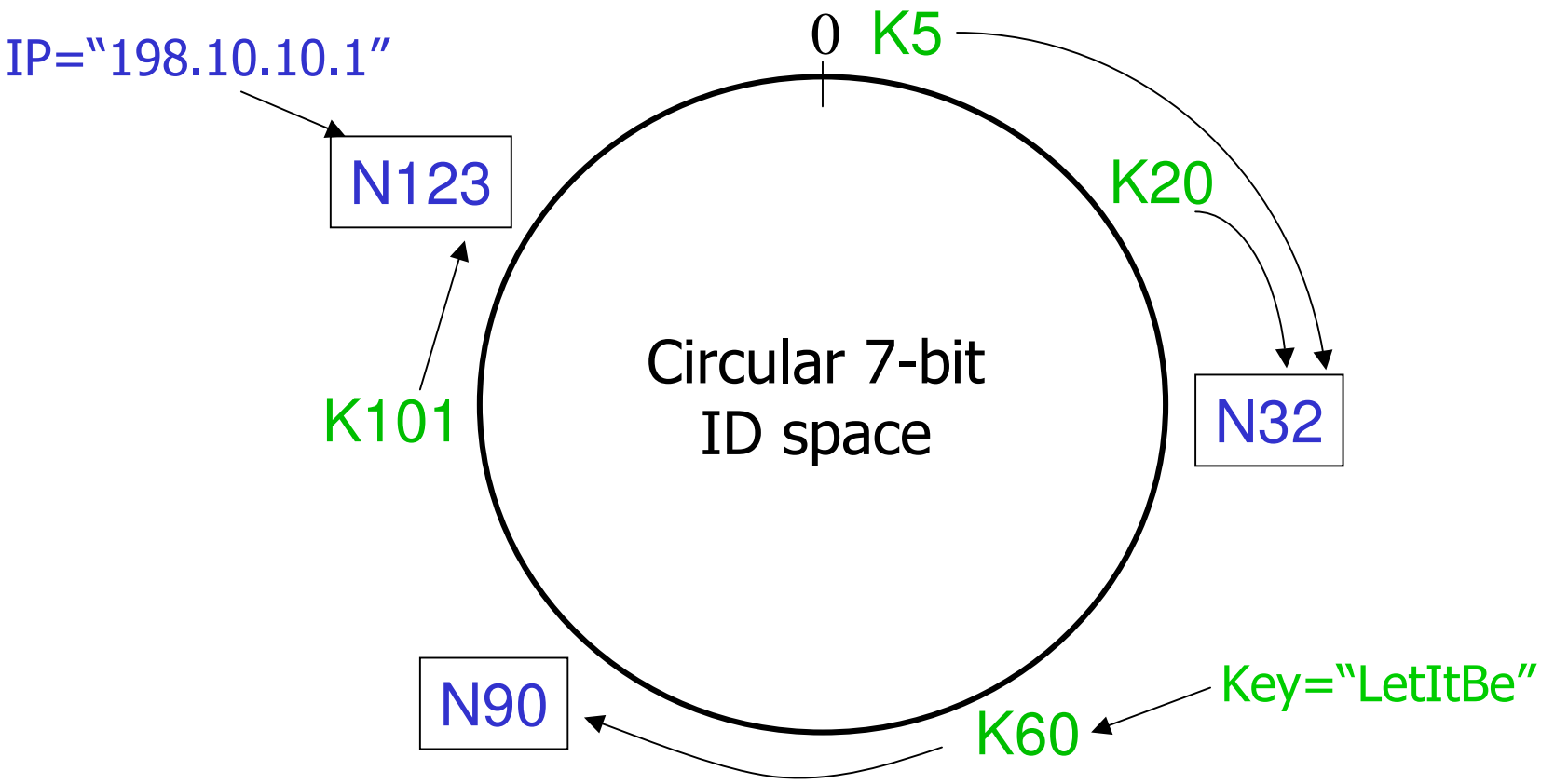
  Key="LetItBe" —SHA-1→ ID=60

- Node identifier = SHA-1(IP address)

  IP="198.10.10.1" —SHA-1→ ID=123

- Both are uniformly distributed
- How to map key IDs to node IDs?

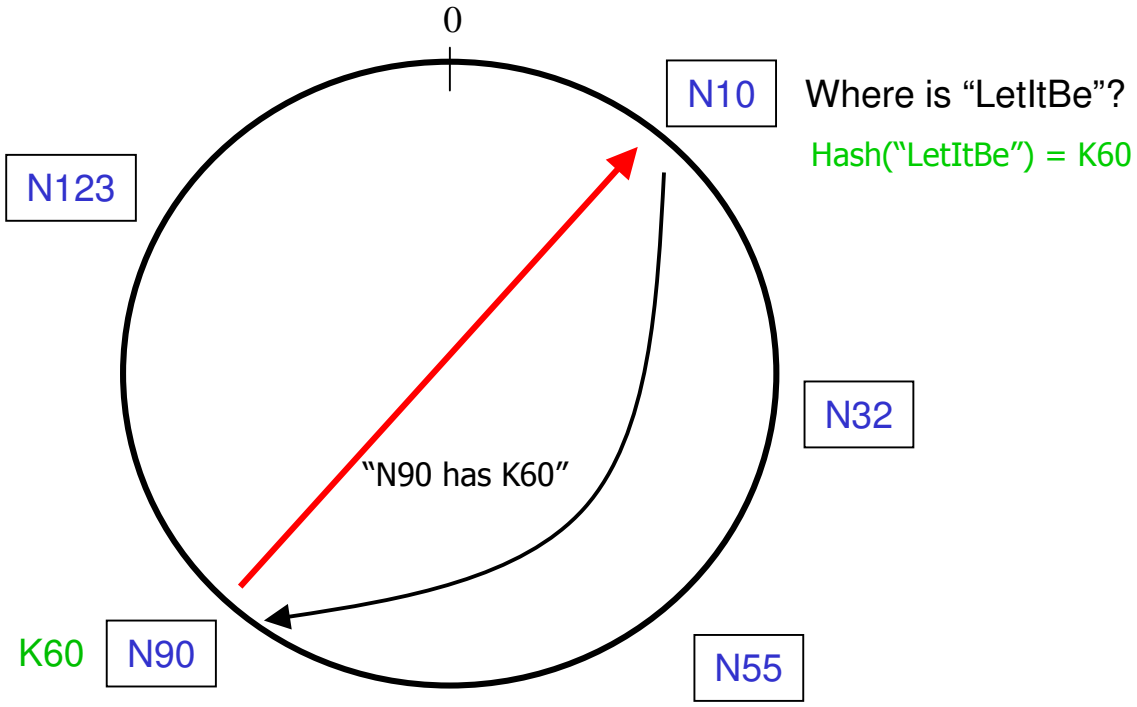
Consistent Hashing [Karger 97]



- A key is stored at its successor: node with next higher ID

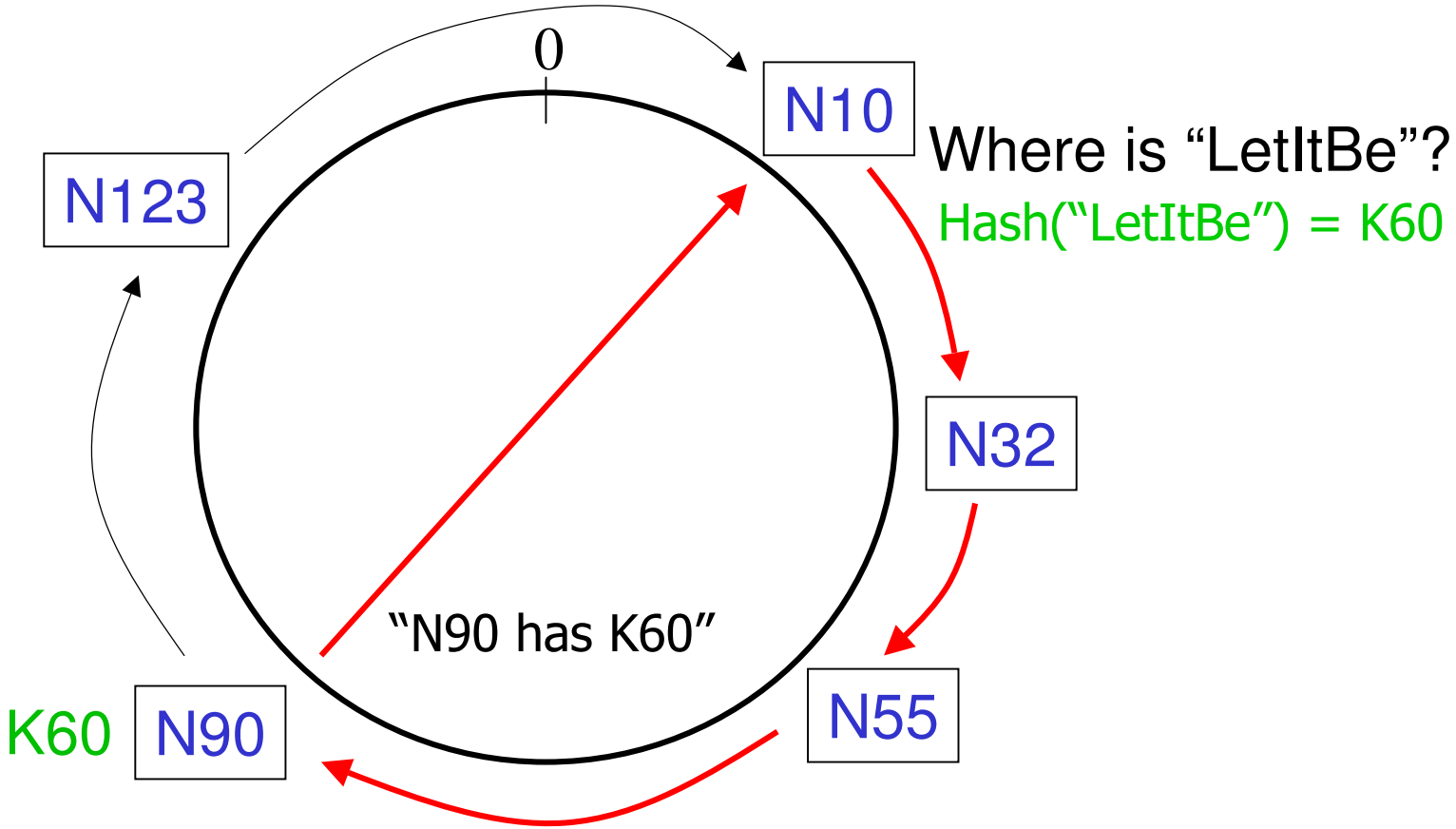
Consistent Hashing

- Every node knows of every other node
  - requires global information
- Routing tables are large O(N)
- Lookups are fast O(1)



Chord: Basic Lookup

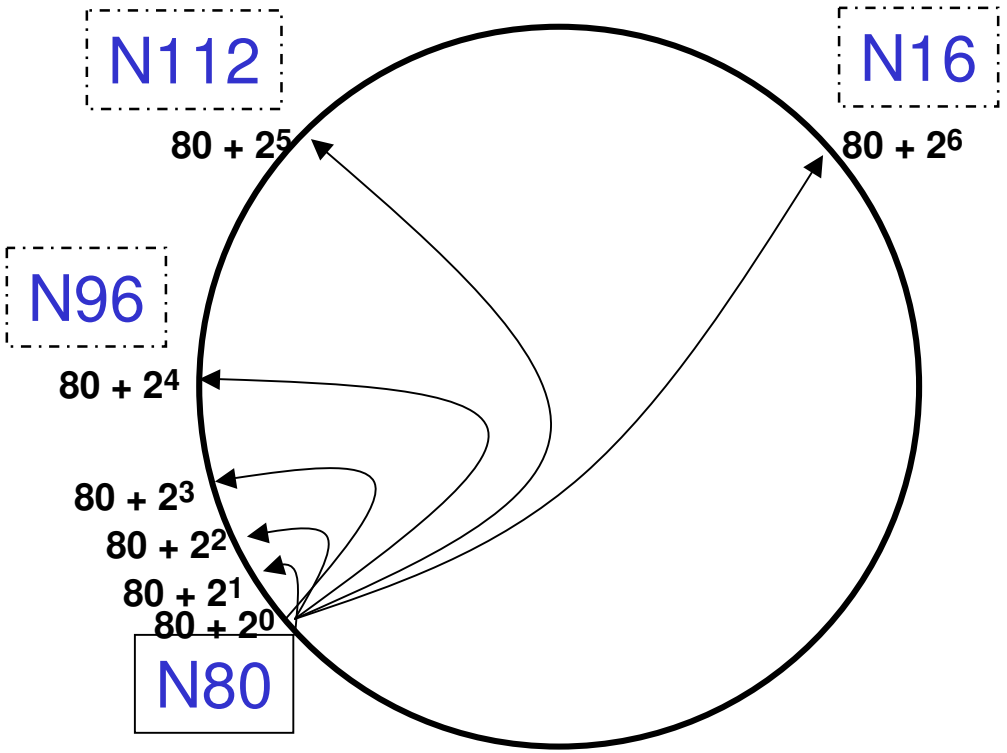
- Every node knows its successor in the ring



- requires O(N) time

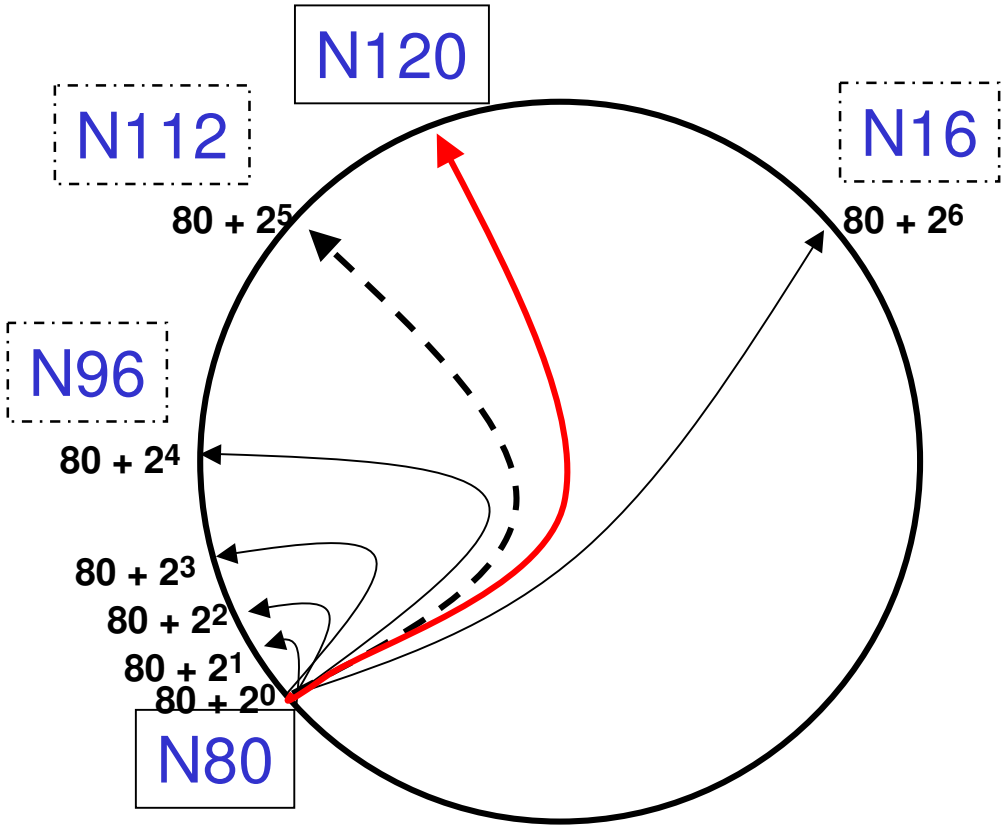
"Finger Tables"

- Every node knows *m* other nodes in the ring
- Increase distance exponentially



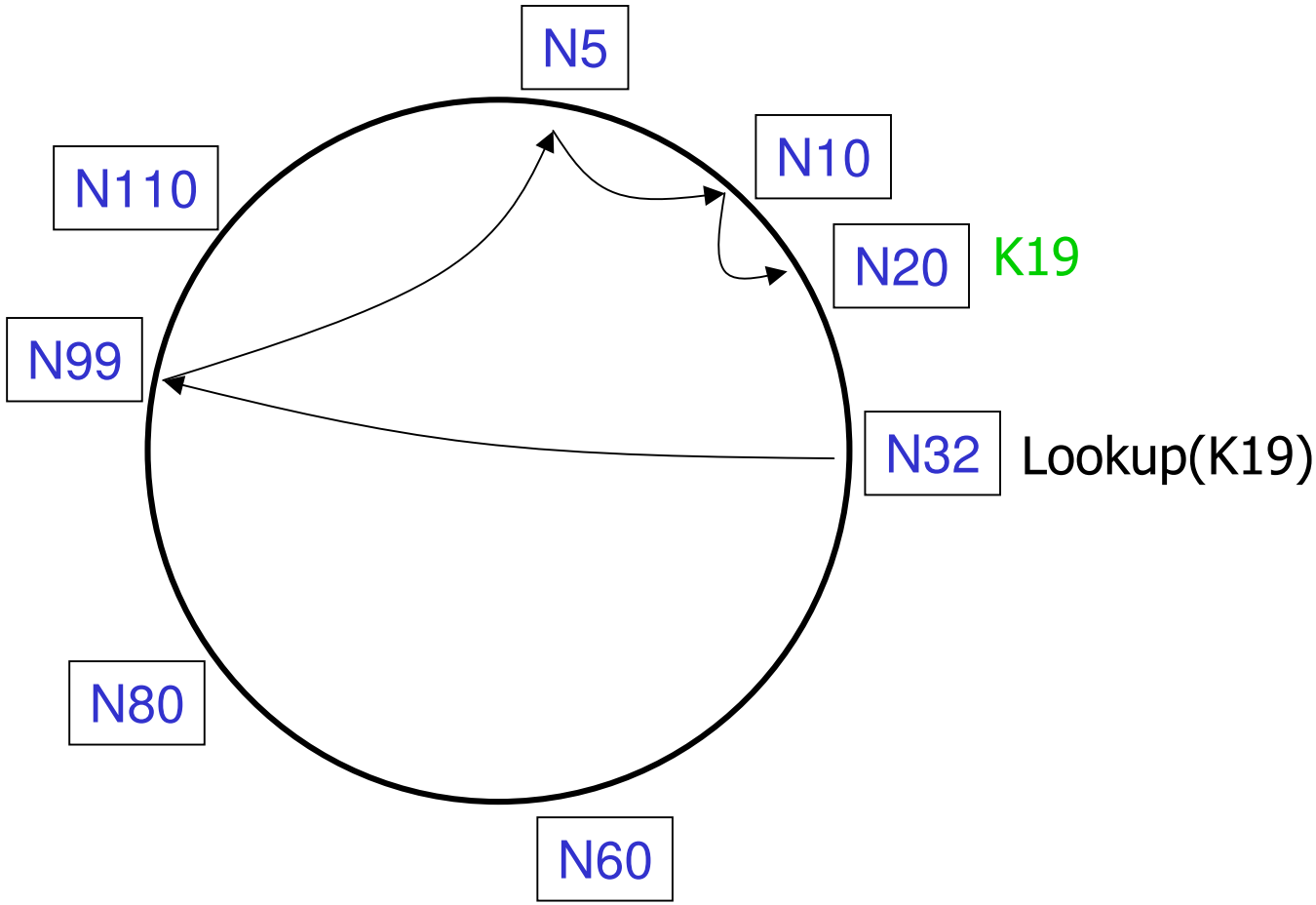
"Finger Tables"

- Finger *i* points to successor of $n+2^i$



Lookups are Faster

- Lookups take O(Log N) hops

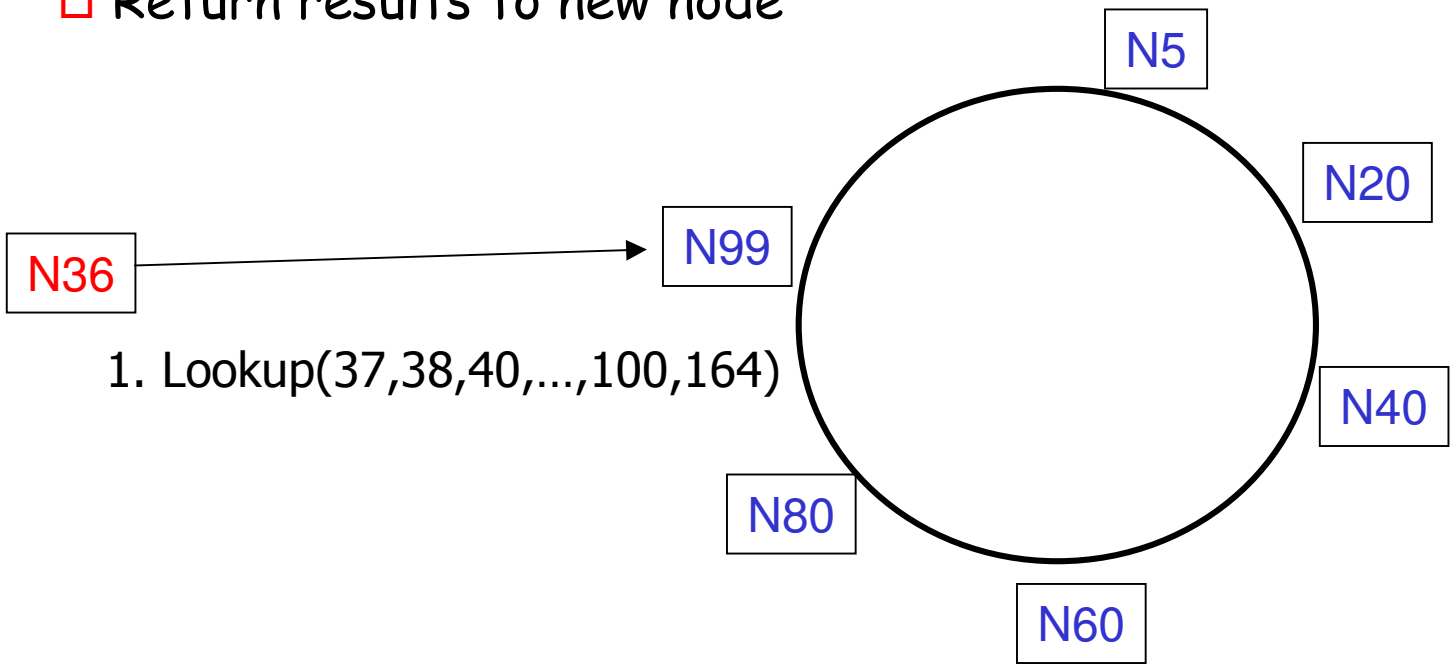


Joining the Ring

- Three step process:
  - Initialize all fingers of new node
  - Update fingers of existing nodes
  - Transfer keys from successor to new node
- Less aggressive mechanism (lazy finger update):
  - Initialize only the finger to successor node
  - Periodically verify immediate successor, predecessor
  - Periodically refresh finger table entries

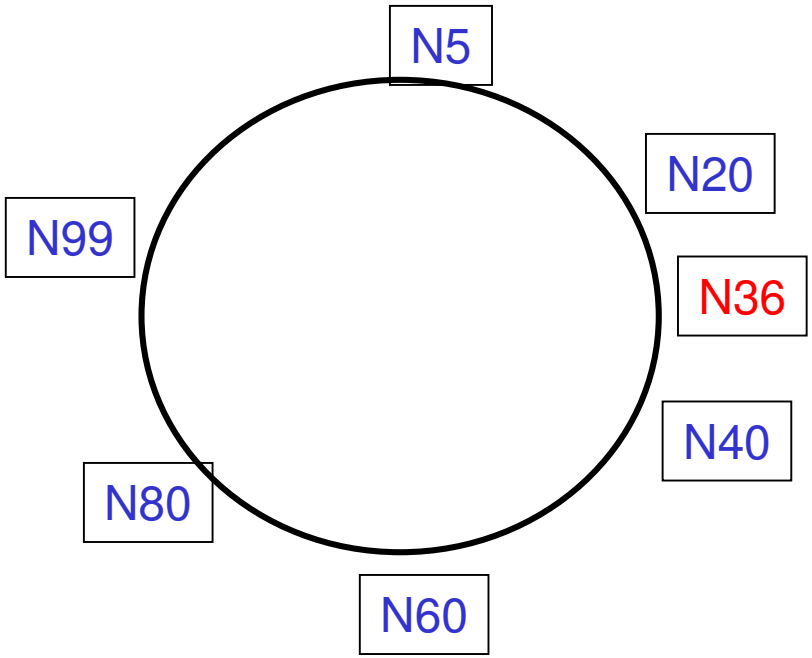
Joining the Ring - Step 1

- Initialize the new node finger table
  - Locate any node *p* in the ring
  - Ask node *p* to lookup fingers of new node N36
  - Return results to new node



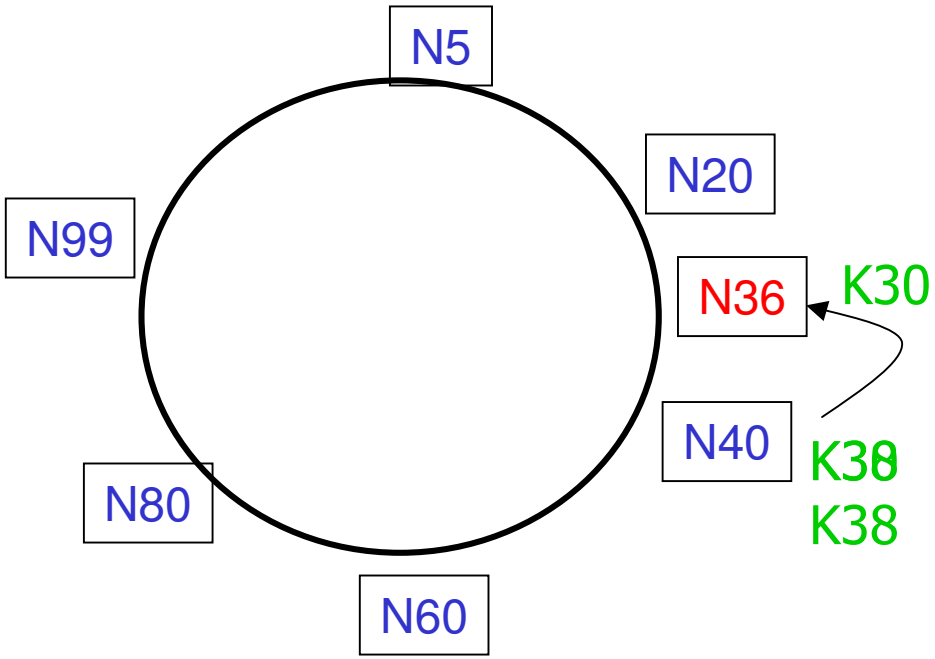
Joining the Ring - Step 2

- Updating fingers of existing nodes
  - new node calls update function on existing nodes
  - existing nodes can recursively update fingers of other nodes



Joining the Ring - Step 3

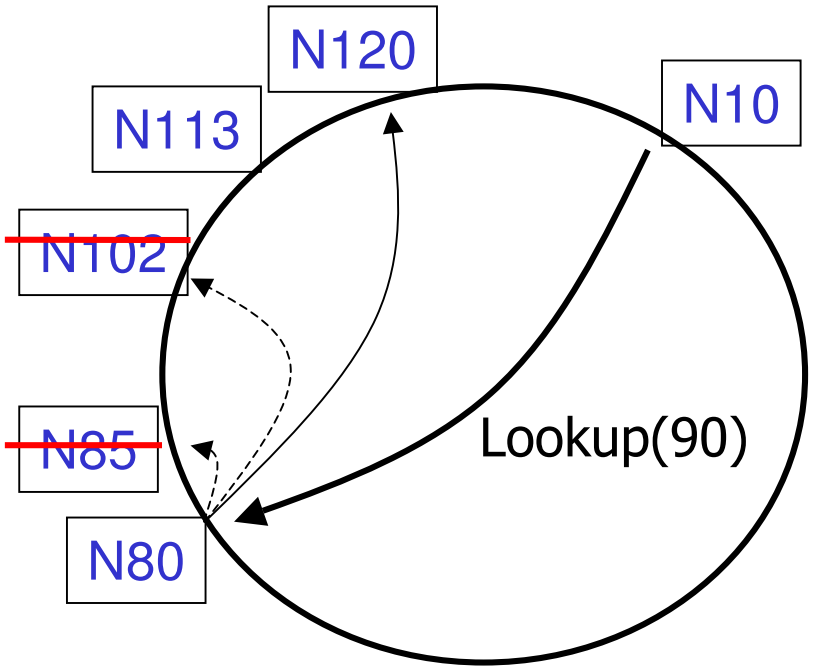
- Transfer keys from successor node to new node
  - only keys in the range are transferred



Copy keys 21..36
from N40 to N36

Handing Failures

- Failure of nodes might cause incorrect lookup



- N80 doesn't know correct successor, so lookup fails
- Successor fingers are enough for correctness

Handling Failures

- Use successor list
  - Each node knows *r* immediate successors
  - After failure, will know first live successor
  - Correct successors guarantee correct lookups

- Guarantee is with some probability
  - Can choose *r* to make probability of lookup failure arbitrarily small

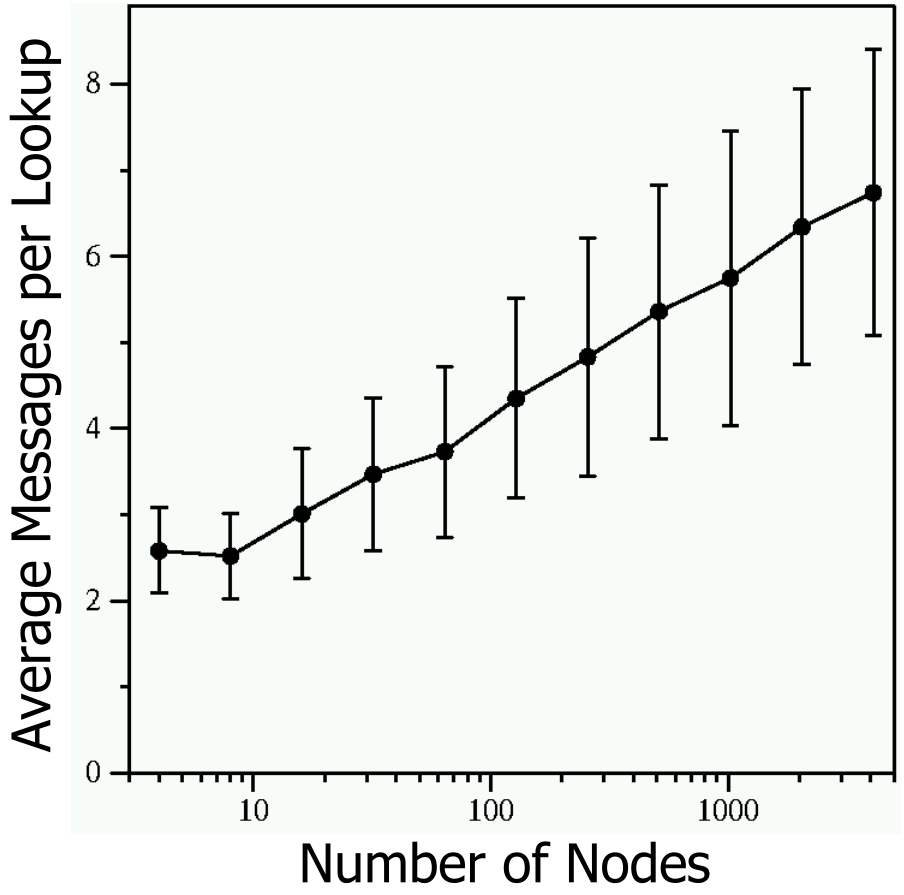
Evaluation Overview

- Quick lookup in large systems
- Low variation in lookup costs
- Robust despite massive failure

- Experiments confirm theoretical results

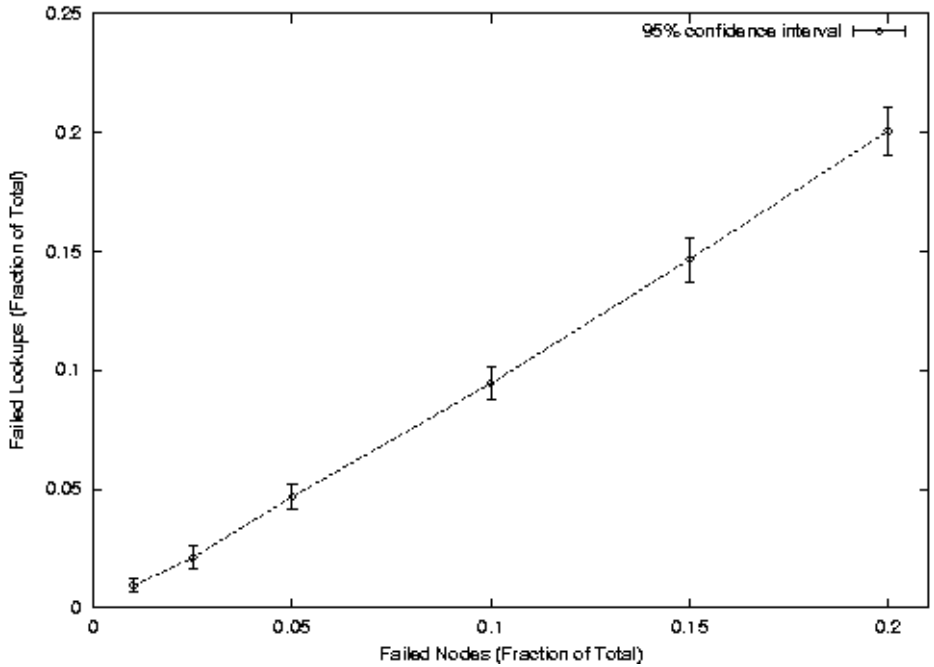
Cost of lookup

- Cost is O(Log N) as predicted by theory
- constant is 1/2



Robustness

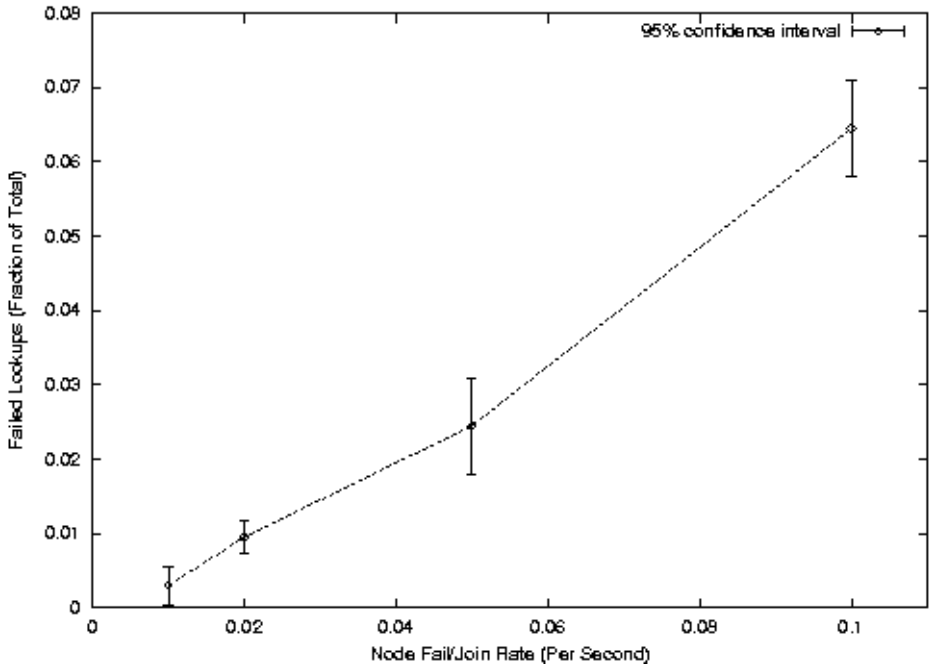
- Simulation results: static scenario
- Failed lookup means original node with key failed (no replica of keys)



- Result implies good balance of keys among nodes!

Robustness

- Simulation results: dynamic scenario
- Failed lookup means finger path has a failed node



- 500 nodes initially
- average *stabilize( )* call 30s
- 1 lookup per second (Poisson)
- x join/fail per second (Poisson)

Current implementation

- Chord library: 3,000 lines of C++
- Deployed in small Internet testbed
- Includes:
  - Correct concurrent join/fail
  - Proximity-based routing for low delay (?)
  - Load control for heterogeneous nodes (?)
  - Resistance to spoofed node IDs (?)

Strengths

- Based on theoretical work (consistent hashing)
- Proven performance in many different aspects
  - "with high probability" proofs
- Robust (Is it?)

Weakness

- **NOT** that simple (compared to CAN)
- Member joining is complicated
  - aggressive mechanisms requires too many messages and updates
  - no analysis of convergence in lazy finger mechanism
- Key management mechanism mixed between layers
  - upper layer does insertion and handle node failures
  - Chord transfer keys when node joins (no leave mechanism!)
- Routing table grows with # of members in group
- Worst case lookup can be slow

Discussions

- Network proximity (consider latency?)
- Protocol security
  - Malicious data insertion
  - Malicious Chord table information
- Keyword search and indexing
- ...

7. Tapestry: Decentralized Routing and Location

Ben Y. Zhao
CS Division, U. C. Berkeley

Outline

- Problems facing wide-area applications
- Tapestry Overview
- Mechanisms and protocols
- Preliminary Evaluation
- Related and future work

Motivation

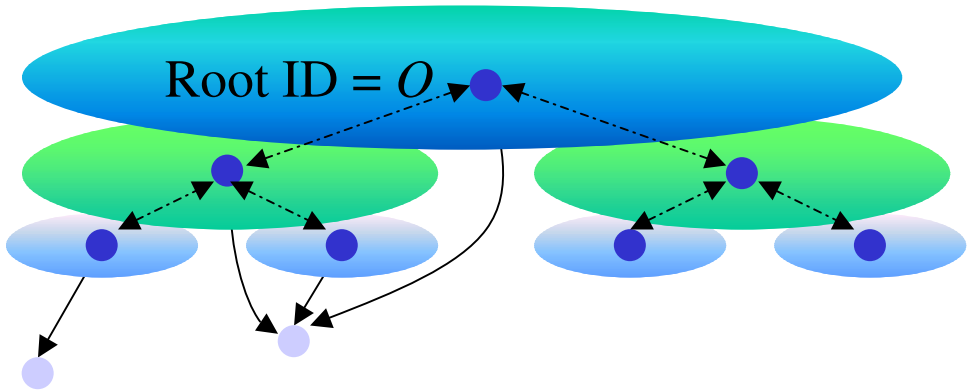
- Shared Storage systems need an data location/routing mechanism
  - Finding the peer in a scalable way is a difficult problem
  - Efficient insertion and retrieval of content in a large distributed storage infrastructure
- Existing solutions
  - Centralized: expensive to scale, less fault tolerant,vulnerable to DoS attacks (e.g. Napster,DNS,SDS)
  - Flooding: not scalable (e.g. Gnutella)

Key: Location and Routing

- Hard problem:
  - Locating and messaging to resources and data
- Approach: wide-area *overlay infrastructure*:
  - Scalable, Dynamic, Fault-tolerant, Load balancing

Decentralized Hierarchies

- Centralized hierarchies
  - Each higher level node responsible for locating objects in a greater domain
- Decentralize: Create a tree for object *O* (really!)
  - Object *O* has its own root and subtree
  - Server on each level keeps pointer to **nearest** object in domain
  - Queries search up in hierarchy



Directory servers tracking 2 replicas

What is Tapestry?

- A prototype of a *decentralized, scalable, fault-tolerant, adaptive* location and routing infrastructure
  (Zhao, Kubiatowicz, Joseph et al. U.C. Berkeley)
- Network layer of OceanStore global storage system
  Suffix-based hypercube routing
  - Core system inspired by **Plaxton Algorithm** (Plaxton, Rajamaran, Richa (SPAA97))
- Core API:
  - *publishObject(ObjectID, [serverID])*
  - *sendmsgToObject(ObjectID)*
  - *sendmsgToNode(NodeID)*

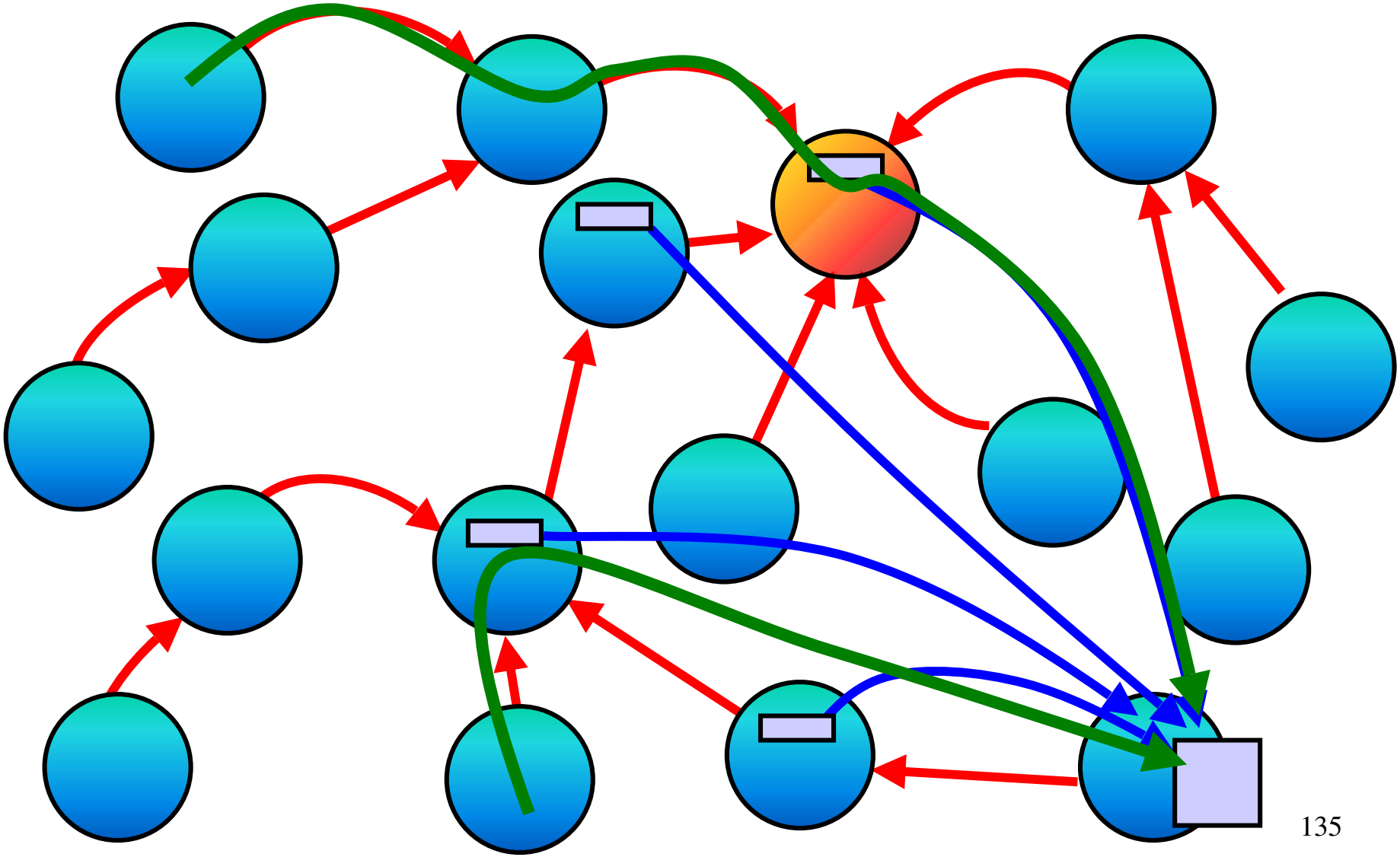
Incremental Suffix Routing

- Namespace (nodes and objects)
  - large enough to avoid collisions (~2^{160}?) (size N in $Log_2(N)$ bits)
- Insert Object:
  - Hash Object into namespace to get ObjectID
  - For ($i=0$, $i<Log_2(N)$, $i+j$) { //Define hierarchy
    - j is base of digit size used, (j = 4 → hex digits)
    - Insert entry into nearest node that matches on last i bits
    - When no matches found, then pick node matching ($i - n$) bits with highest ID value, terminate

Routing to Object

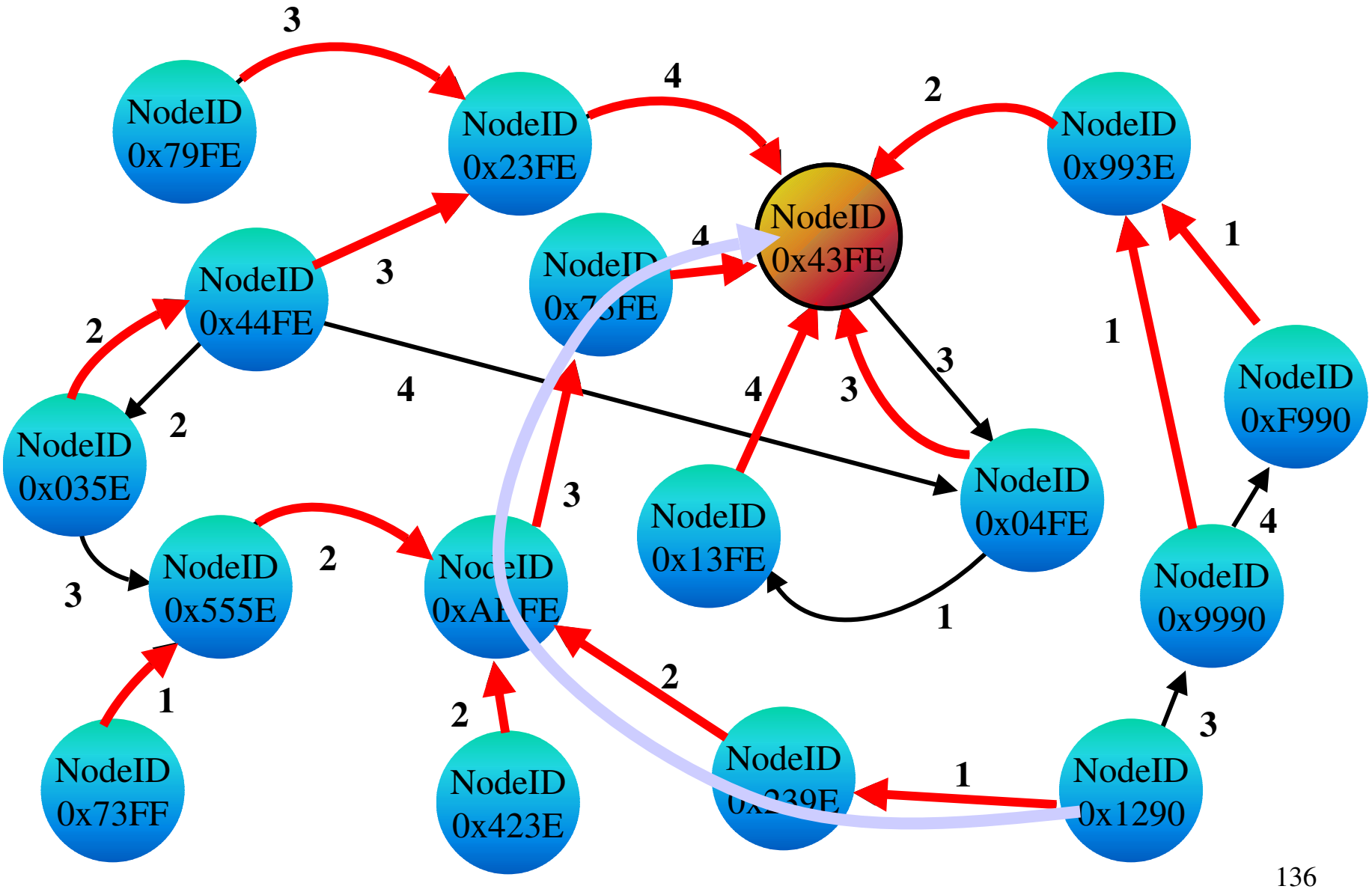
- Lookup object
  - Traverse same relative nodes as insert, **except searching for entry at each node**
  - For ($i=0$, $i<Log_2(N)$, $i+n$)
    Search for entry in nearest node matching on last i bits
- Each object maps to hierarchy defined by single root
  - f(ObjectID) = RootID
- Publish / search both route incrementally to root
- Root node = $f(O)$, is responsible for "knowing" object's location

Object Location
Randomization *and* Locality



Tapestry Mesh

Incremental suffix-based routing



Contribution of this work

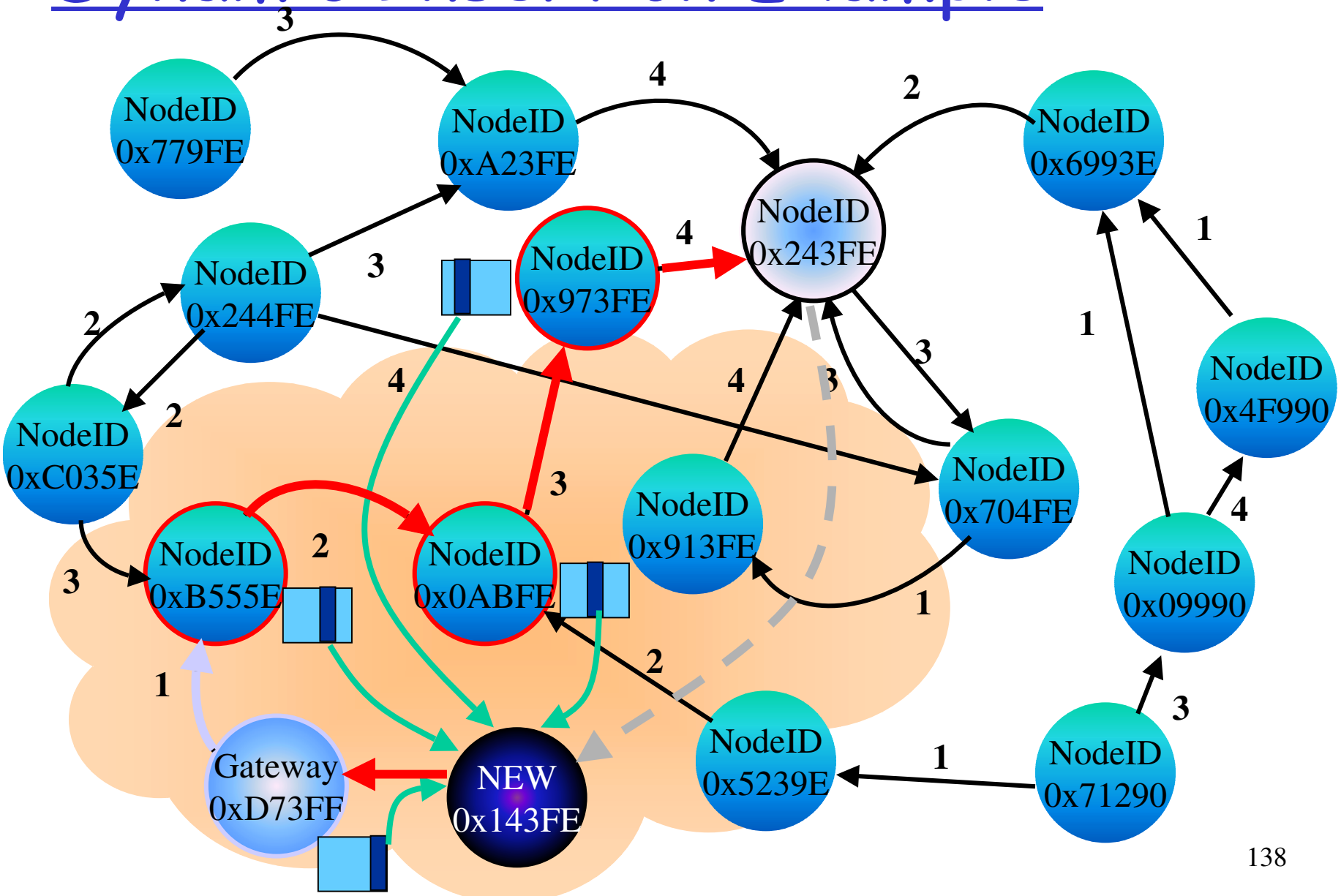
Plaxtor Algorithm

- Limitations
  - Global knowledge algorithms
  - Root node vulnerability
  - Lack of adaptability

Tapestry

- Distributed algorithms
  - Dynamic node insertion
  - Dynamic root mapping
  - Redundancy in location and routing
  - Fault-tolerance protocols
  - *Self-configuring / adaptive*
  - *Support for mobile objects*
- Application Infrastructure

Dynamic Insertion Example



Fault-tolerant Location

- Minimized soft-state vs. explicit fault-recovery
- Multiple roots
  - Objects hashed w/ small salts → multiple names/roots
  - Queries and publishing utilize all roots in parallel
  - P(finding Reference w/ partition) = $1 - (1/2)^n$ where n = # of roots
- Soft-state periodic republish
  - 50 million files/node, daily republish, b = 16, $N = 2^{160}$, 40B/msg, worst case update traffic: 156 kb/s,
  - expected traffic w/ 2^{40} real nodes: 39 kb/s

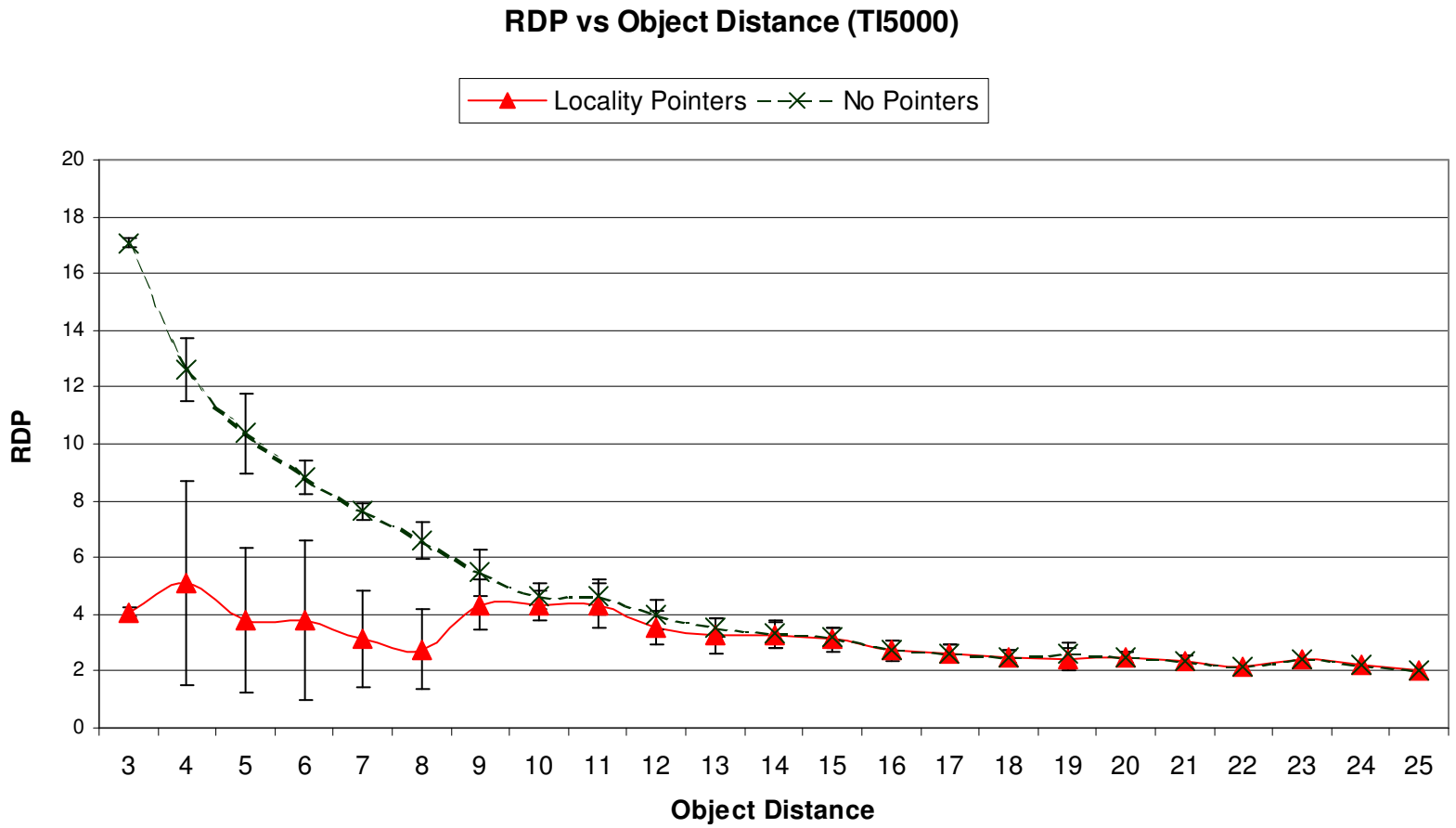
Fault-tolerant Routing

- Detection:
  - Periodic probe packets between neighbors
- Handling:
  - Each entry in routing map has 2 alternate nodes
  - Second chance algorithm for intermittent failures
  - Long term failures → alternates found via routing tables
- Protocols:
  - First Reachable Link Selection
  - Proactive Duplicate Packet Routing

Simulation Environment

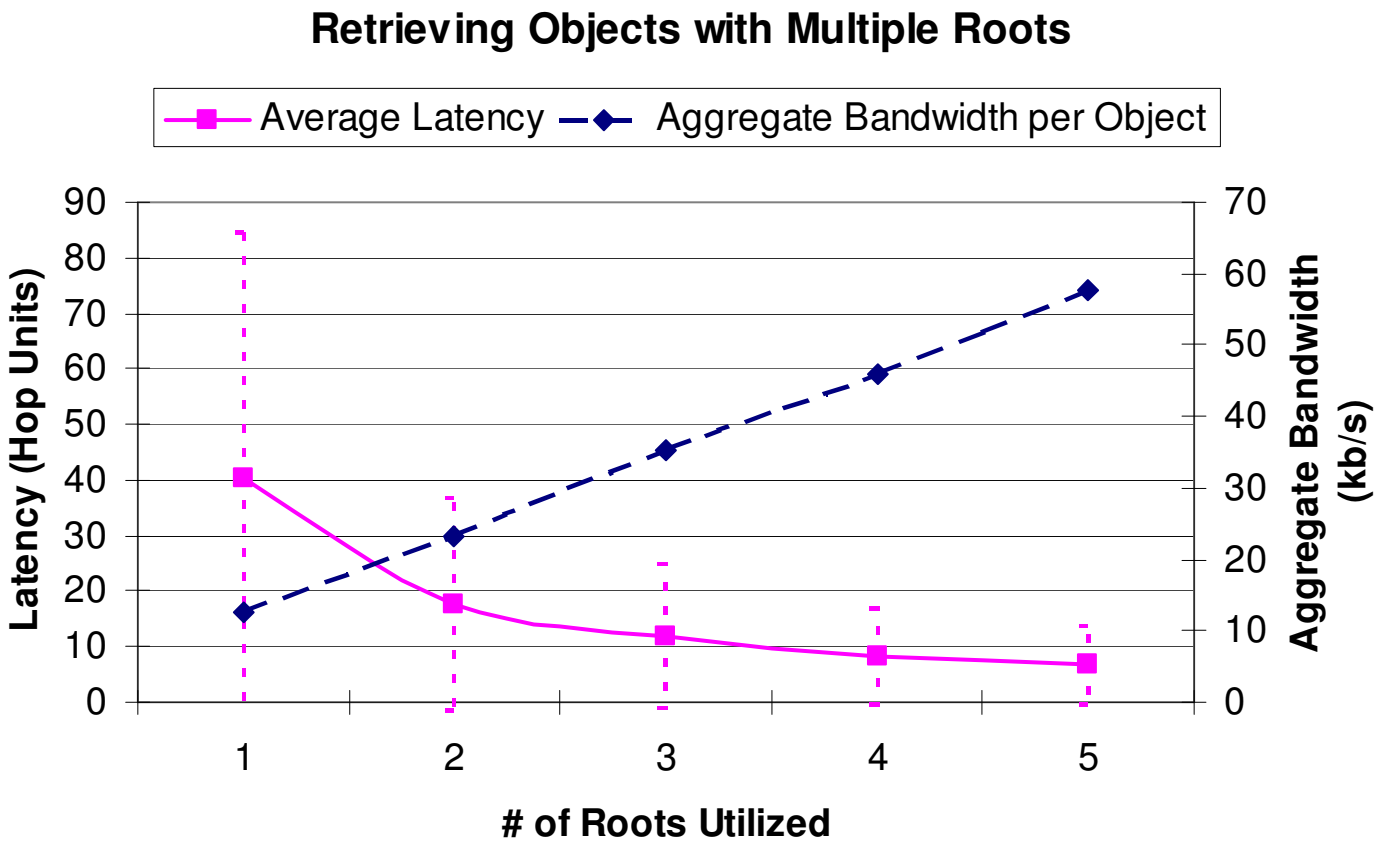
- Implemented Tapestry routing as packet-level simulator
- Delay is measured in terms of network hops
- Do not model the effects of cross traffic or queuing delays
- Four topologies: AS, MBone, GT-ITM, TIERS

Results: Location Locality



Measuring effectiveness of locality pointers (TIERS 5000)

Results: Stability via Redundancy



Parallel queries on multiple roots. Aggregate bandwidth measures b/w used for soft-state republish 1/day and b/w used by requests at rate of 1/s.

Related Work

- Content Addressable Networks
  - Ratnasamy et al., (ACIRI / UCB)
- Chord
  - Stoica, Morris, Karger, Kaashoek, Balakrishnan (MIT / UCB)
- Pastry
  - Druschel and Rowstron (Rice / Microsoft Research)

Strong Points

- Designed system based on Theoretically proven idea (Plaxton Algorithm)
- Fully decentralized and scalable solution for deterministic location and routing problem

Weaknesses/Improvements

- Substantially complicated
  - Esp, dynamic node insertion algorithm is non-trivial, and each insertion will take a non-negligible amount of time.
  - Attempts to insert a lot of nodes at the same time
- Where to put "root" node for a given object
  - Needs universal hashing function
  - Possible to put "root" to near expected clients dynamically?

Routing to Nodes

Example: Octal digits, 2^{18} namespace, 005712 → 627510

005712	0	1	2	3	4	5	6	7
340880	0	1	2	3	4	5	6	7
943210	0	1	2	3	4	5	6	7
834510	0	1	2	3	4	5	6	7
387510	0	1	2	3	4	5	6	7
727510	0	1	2	3	4	5	6	7
627510	0	1	2	3	4	5	6	7

Neighbor Map
For "5712" (Octal)

0712	x012	xx02	xxx0
1712	x112	5712	xxx1
2712	x212	xx22	5712
3712	x312	xx32	xxx3
4712	x412	xx42	xxx4
5712	x512	xx52	xxx5
6712	x612	xx62	xxx6
7712	5712	xx72	xxx7
4	3	2	1

Routing Levels

Dynamic Insertion

Operations necessary for *N* to become fully integrated:

- Step 1: Build up *N*s routing maps
  - Send messages to each hop along path from gateway to current node *N'* that best approximates *N*
  - The *i*th hop along the path sends its *i*th level route table to *N*
  - *N* optimizes those tables where necessary
- Step 2: Send notify message via acked multicast to nodes with null entries for *N*s ID, setup forwarding ptrs
- Step 3: Each notified node issues republish message for relevant objects
- Step 4: Remove forward ptrs after one republish period

Dynamic Root Mapping

- Problem: choosing a root node for every object
  - Deterministic over network changes
  - Globally consistent
- Assumptions
  - All nodes with same matching suffix contains same null/non-null pattern in next level of routing map
  - Requires: consistent knowledge of nodes across network

Plaxton Solution

- Given desired ID *N*,
  - Find set *S* of nodes in existing network nodes *n* matching most # of suffix digits with *N*
  - Choose S_i = node in *S* with highest valued ID
- Issues:
  - Mapping must be generated statically using global knowledge
  - Must be kept as hard state in order to operate in changing environment
  - Mapping is not well distributed, many nodes in *n* get no mappings

8. A Scalable, Content-Addressable Network

Sylvia Ratnasamy[1,2], Paul Francis, [3]Mark Handley, [1]

Richard Karp[1,2], Scott Shenker [1]

[1] ACIRI

[2] U.C.Berkeley

[3] Tahoe Networks

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Internet-scale hash tables

- Hash tables
  - essential building block in software systems
- Internet-scale distributed hash tables
  - equally valuable to large-scale distributed systems?

Internet-scale hash tables

- Hash tables
  - essential building block in software systems
- Internet-scale distributed hash tables
  - equally valuable to large-scale distributed systems?
    - peer-to-peer systems
      - Napster, Gnutella,, FreeNet, MojoNation…
    - large-scale storage management systems
      - Publius, OceanStore,, CFS ...
    - mirroring on the Web

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)
- Properties
  - scalable
  - operationally simple
  - good performance (w/ improvement)

Content-Addressable Network (CAN)

- CAN: Internet-scale hash table
- Interface
  - insert(key,value)
  - value = retrieve(key)
- Properties
  - scalable
  - operationally simple
  - good performance
- Related systems: Chord/Pastry/Tapestry/Buzz/Plaxton ...

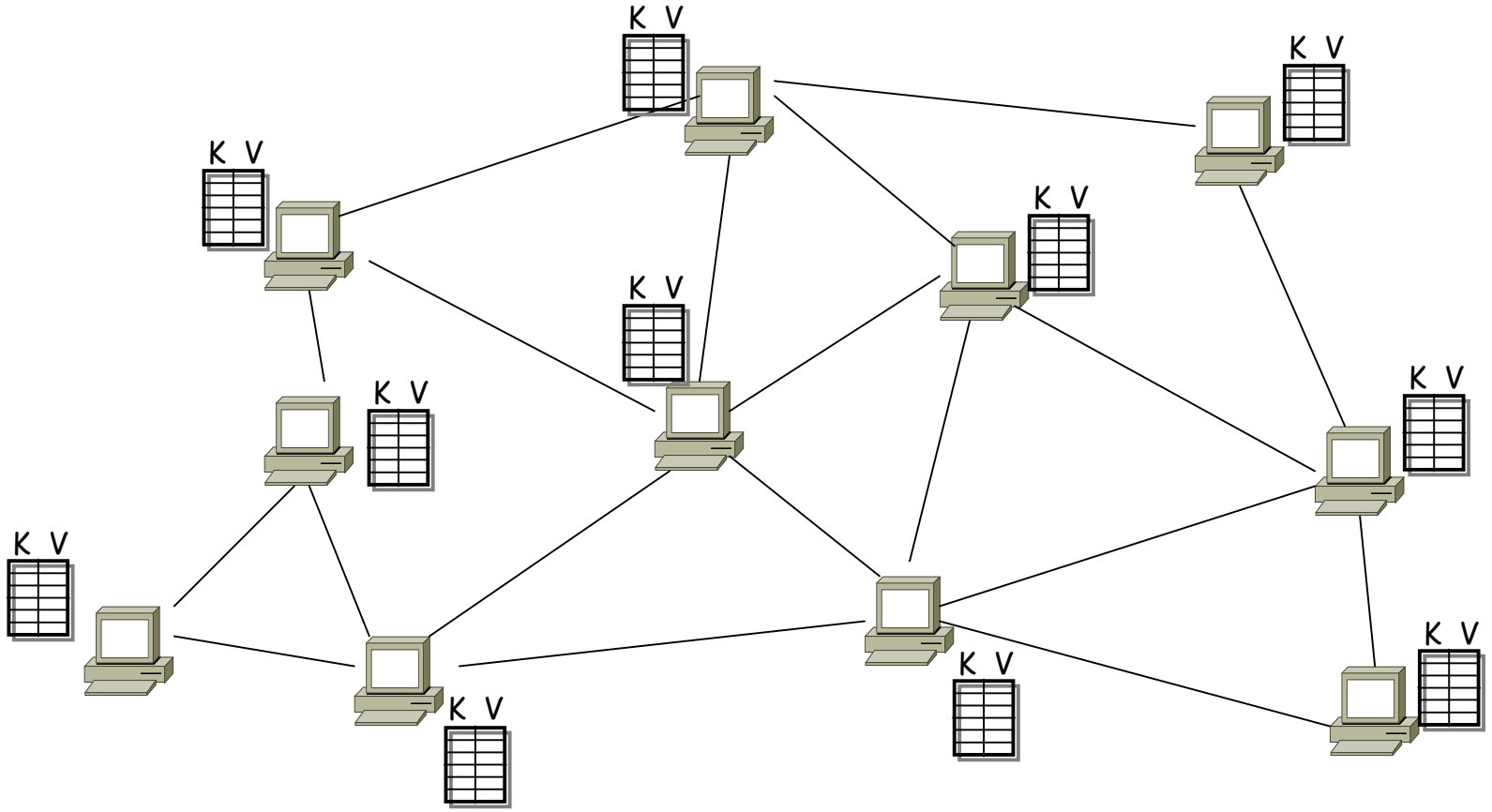
Problem Scope

- ✔ Design a system that provides the interface
  - scalability
  - robustness
  - performance
  - × security
- ✖ Application-specific, higher level primitives
  - keyword searching
  - mutable content
  - anonymity

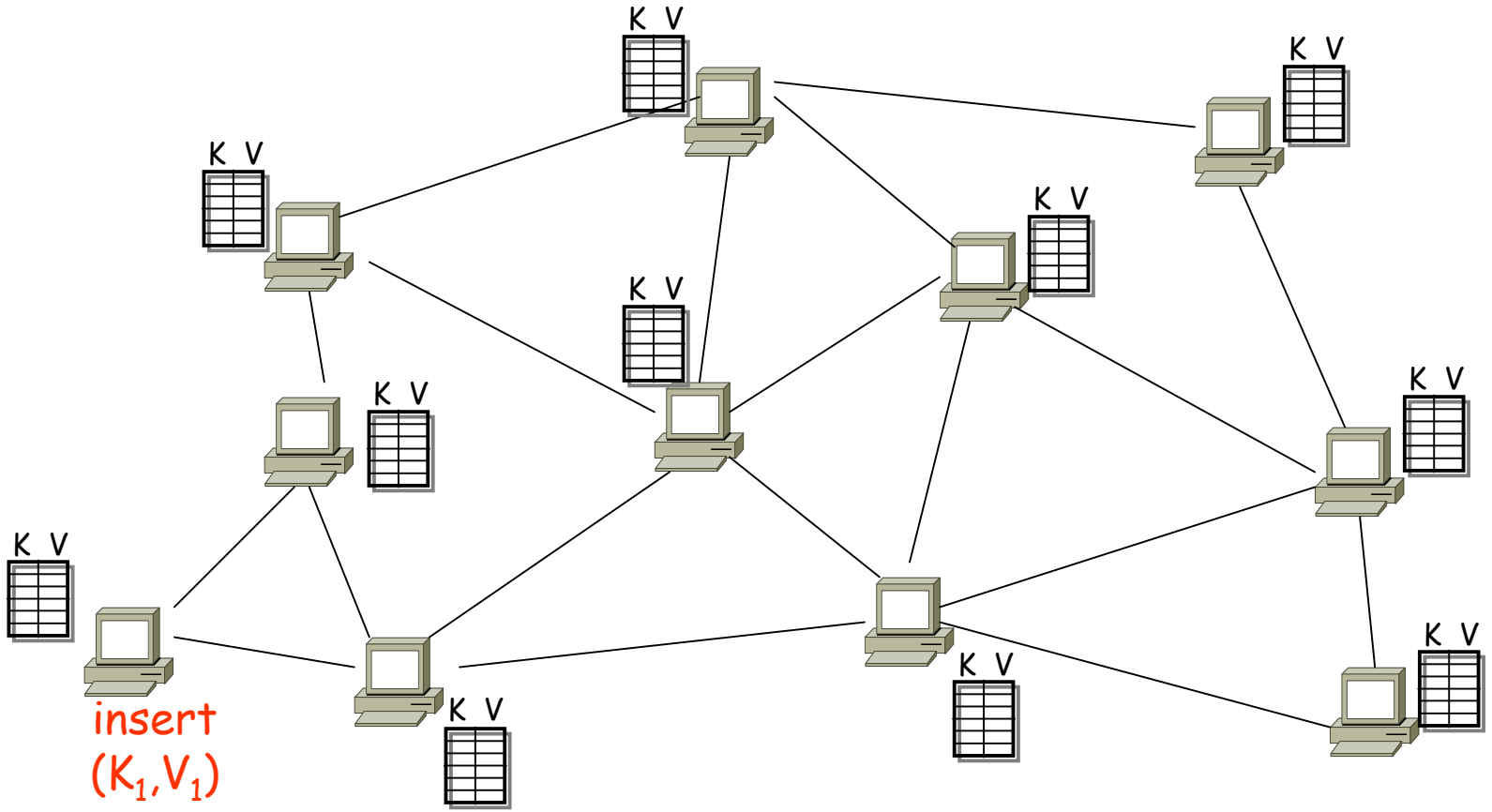
Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

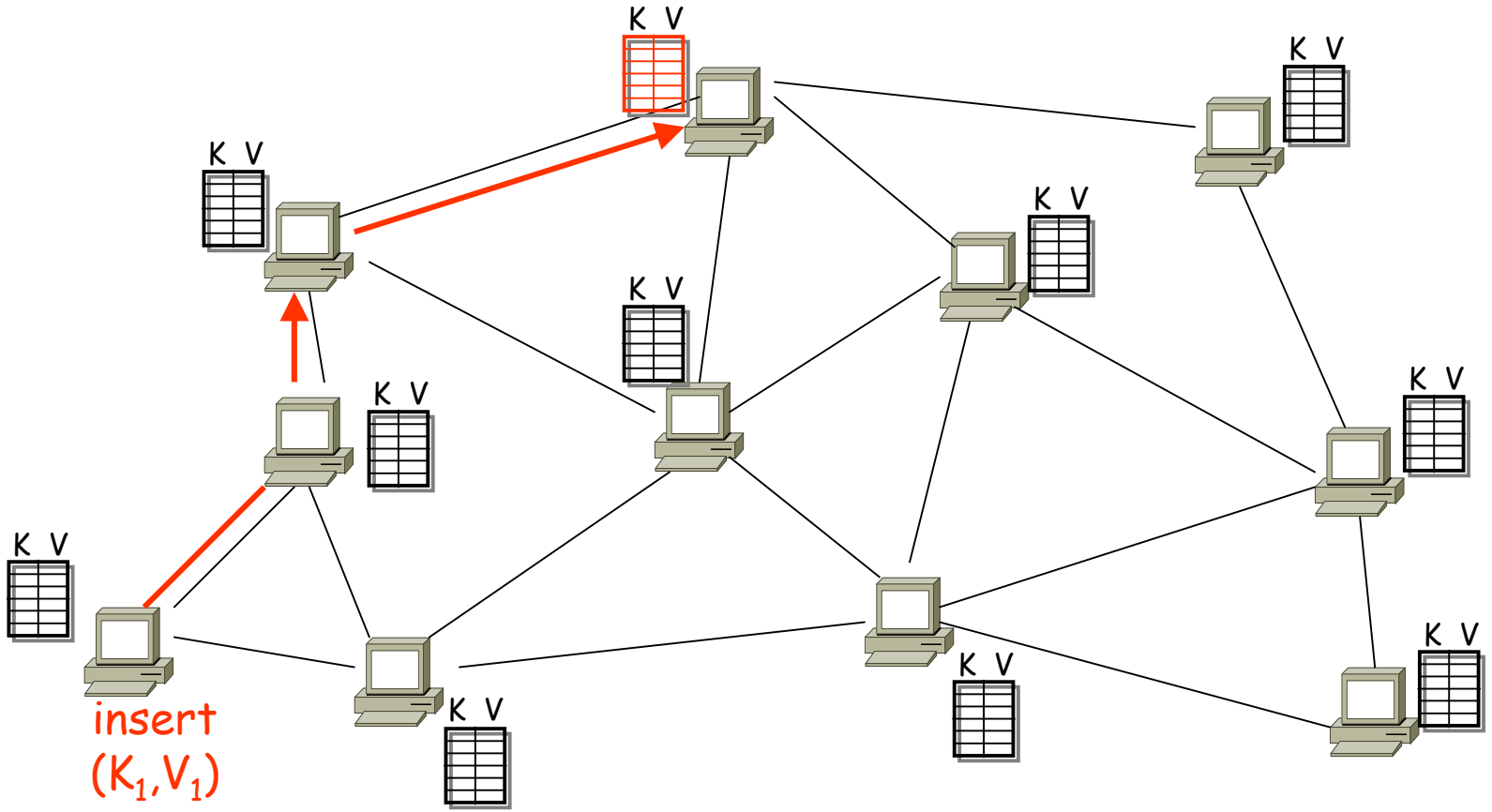
CAN: basic idea



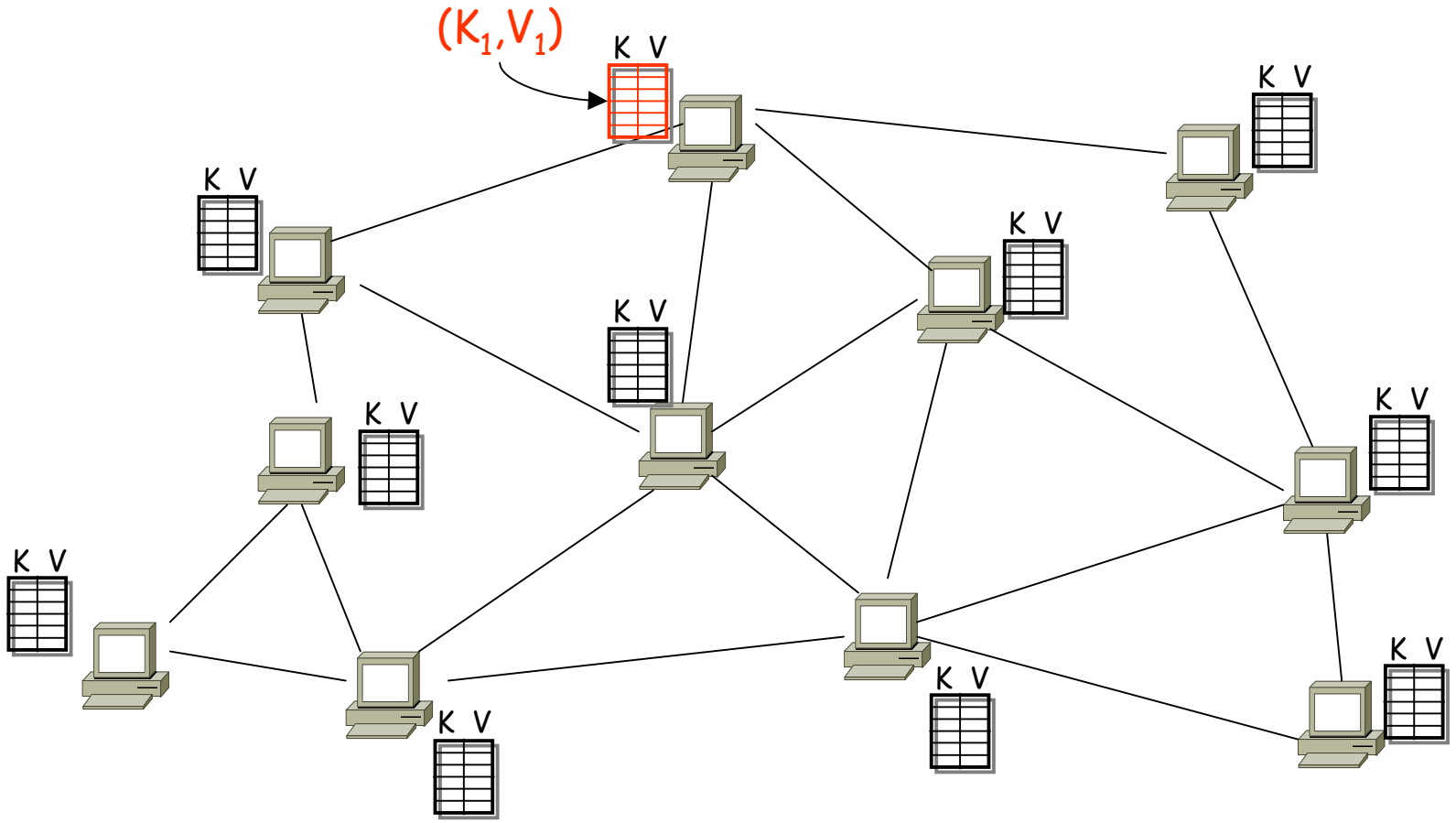
CAN: basic idea



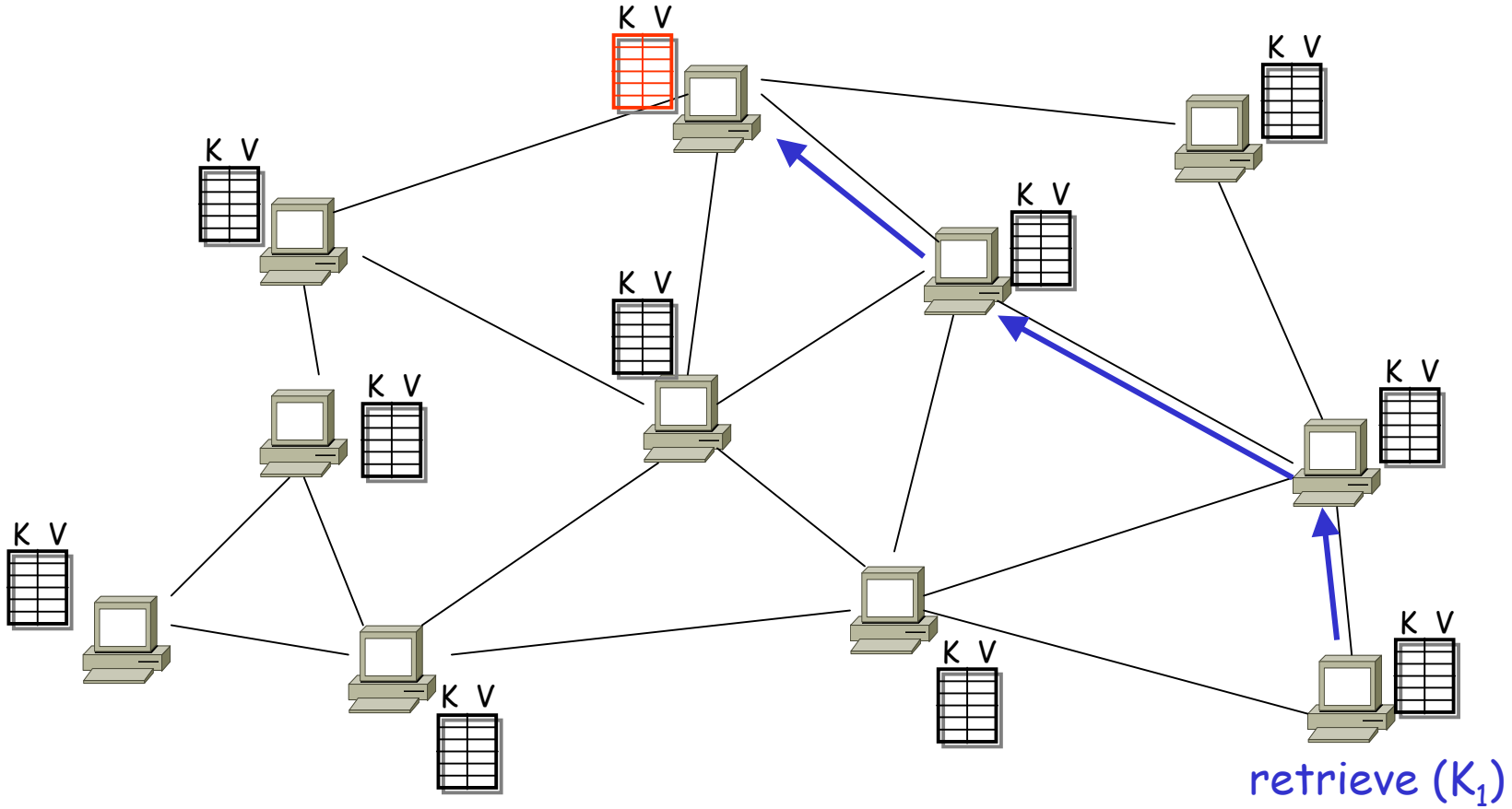
CAN: basic idea



CAN: basic idea



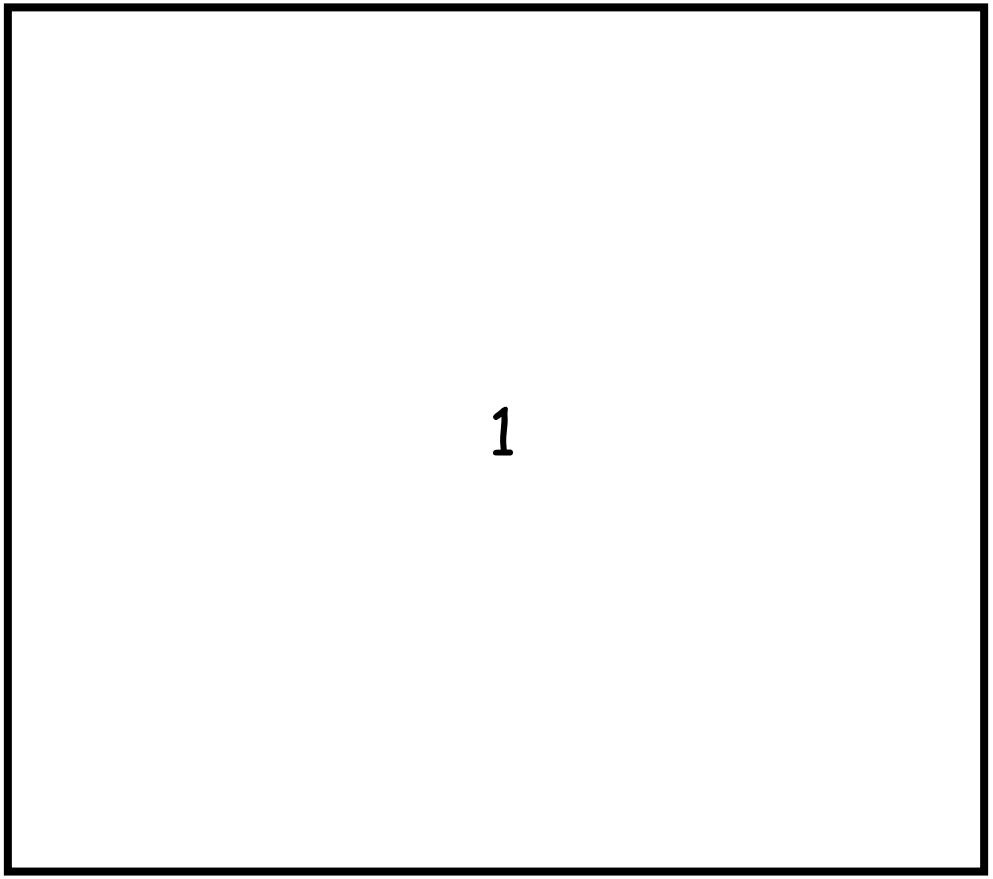
CAN: basic idea



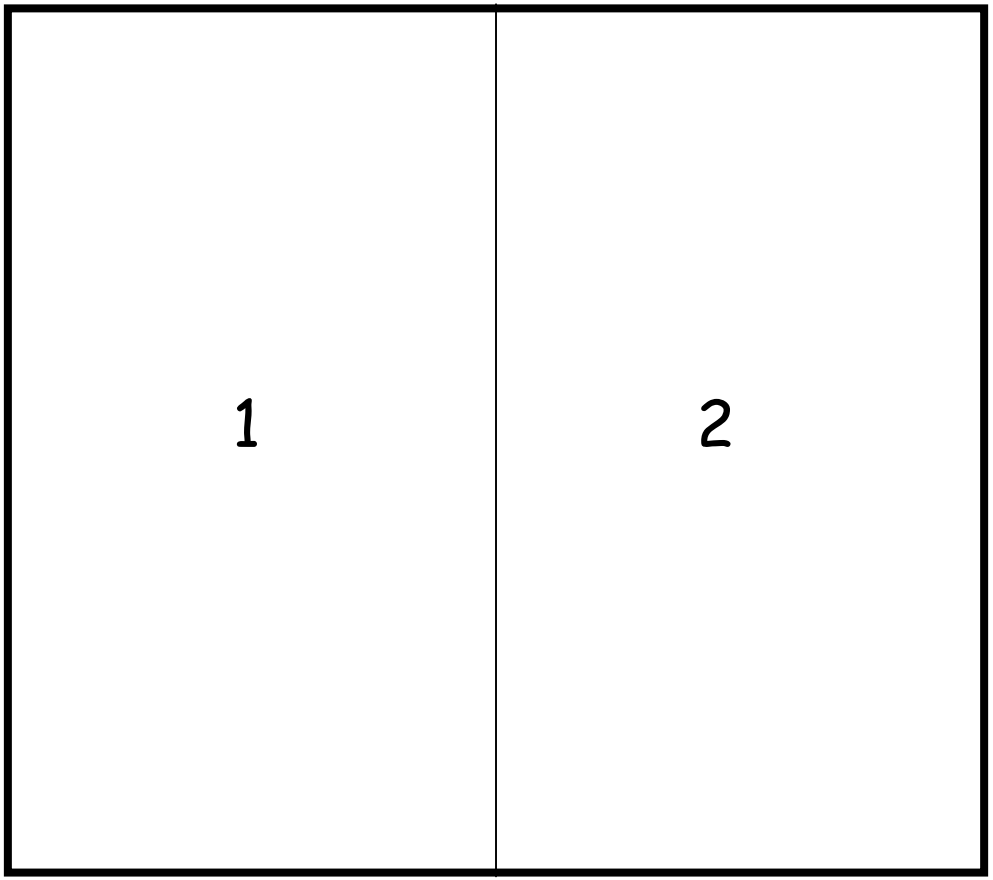
CAN: solution

- virtual Cartesian coordinate space
- entire space is partitioned amongst all the nodes
  - every node "owns" a zone in the overall space
- abstraction
  - can store data at "points" in the space
  - can route from one "point" to another
- point = node that owns the enclosing zone

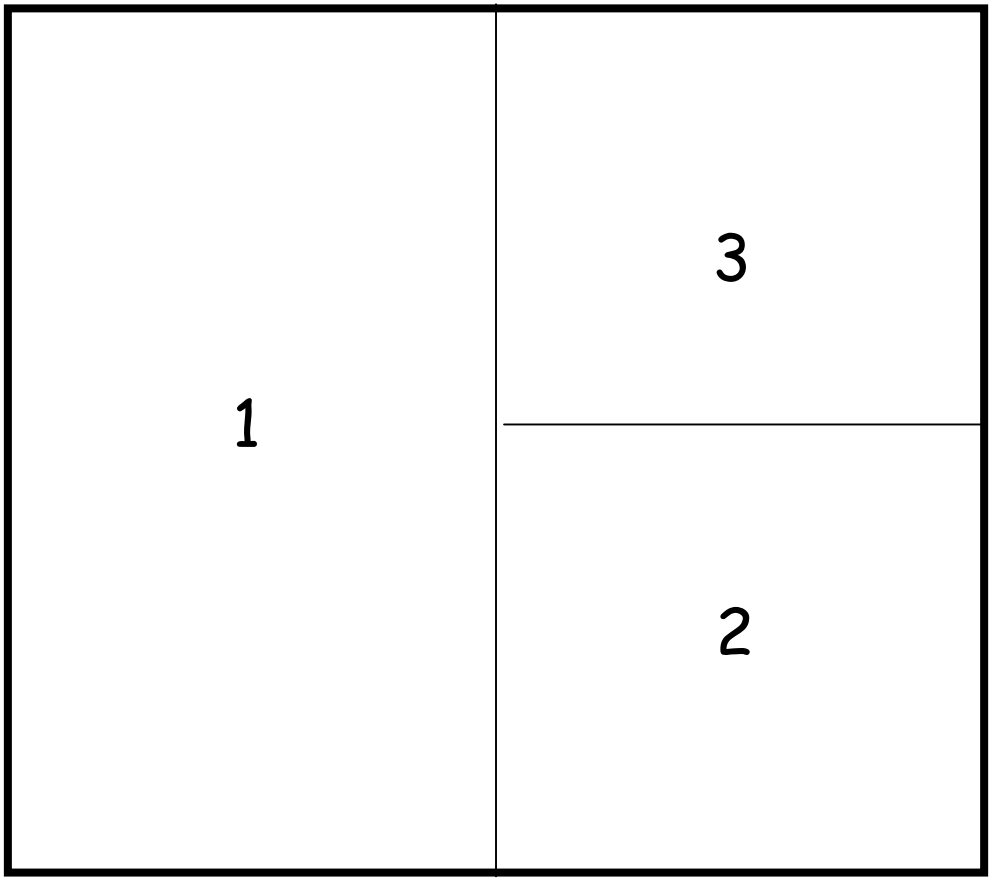
CAN: simple example



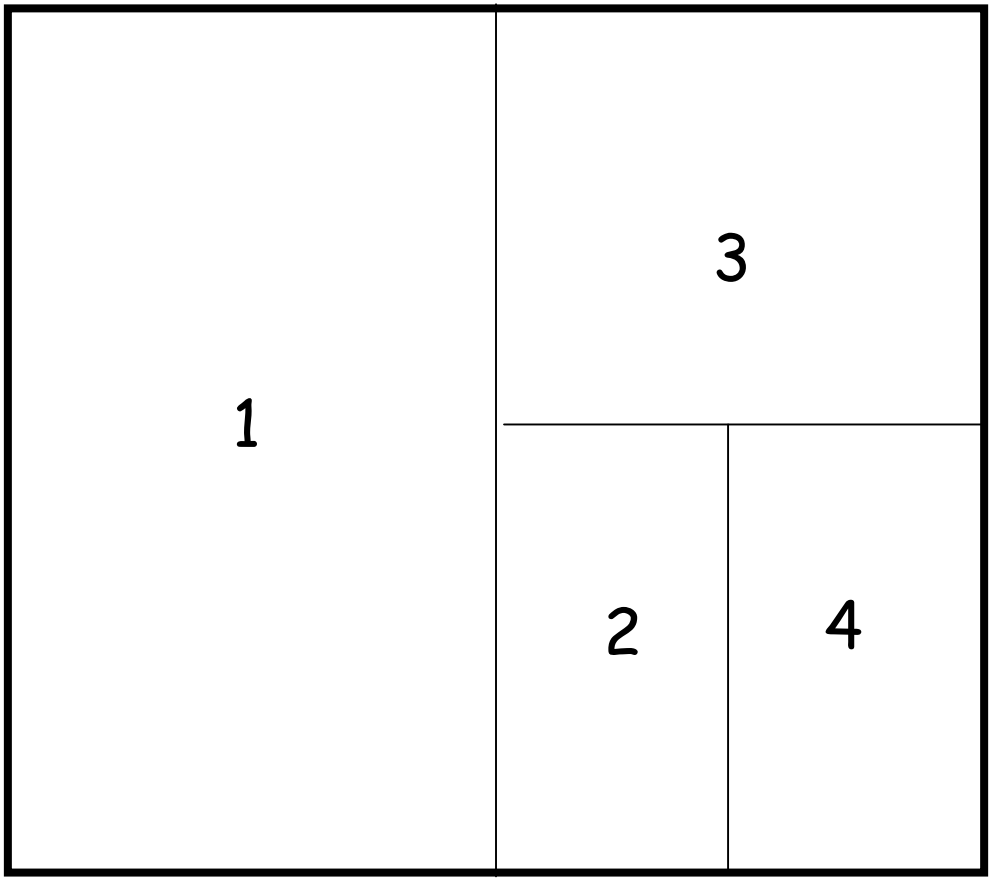
CAN: simple example



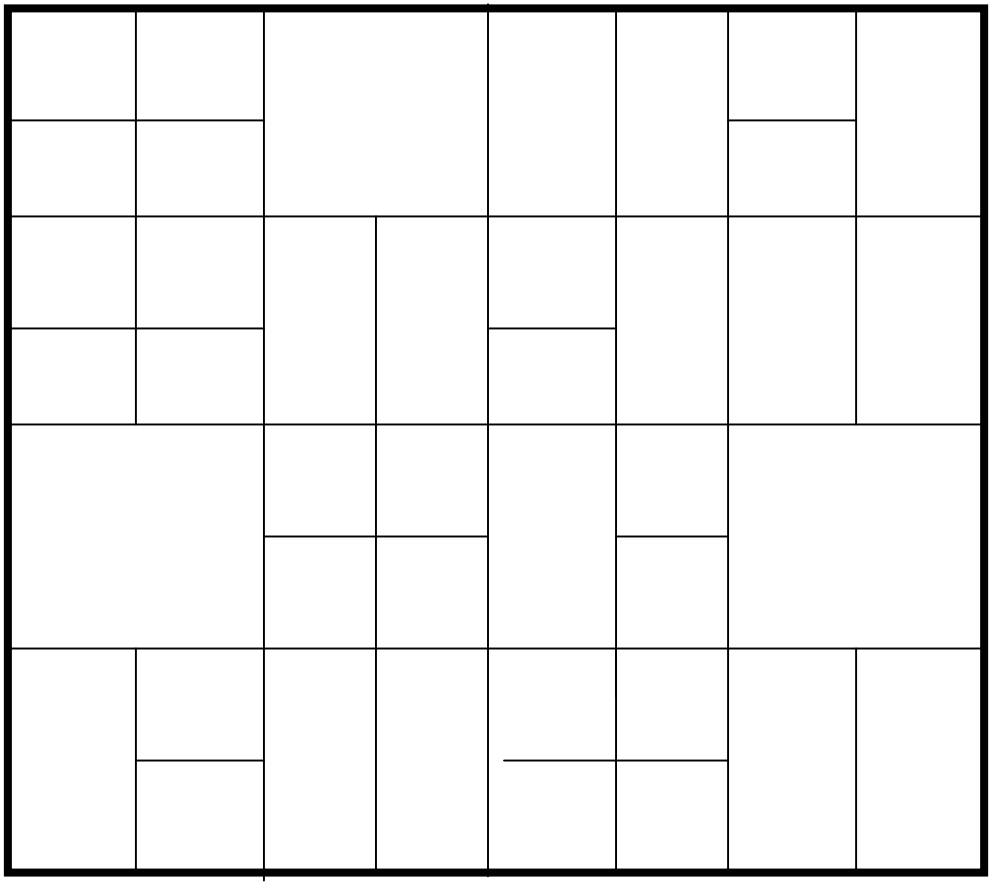
CAN: simple example



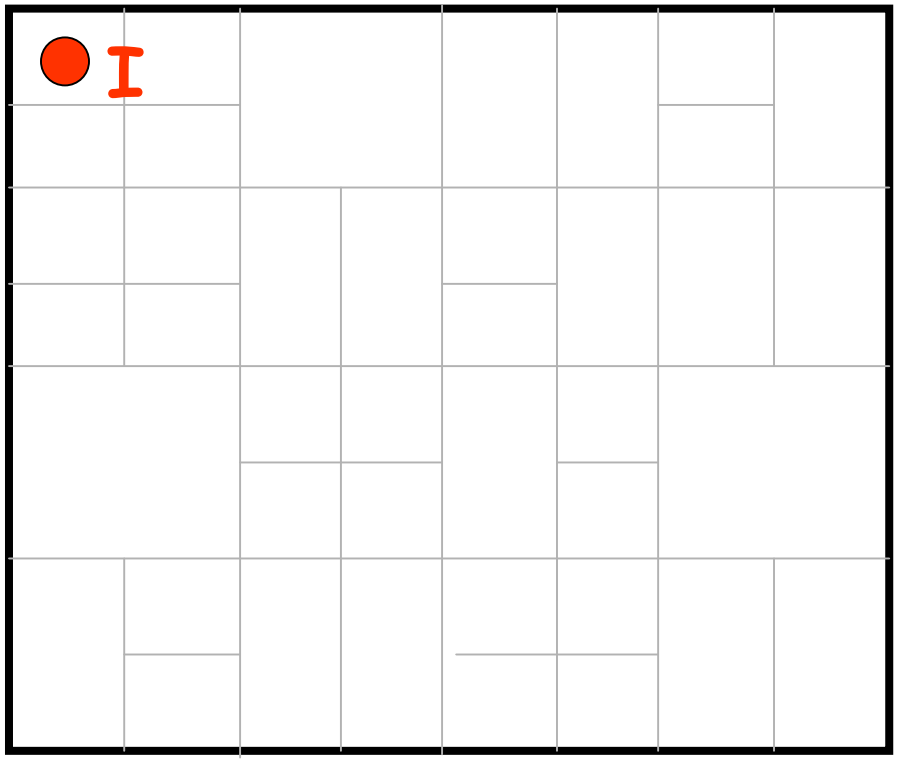
CAN: simple example



CAN: simple example

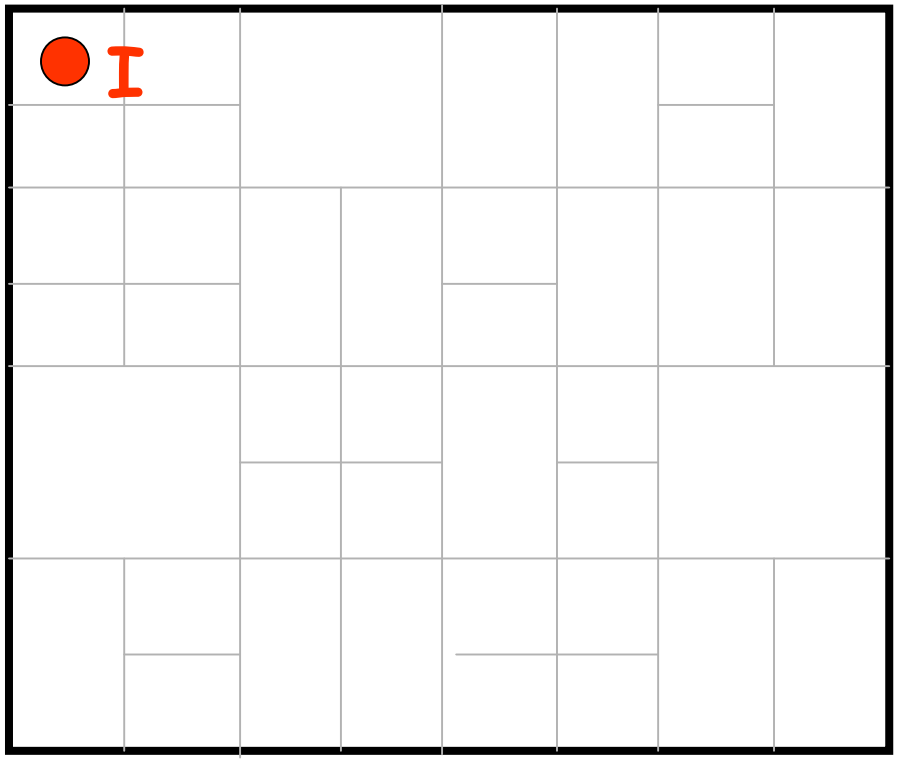


CAN: simple example



CAN: simple example

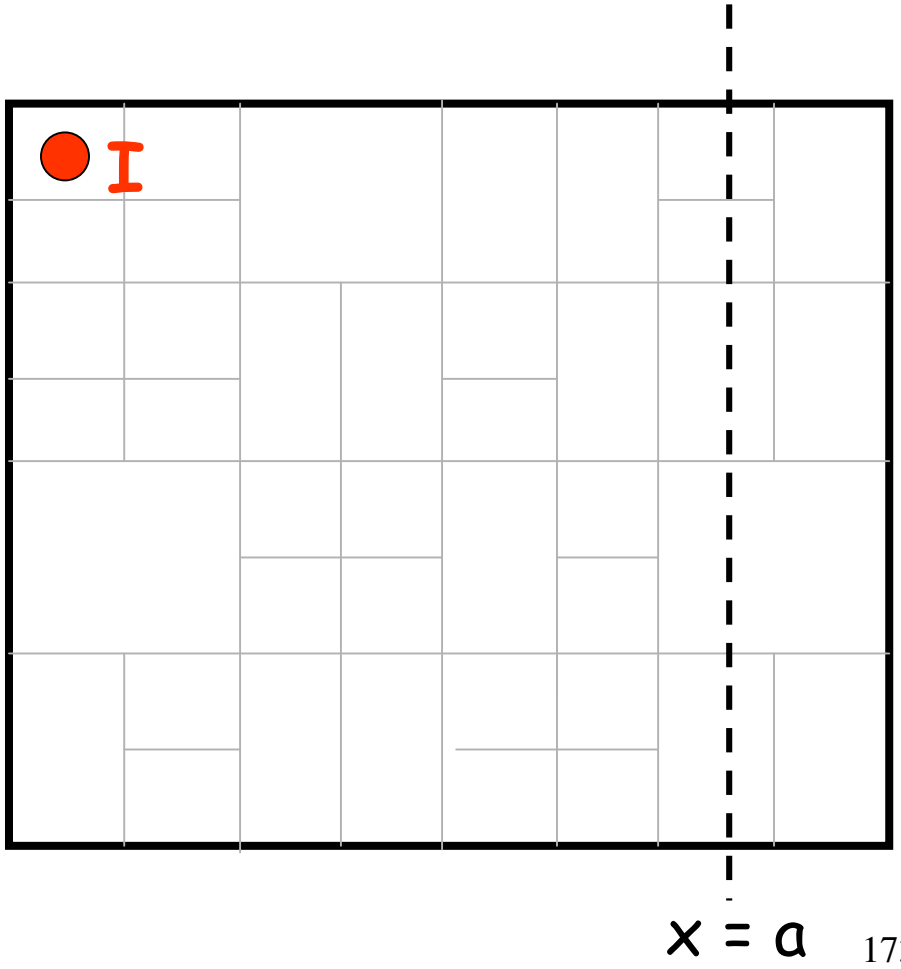
node I::insert(K,V)



CAN: simple example

node I::insert(K,V)

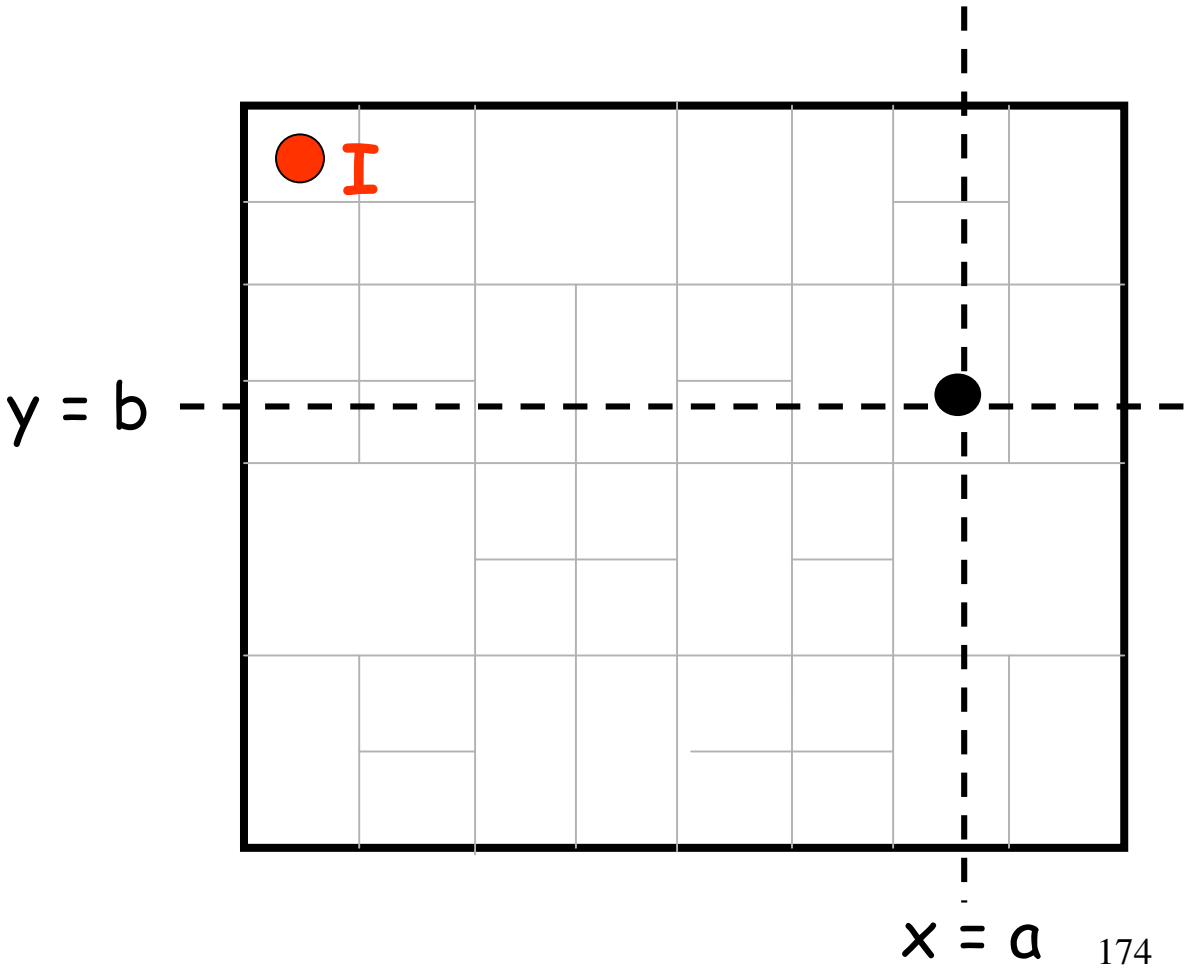
(1) a = h_x(K)



CAN: simple example

node I::insert(K,V)

(1) a = h_x(K)
b = h_y(K)

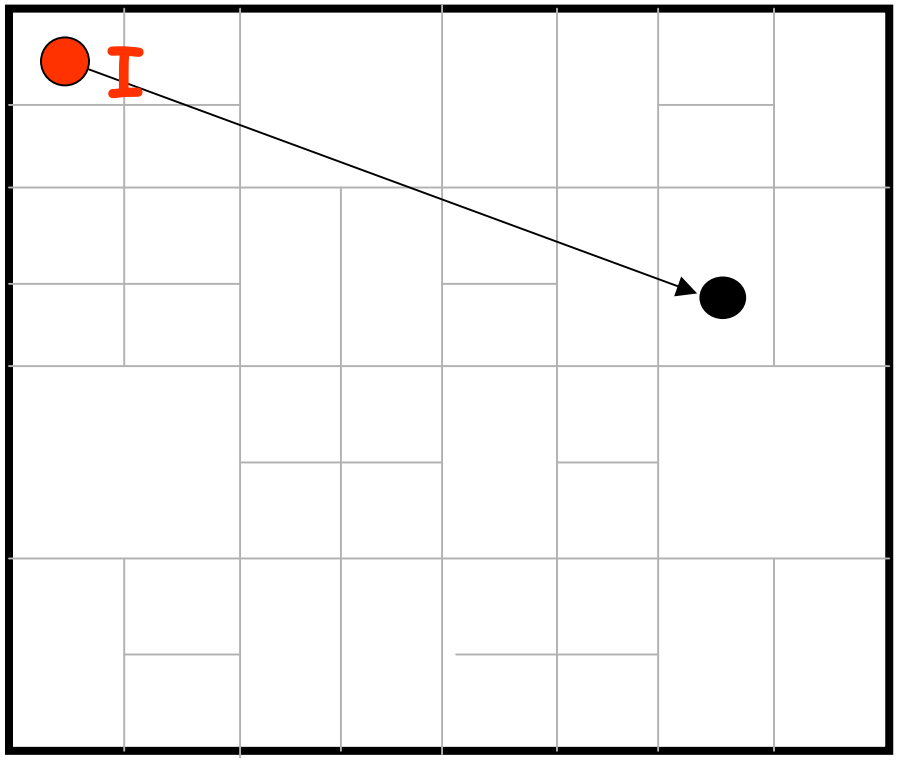


CAN: simple example

node I::insert(K,V)

(1) a = h_x(K)
b = h_y(K)

(2) route(K,V) -> (a,b)



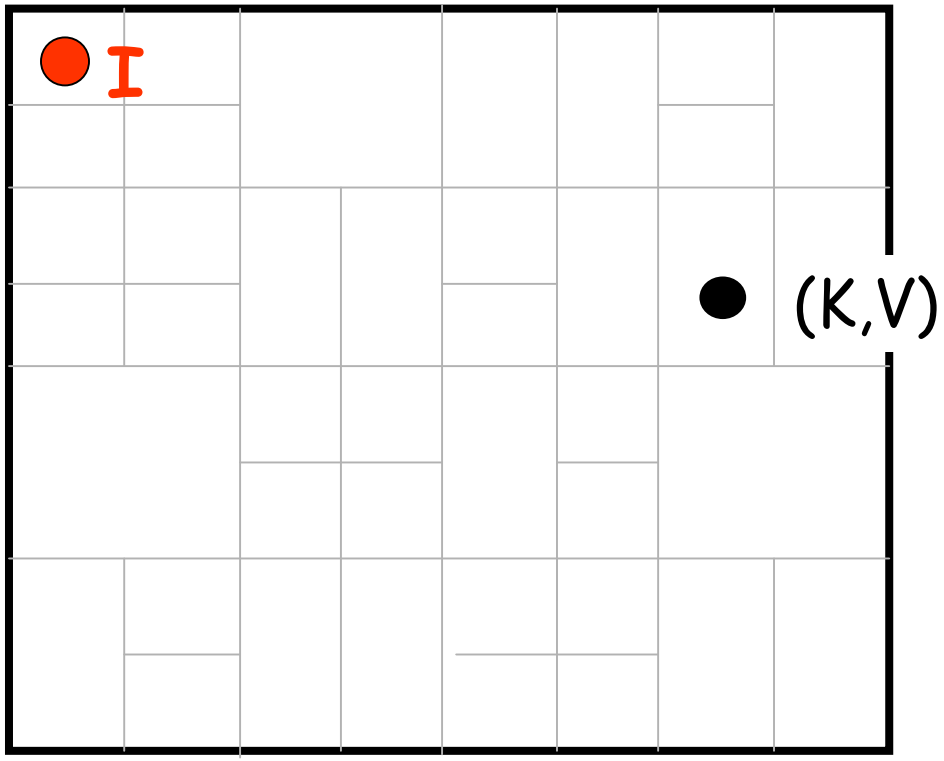
CAN: simple example

node I::insert(K,V)

(1) a = h_x(K)
b = h_y(K)

(2) route(K,V) -> (a,b)

(3) (a,b) stores (K,V)

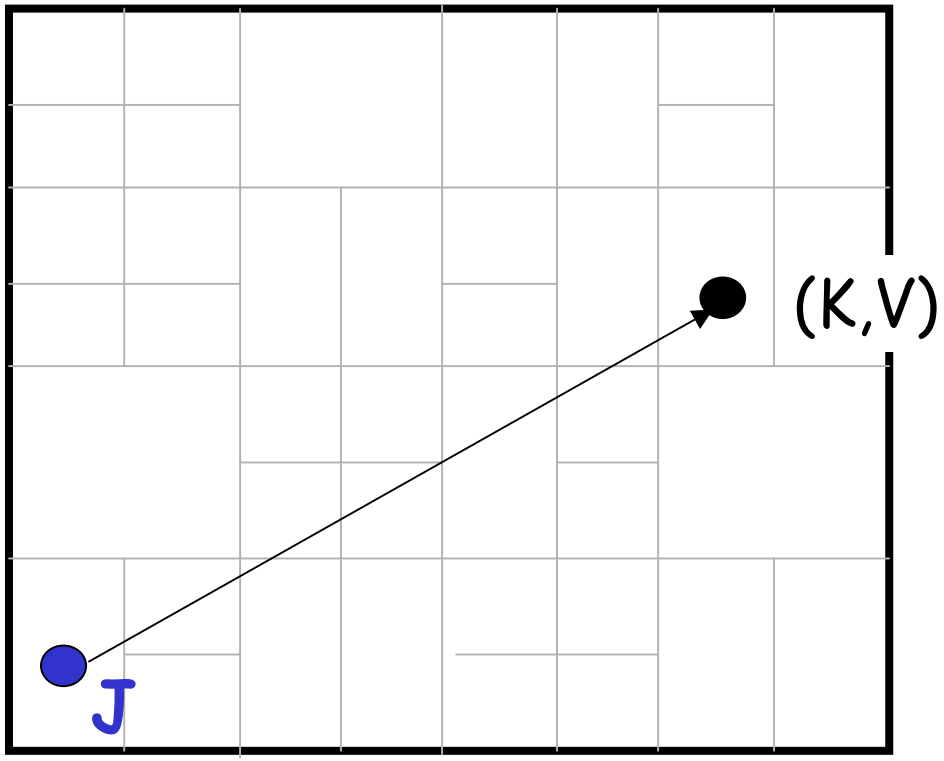


CAN: simple example

node J::retrieve(K)

(1) a = h_x(K)
b = h_y(K)

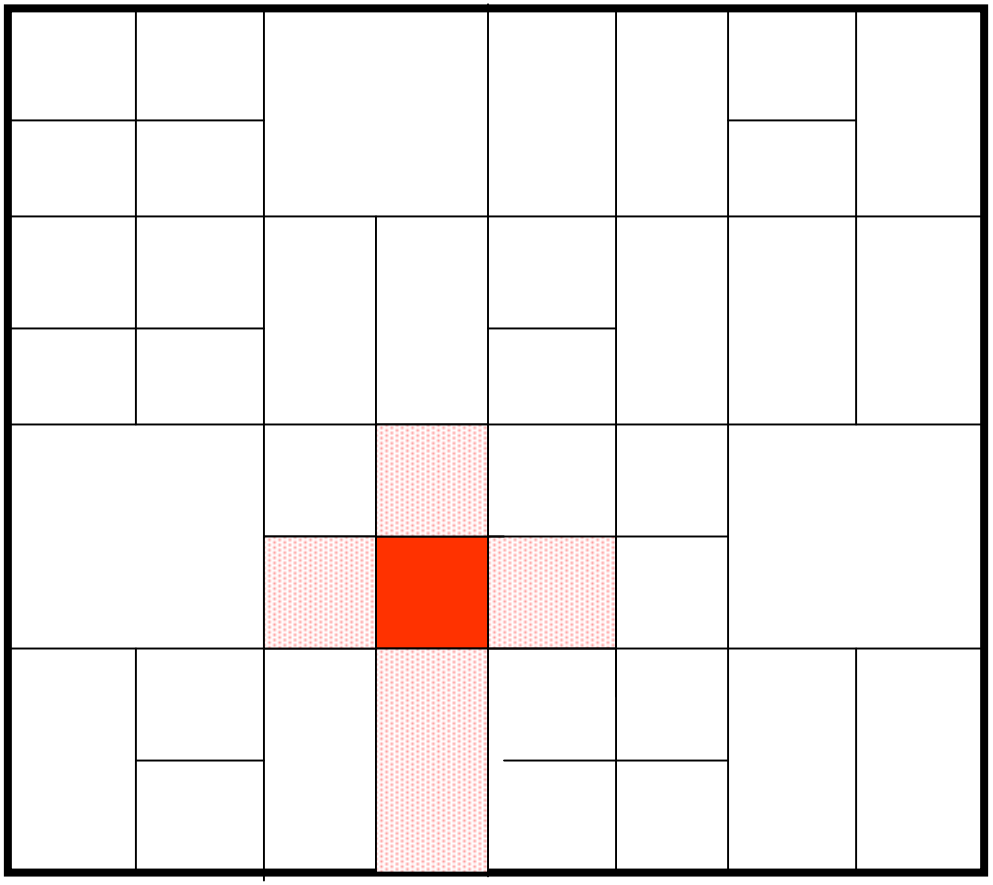
(2) route "retrieve(K)" to (a,b)



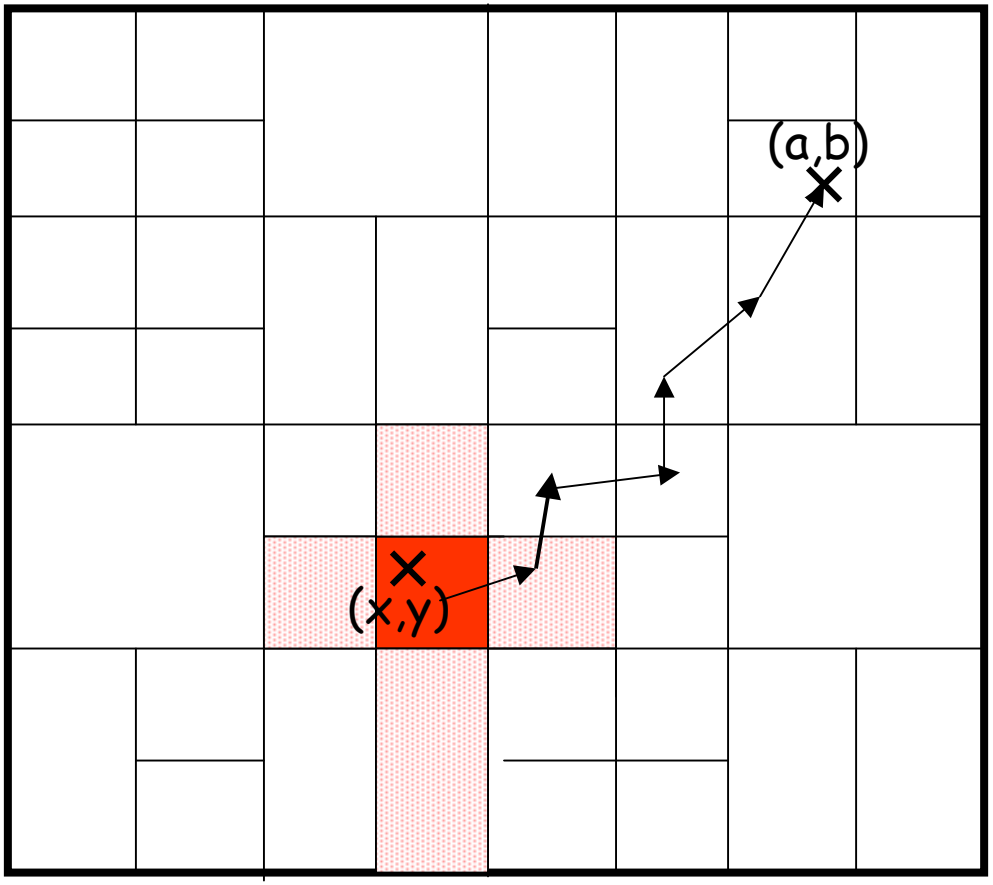
CAN

Data stored in the CAN is addressed by name (i.e. key), not location (i.e. IP address)

CAN: routing table



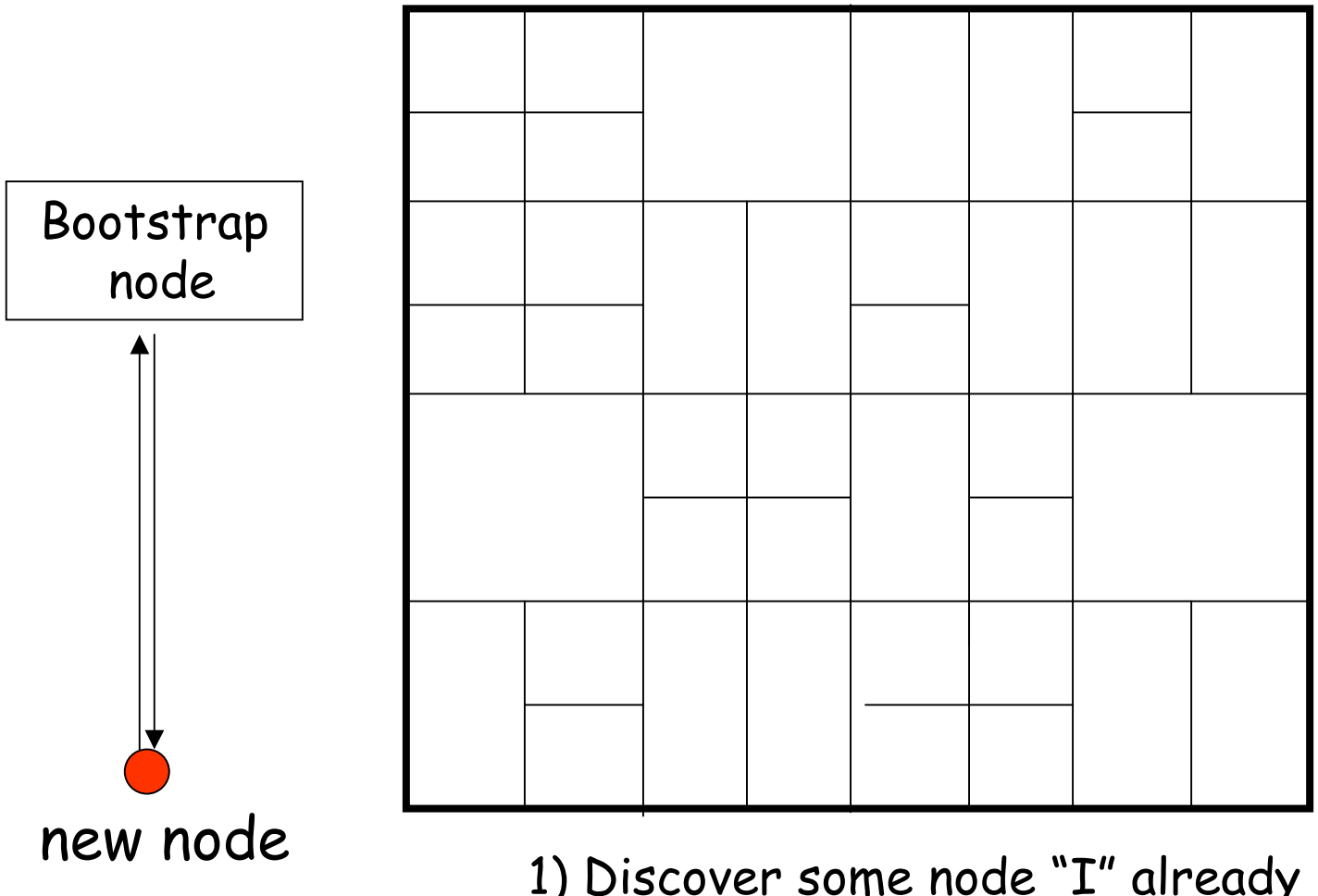
CAN: routing



CAN: routing

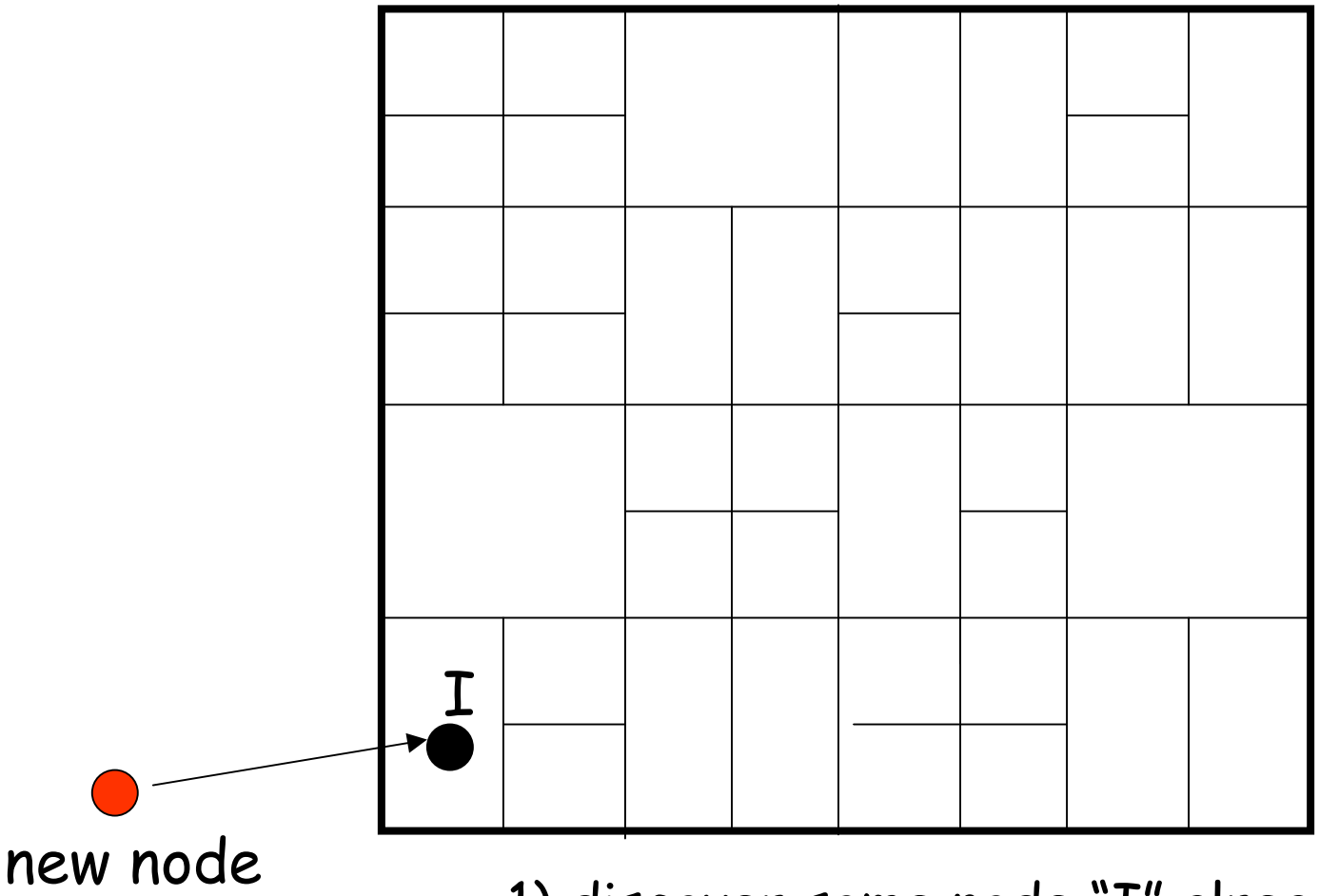
A node only maintains state for its immediate neighboring nodes

CAN: node insertion



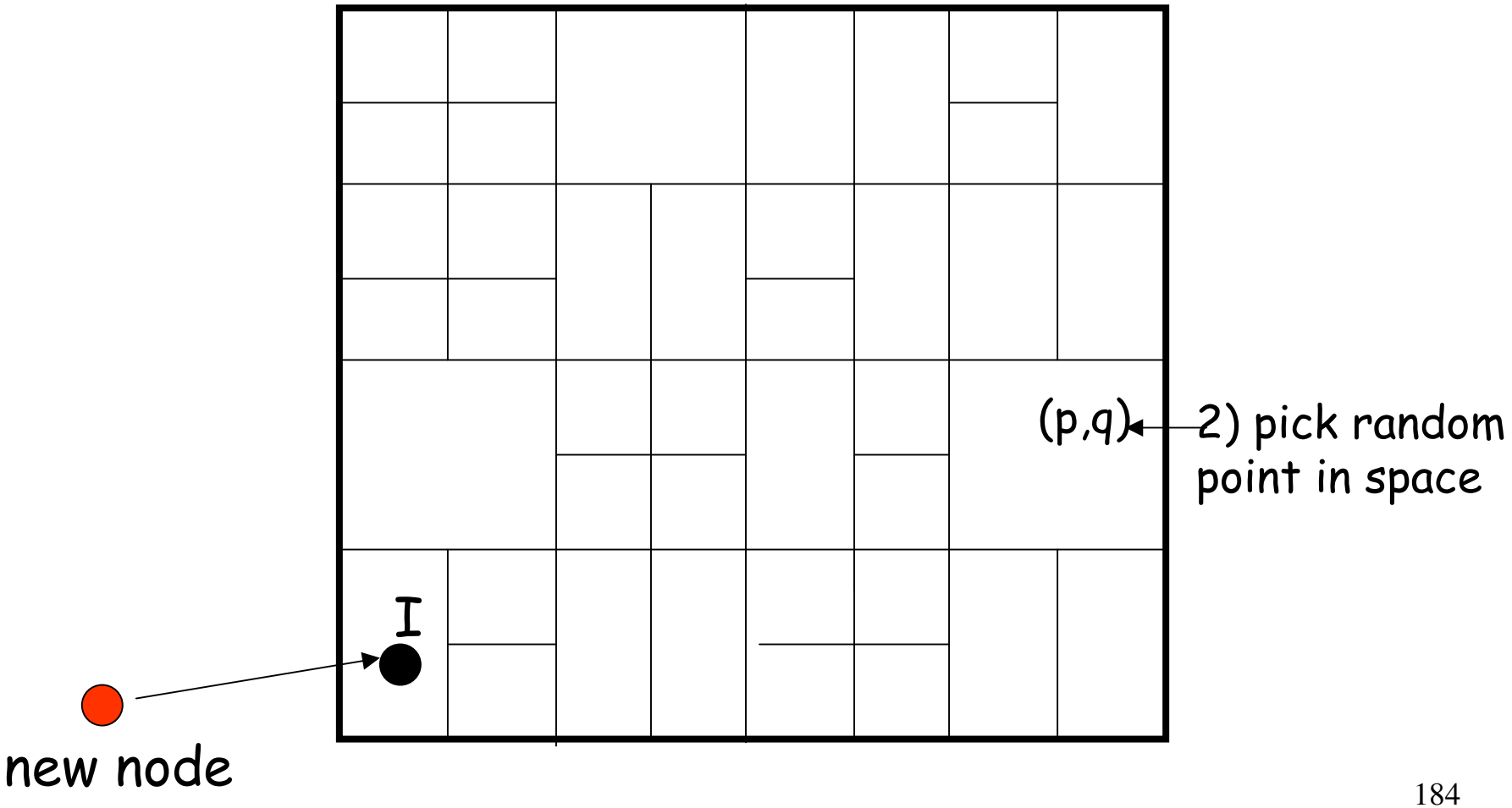
1) Discover some node "I" already in CAN

CAN: node insertion

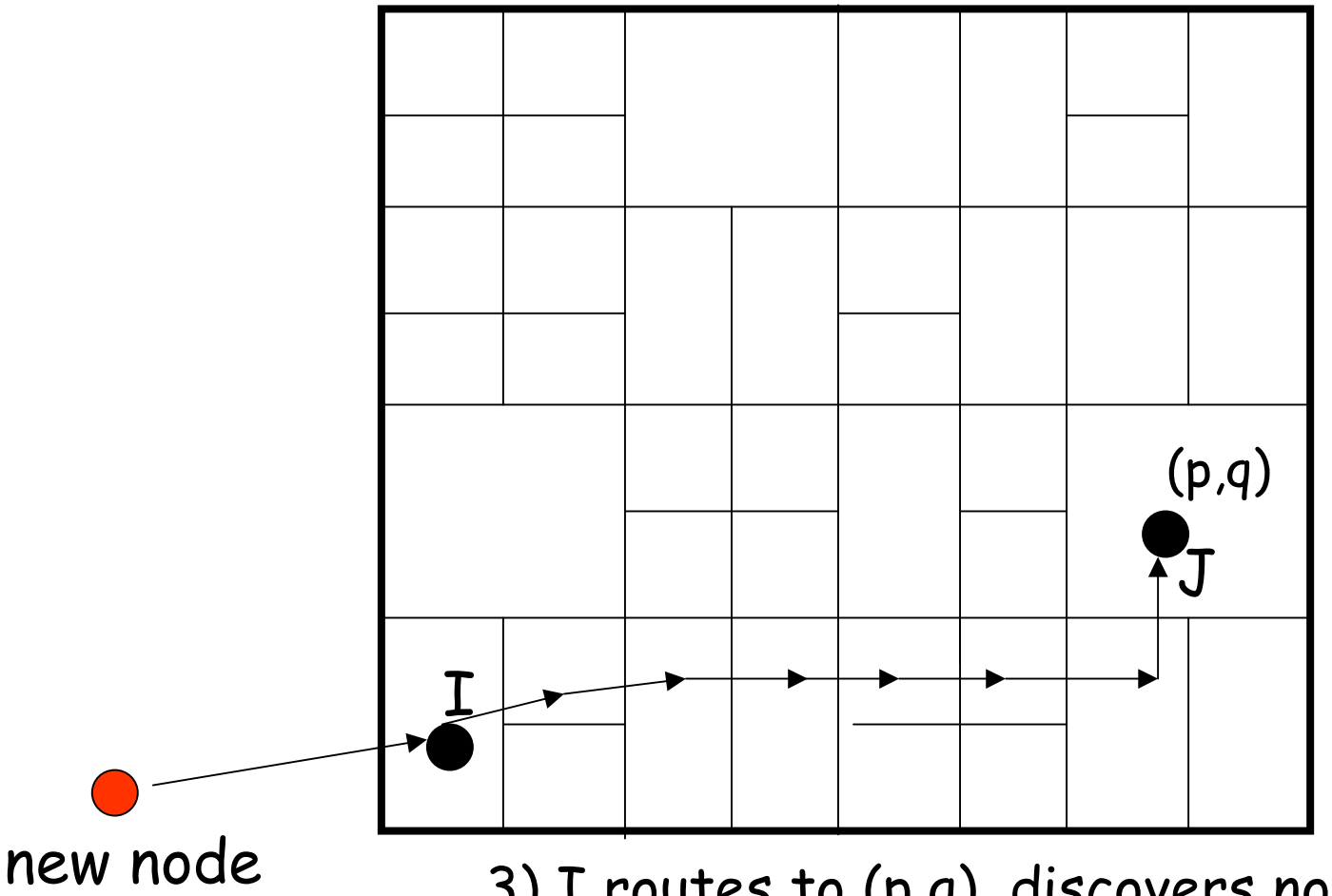


1) discover some node "I" already in CAN

CAN: node insertion

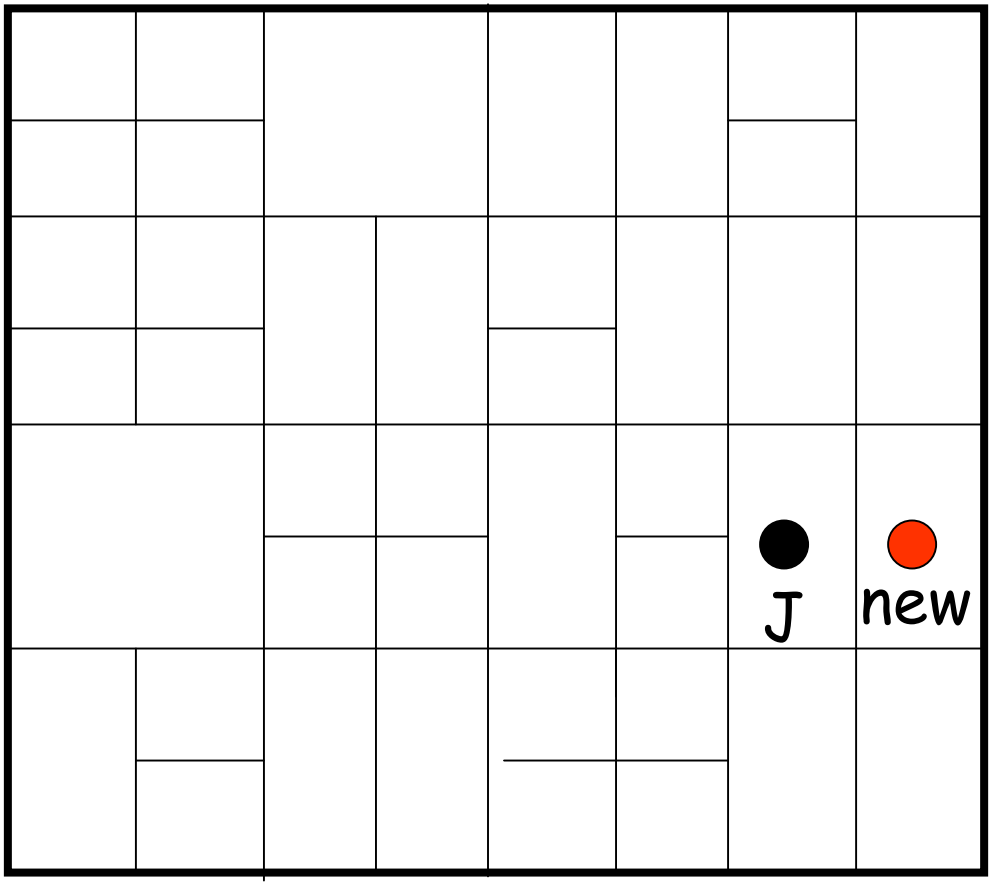


CAN: node insertion



3) I routes to (p,q), discovers node J

CAN: node insertion



4) split J's zone in half… new owns one half

CAN: node insertion

Inserting a new node affects only a single other node and its immediate neighbors

CAN: node failures

- Need to repair the space
  - recover database (weak point)
    - soft-state updates
    - use replication, rebuild database from replicas
  - repair routing
    - takeover algorithm

CAN: takeover algorithm

- Simple failures
  - know your neighbor's neighbors
  - when a node fails, one of its neighbors takes over its zone
- More complex failure modes
  - simultaneous failure of multiple adjacent nodes
  - scoped flooding to discover neighbors
  - hopefully, a rare event

CAN: node failures

Only the failed node's immediate neighbors are required for recovery

Design recap

- Basic CAN
  - completely distributed
  - self-organizing
  - nodes only maintain state for their immediate neighbors
- Additional design features
  - multiple, independent spaces (realities)
  - background load balancing algorithm
  - simple heuristics to improve performance

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Evaluation

- Scalability
- Low-latency
- Load balancing
- Robustness

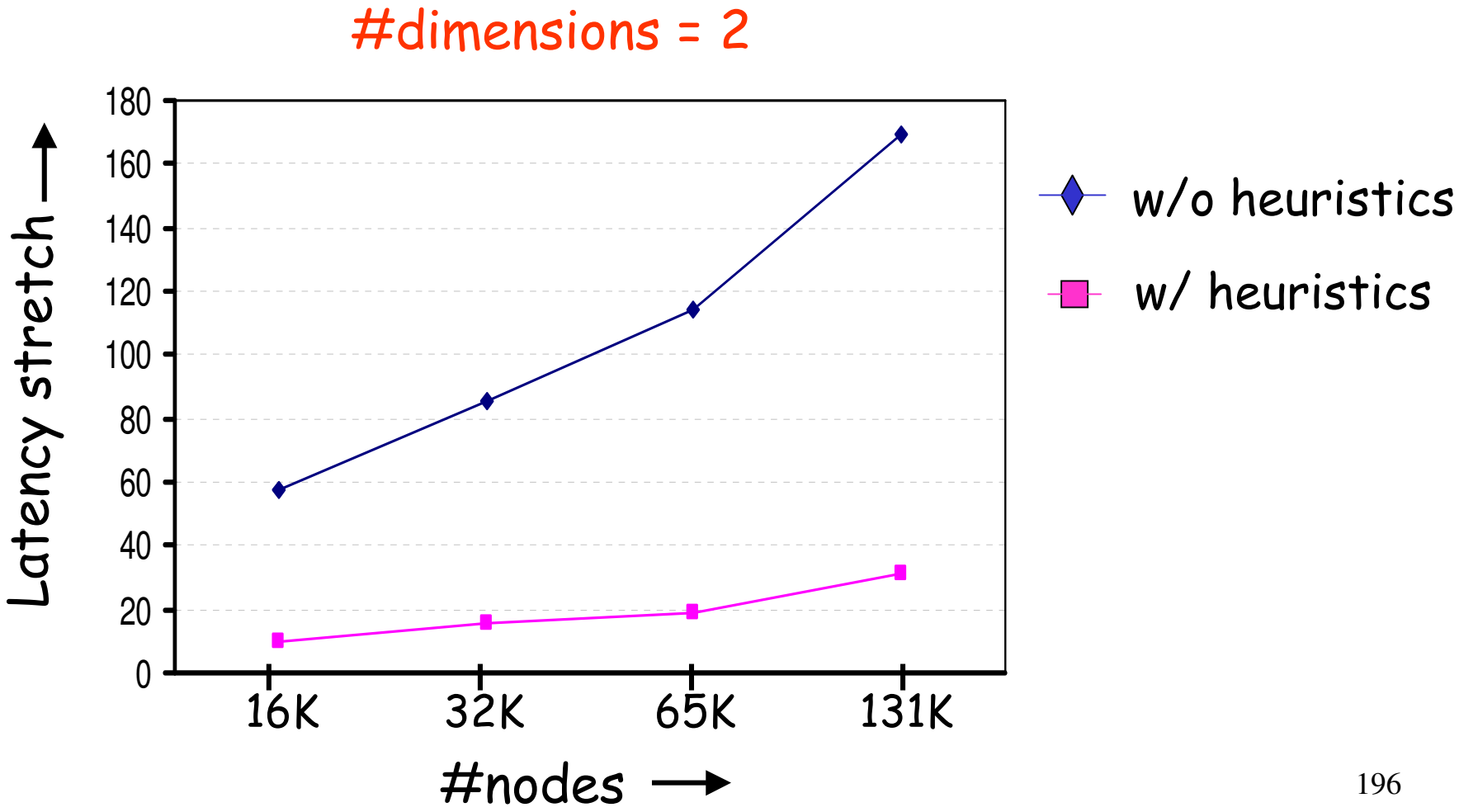
CAN: scalability

- For a uniformly partitioned space with **n** nodes and **d** dimensions
  - per node, number of neighbors is 2d
  - average routing path is $(dn^{1/d})/4$ hops
  - simulations show that the above results hold in practice
- Can scale the network without increasing per-node state
- Chord/Plaxton/Tapestry/Buzz
  - log(n) nbrs with log(n) hops

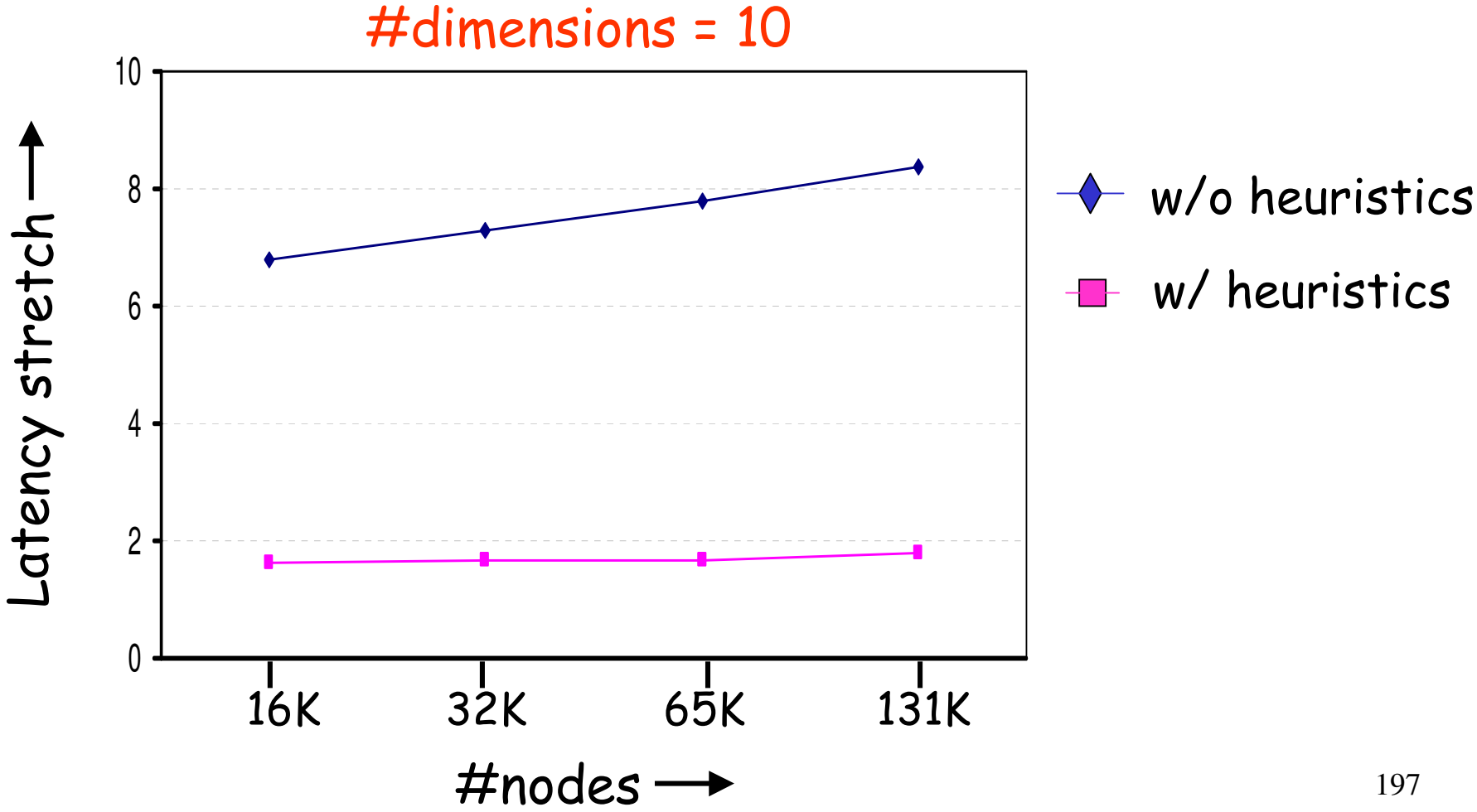
CAN: low-latency

- Problem
  - latency stretch = $\frac{\text{(CAN routing delay)}}{\text{(IP routing delay)}}$
  - application-level routing may lead to high stretch
- Solution
  - increase dimensions, realities (reduce the path length)
  - Heuristics (reduce the per-CAN-hop latency)
    - RTT-weighted routing
    - multiple nodes per zone (peer nodes)
    - deterministically replicate entries

CAN: low-latency



CAN: low-latency



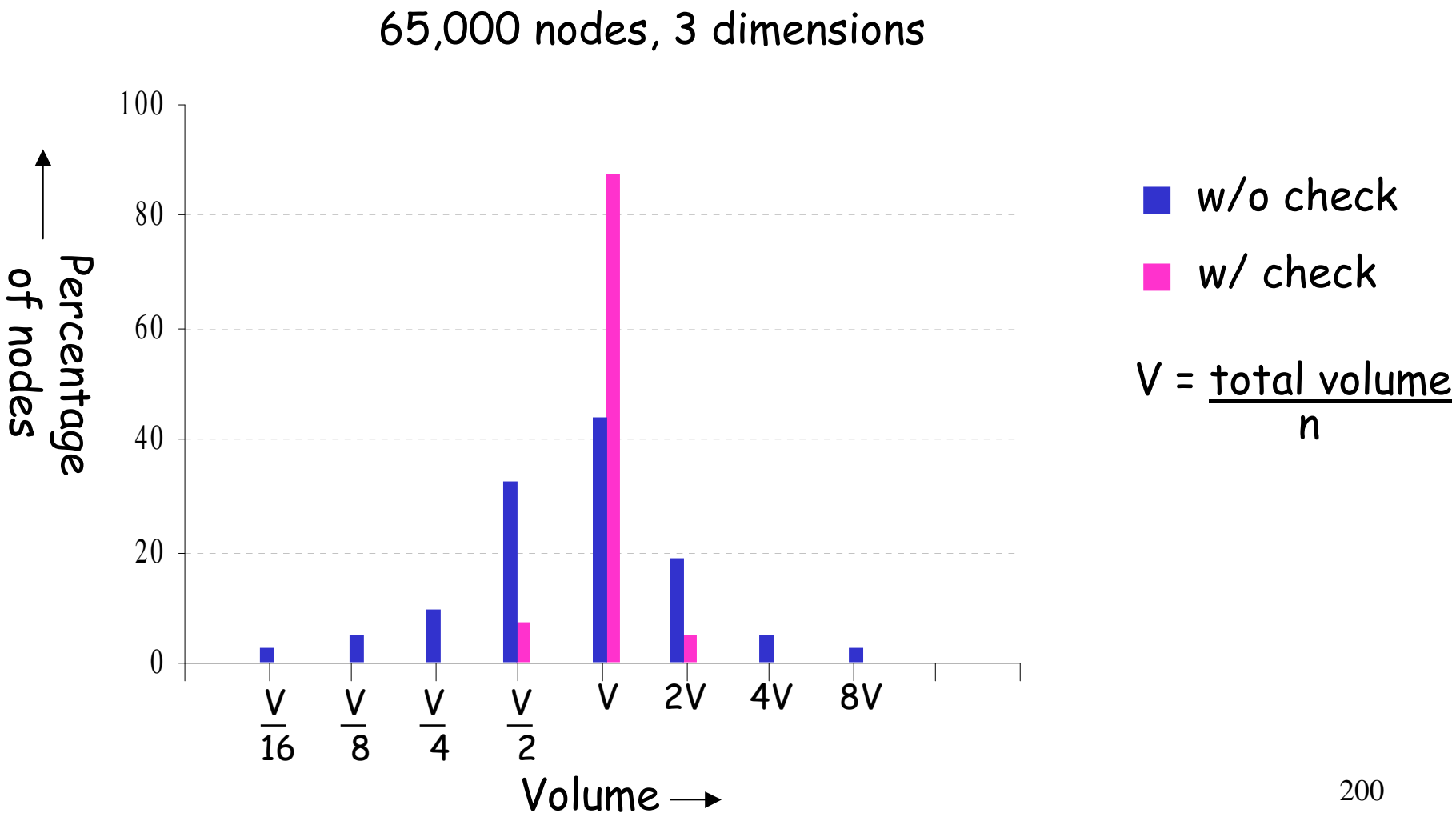
CAN: load balancing

- Two pieces
  - Dealing with hot-spots
    - popular (key,value) pairs
    - nodes cache recently requested entries
    - overloaded node replicates popular entries at neighbors
  - Uniform coordinate space partitioning
    - uniformly spread (key,value) entries
    - uniformly spread out routing load

Uniform Partitioning

- Added check
  - at join time, pick a zone
  - check neighboring zones
  - pick the largest zone and split that one

Uniform Partitioning



CAN: Robustness

- Completely distributed
  - no single point of failure (not applicable to pieces of database when node failure happens)
- Not exploring database recovery (in case there are multiple copies of database)
- Resilience of routing
  - can route around trouble

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Strengths

- More resilient than flooding broadcast networks
- Efficient at locating information
- Fault tolerant routing
- Node & Data High Availability (w/ improvement)
- Manageable routing table size & network traffic

Weaknesses

- Impossible to perform a fuzzy search
- Susceptible to malicious activity
- Maintain coherence of all the indexed data (Network overhead, Efficient distribution)
- Still relatively higher routing latency
- Poor performance w/o improvement

Suggestions

- Catalog and Meta indexes to perform search function
- Extension to handle mutable content efficiently for web-hosting
- Security mechanism to defense against attacks

Outline

- Introduction
- Design
- Evaluation
- Strengths & Weaknesses
- Ongoing Work

Ongoing Work

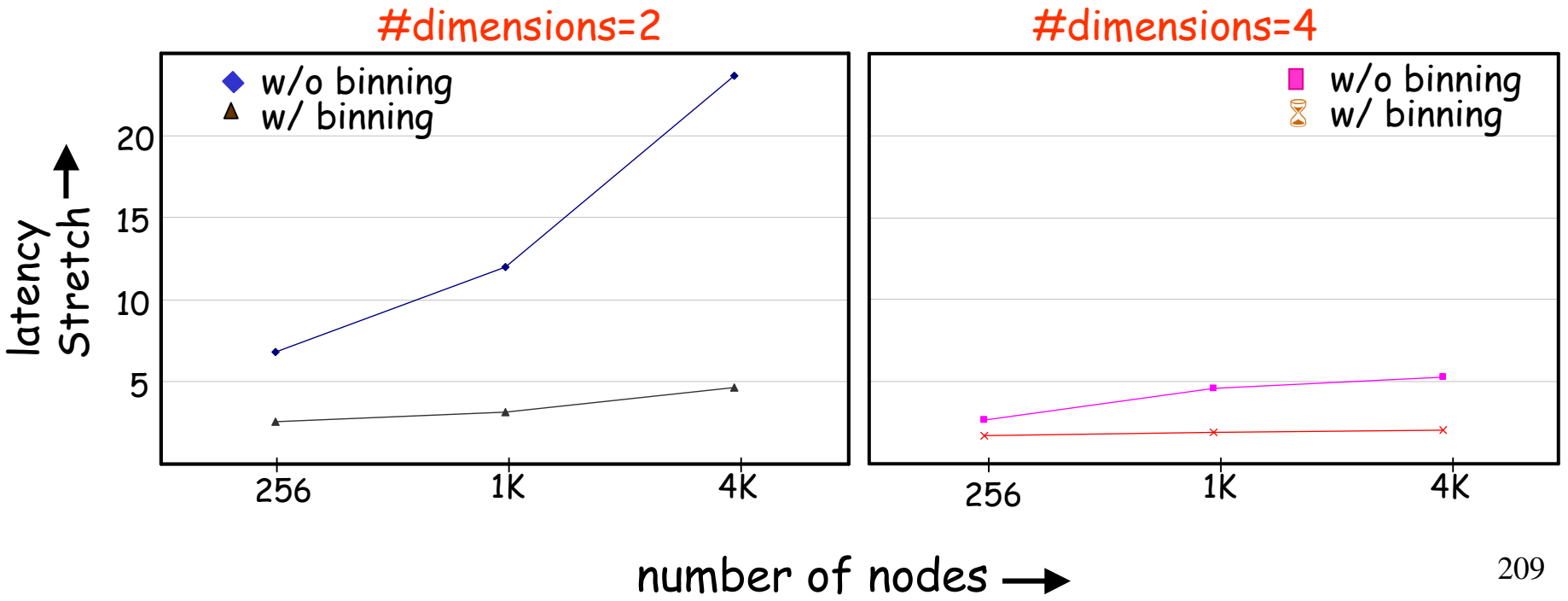
- Topologically-sensitive CAN construction
  - distributed binning

Distributed Binning

- Goal
  - bin nodes such that co-located nodes land in same bin
- Idea
  - well known set of landmark machines
  - each CAN node, measures its RTT to each landmark
  - orders the landmarks in order of increasing RTT
- CAN construction
  - place nodes from the same bin close together on the CAN

Distributed Binning

- 4 Landmarks (placed at 5 hops away from each other)
- naïve partitioning



Ongoing Work (cont'd)

- Topologically-sensitive CAN construction
  - distributed binning

- CAN Security (Petros Maniatis - Stanford)
  - spectrum of attacks
  - appropriate counter-measures

Ongoing Work (cont'd)

- CAN Usage
  - Application-level Multicast (NGC 2001)
  - Grass-Roots Content Distribution
  - Distributed Databases using CANs
    (J.Hellerstein, S.Ratnasamy, S.Shenker, I.Stoica, S.Zhuang)

Summary

- CAN
  - an Internet-scale hash table
  - potential building block in Internet applications
- Scalability
  - O(d) per-node state
- Low-latency routing
  - simple heuristics help a lot
- Robust
  - decentralized, can route around trouble

Lecture 5:Messages,Anycast,Multicast

- What can we do to provide general middleware functions (not just file sharing, indexing, lookup and query routing) –
- I.e. can use p2p for generic middleware services for distributed computing
- Messages, anycast and group communication services….

Project
JXTA

9.Sun's Project JXTA

Technical Overview

Project JXTA entation Outline

- Introduction - what is JXTA
- Goal - what JXTA wants to be
- Technology - what JXTA relies upon
- Structure - how JXTA is built
- Protocols - what protocols JXTA has
- Security - whether JXTA is secure
- Applications - what JXTA can be used for
- Collaboration - how JXTA grows

Project JXTAduction

- "JXTA" - pronounced as "**juxta**" as in "**juxtaposition**"
- started by Sun's Chief Scientist **Bill Joy**
- an effort to create a **common platform** for building distributed services and applications
- Napster, Gnutella, and Freenet provide users with **limited ability** to share resources and are unable to share data with other, similar applications

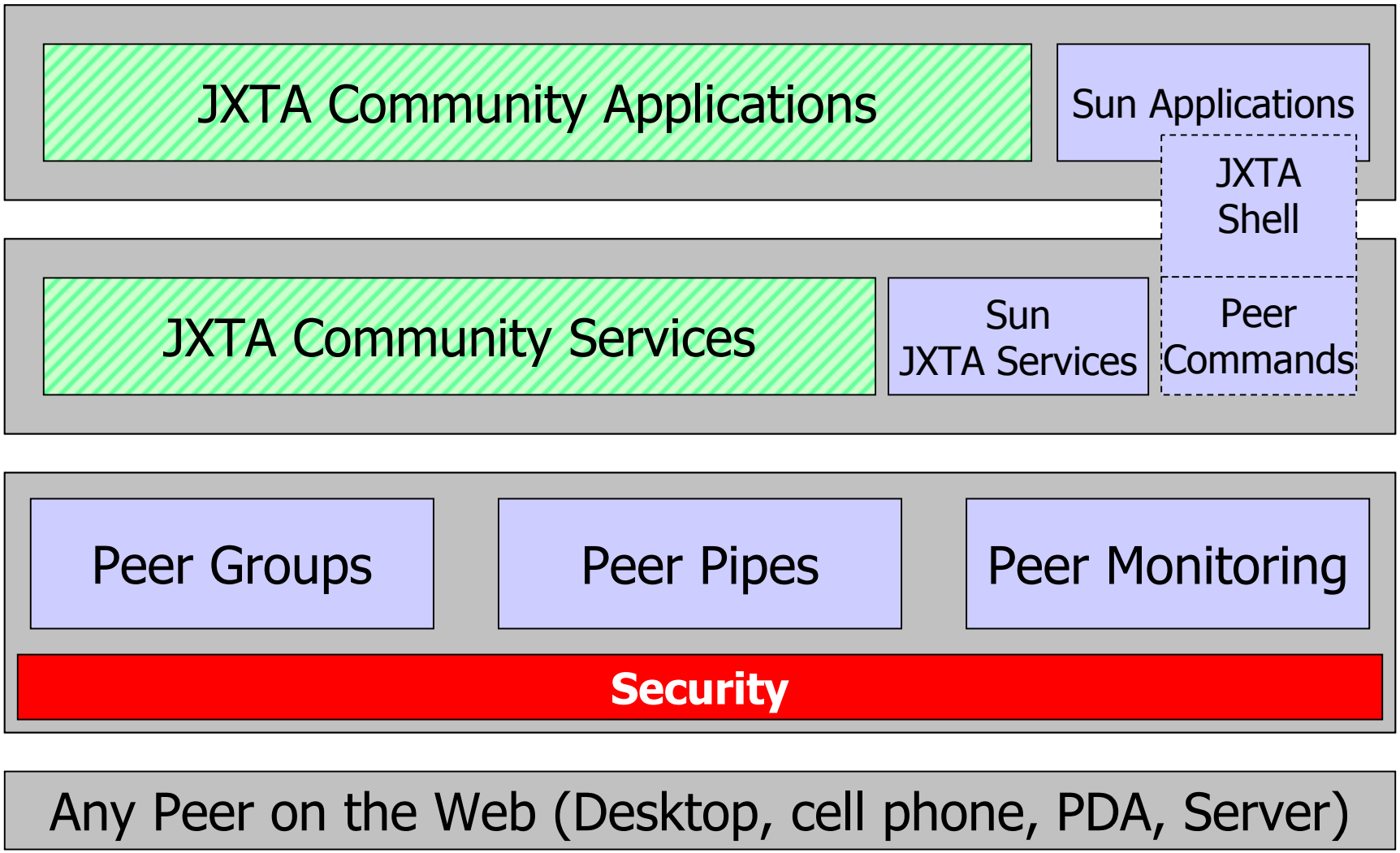
Project JXTA's Purpose

- enable a wide range of distributed computing applications by developing a common set of general purpose P2P protocols
- achieve platform independence - any language, any OS, any hardware
- overcome the limitations found in many today's P2P applications
- enable new applications to run on any device that has a digital heartbeat (desktop computers, servers, PDAs, cell phones, and other connected devices)

Project JXTAnology

- JXTA technology is based on XML, Java technology, and key concepts of UNIX operating system
- Transmitted information is packaged as messages. Messages define an XML envelop to transfer any kind of data.
- The use of Java language is not required - JXTA protocols can be implemented in C, C++, Perl, or any other programming language

Project JXTA:ture



Project JXTA Multi-Layered Structure

- **JXTA Core**
  - **Peer Groups** - mechanisms to/for create and delete, join, advertise, discover, communication, security, content sharing
  - **Peer Pipes** - transfer of data, content, and code in a protocol-independent manner
  - **Peer Monitoring** - including access control, priority setting, traffic metering and bandwidth balancing
- **JXTA Services**
  - expand upon the capabilities of the core and facilitate application development
  - mechanisms for searching, sharing, indexing, and caching code and content to enable cross-application bridging and translation of files
- **JXTA Shell** - much like UNIX OS
  - facilitate access to core-level functions through a command line
- **JXTA Applications** - built using peer services as well as the core layer

Project JXTAcols

- JXTA is a set of six protocols
- Peer Discovery Protocol - find peers, groups, advertisements
- Peer Resolver Protocol - send/receive search queries
- Peer Information Protocol - learn peers' status/properties
- Peer Membership Protocol - sign in, sign out, authentication
- Pipe Binding Protocol - pipe advertisement to pipe endpoint
- Endpoint Routing Protocol - available routes to destination

Project JXTArity

- Confidentiality, integrity, availability - authentication, access control, encryption, secure communication, etc.
- Developing more concrete and precise security architecture is an ongoing project
- JXTA does not mandate certain security polices, encryption algorithms or particular implementations
- JXTA 1.0 provides Security Primitives:
  - crypto library (MD5, RC4, RSA, etc.)
  - Pluggable Authentication Module (PAM)
  - password-based login
  - transport security mechanism modeled after SSL/TLS

Project JXTA Potential Applications

- Search the entire web and all its connected devices (not just servers) for needed information
- Save files and information to distributed locations on the network
- Connect game systems so that multiple people in multiple locations
- Participate in auctions among selected groups of individuals
- Collaborate on projects from anywhere using any connected device
- Share compute services, such as processor cycles or storage systems, regardless of where the systems or the users are located

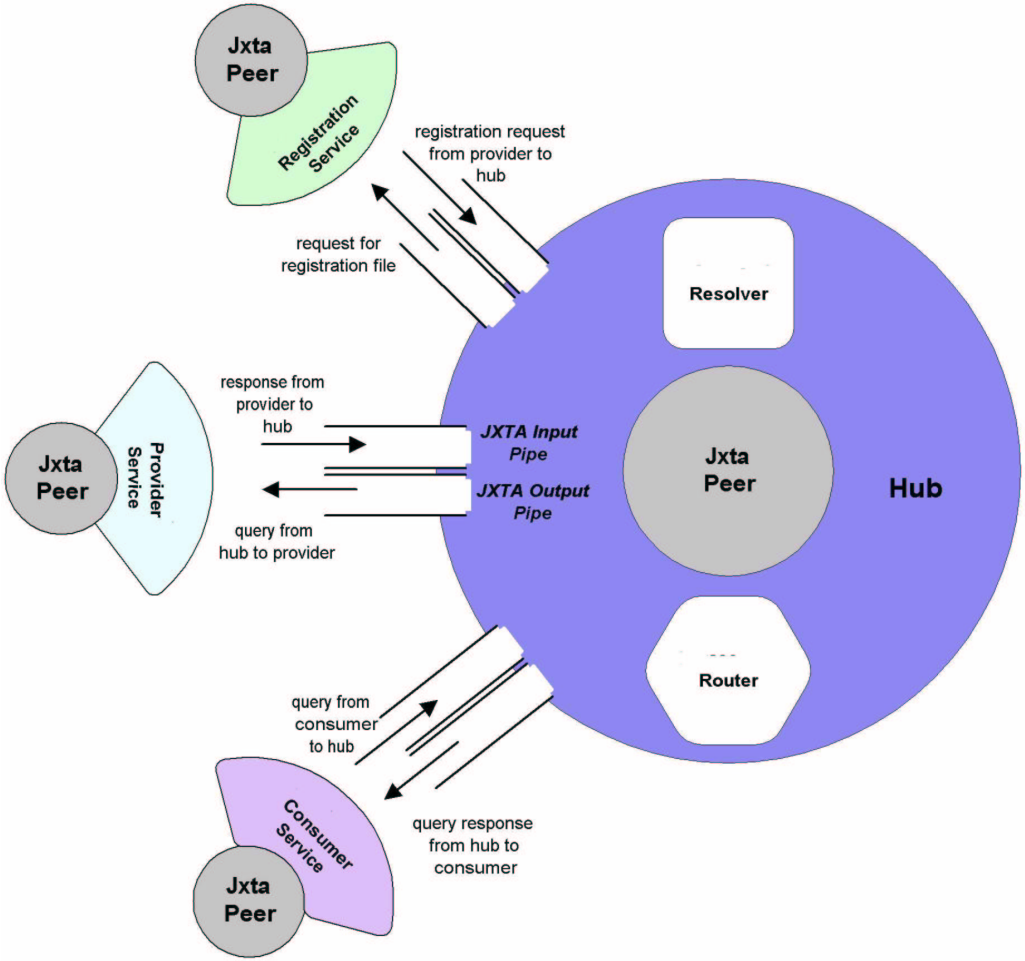
Project JXTA - A Search Overview

- Started in June 2000 by Infrasearch as an idea to distribute queries to network peers best capable of answering them
- Now it is the default searching methodology for the JXTA framework in the form of JXTA Search
- Communication via an XML protocol called Query Routing Protocol (QRP)
- Network components: Providers, Consumers, Hubs
- Capable of providing both wide and deep search; deep search shows the most benefits
- Design goals: Simplicity, Structure, Extensibility, Scalability

Project JXTA JXTA Search Benefits

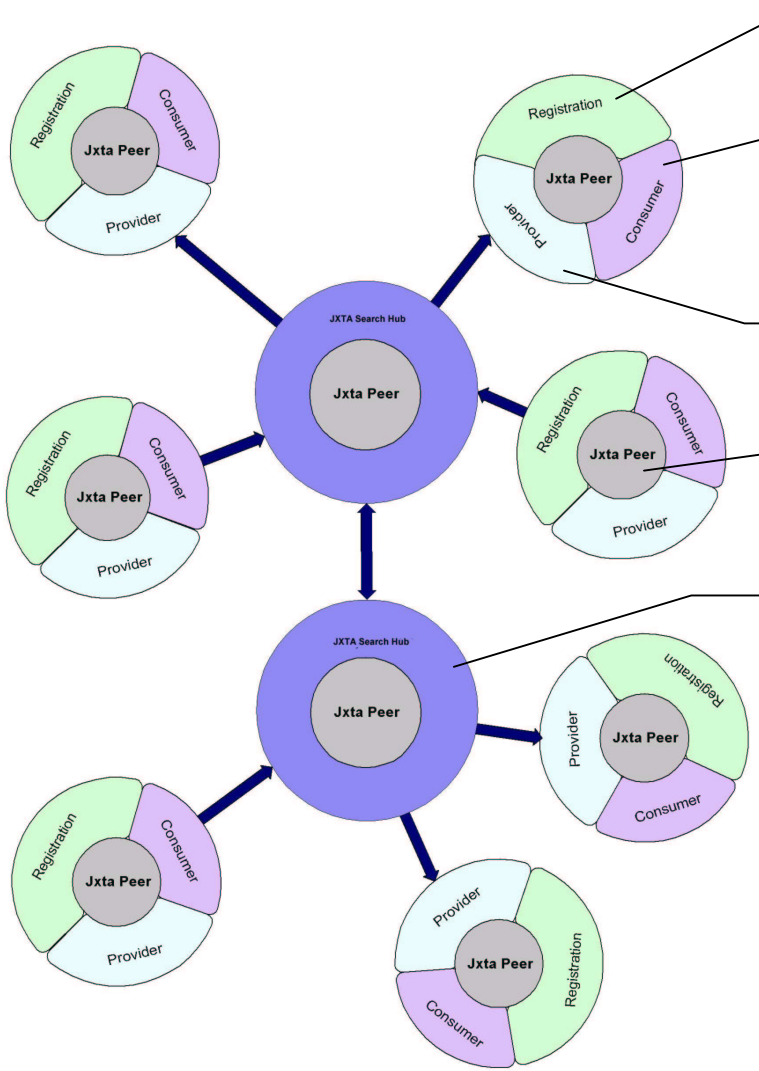
- Speed of update - especially noticeable in deep search, where large data in databases are accessed directly without a need to create a central index.
- Access - in crawling based approach many companies are resilient to grant access to web crawlers. In distributed approach the companies can serve the data as they feel appropriate.
- Efficiency - no need to create a centrally placed and maintained index for the whole web.

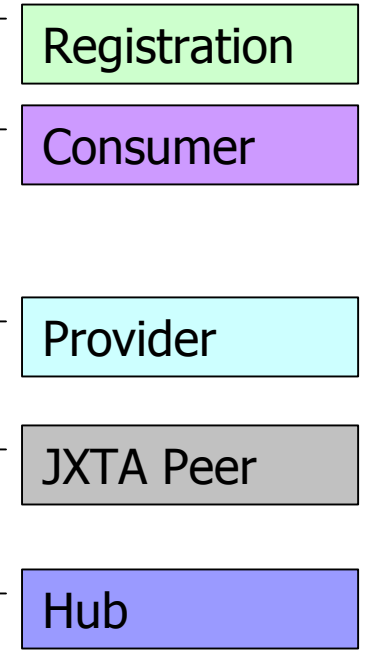
JXTA Search Architecture



- Each JXTA peer can run instances of Provider, Consumer, and Registration services on top of its JXTA core.
- Each peer interacts with the JXTA Search Hub Services, which is also running on top of the JXTA core.

Project JXTA JXTA Search Architecture





- Each peer within the network interacts with the hub using its appropriate service

Project JXTAollaboration

- Currently over 25 companies are participating in developing JXTA projects.
- Core (7 projects)
  - security, juxta-c, juxtaperl, pocketjxta
- Services (20 projects)
  - search, juxtaspaces, p2p-email, juxta-grid, payment, monitoring
- Applications (12 projects)
  - shell, jnushare, dfwbase, brando
- Other projects (5) - demos, tutorials, etc.

Project JXTA

Future Work, Questions

- Future Work
  - C/C++ implementation
  - KVM based implementation (PDAs, cell phones)
  - Naming and binding services
  - Security services (authentication, access control)
  - Solutions for firewalls and NAT gateways
- Is this the right structure?
- Do JXTA protocols dictate too much or too little?

10. Application Layer Anycasting: A Server Selection Architecture and Use in Replicated Web Service

Ellen W. Zegura
Mostafa H. Amamr
Zongming Fei
Networking and Telecommunications Group
Georgia Tech, Atlanta, GA

Samrat Battacharjee
Department of Computer Science
University of Maryland, College Park, MD

Agenda

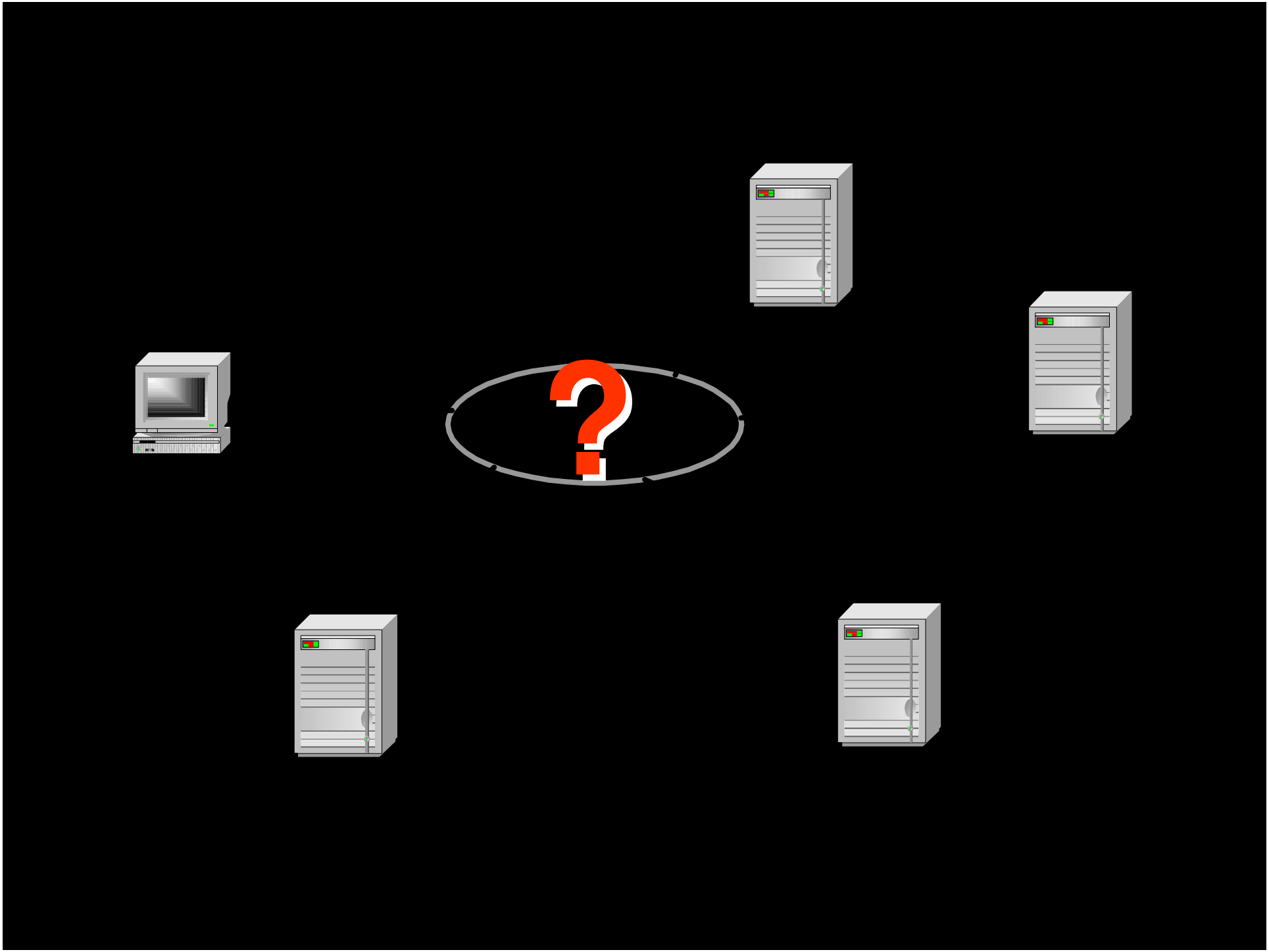
- Problem Statement
- The Anycasting Communication Paradigm
- Some Related Work
- Application Layer Anycasting
- Experimental Results
- Conclusions

Problem Statement

- Efficient service provision in wide area networks
- Replicated services
- Applications want access to the best server
- Best may depend on time, performance, policy

Server Replication

- Standard technique to improve scalability of a service
- Issues in server replication
  - Location of servers
  - Consistency across servers
  - Server selection
- Server selection problem
  - How does a client determine which of the replicated servers to access ?

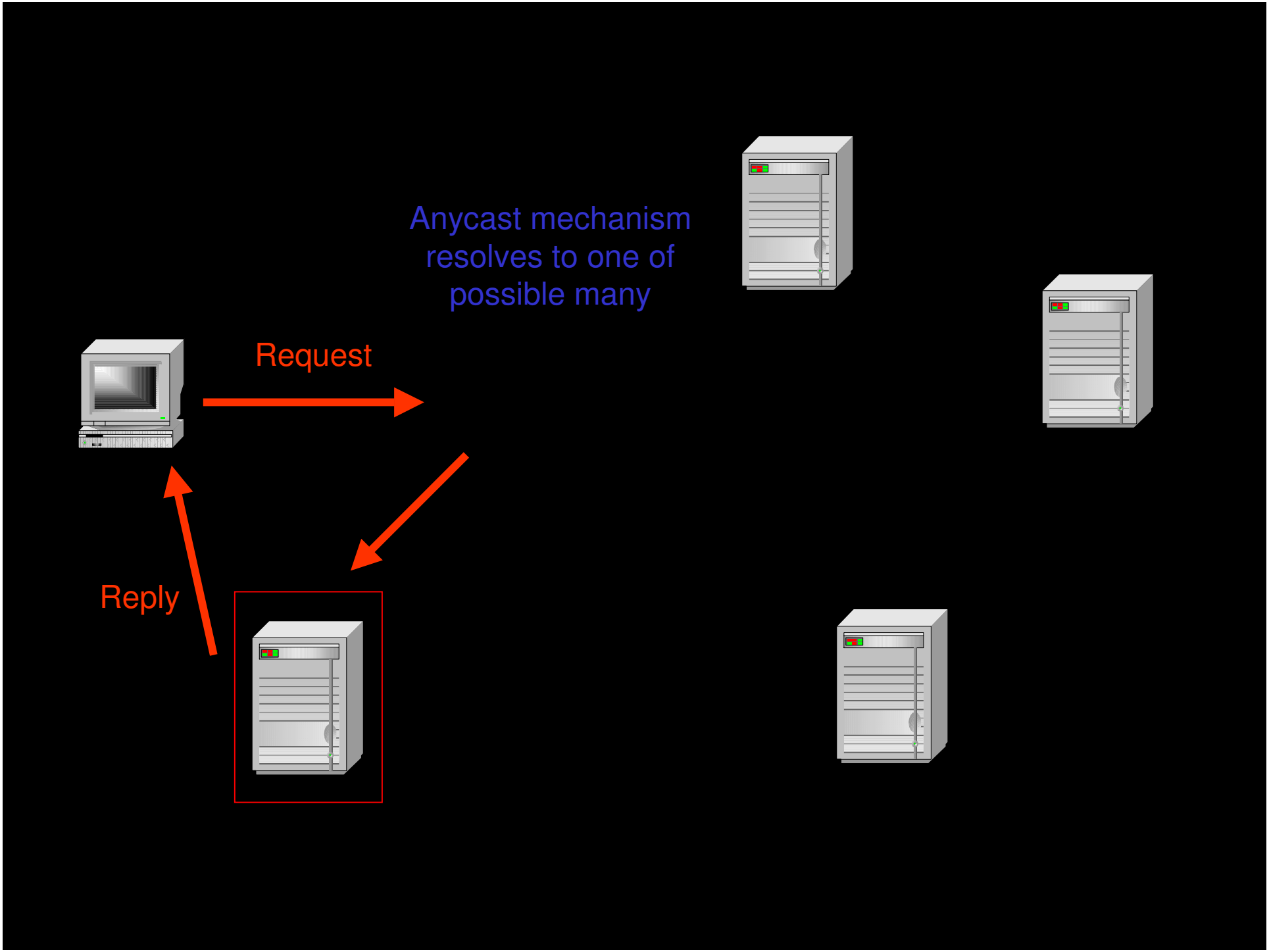


Server Selection

- Alternatives
  - Designated (e.g. nearest) server
  - Round robin assignment (e.g. DNS rotator)
  - Explicit list with user selection
  - Selection architecture (e.g. Cisco DistributedDirector)
- Application-Layer Anycasting:
  - Client requests connection to anycast group
  - Anycast group consists of replicated (equivalent) servers
  - System connects client to any good server

Anycasting Communication Paradigm

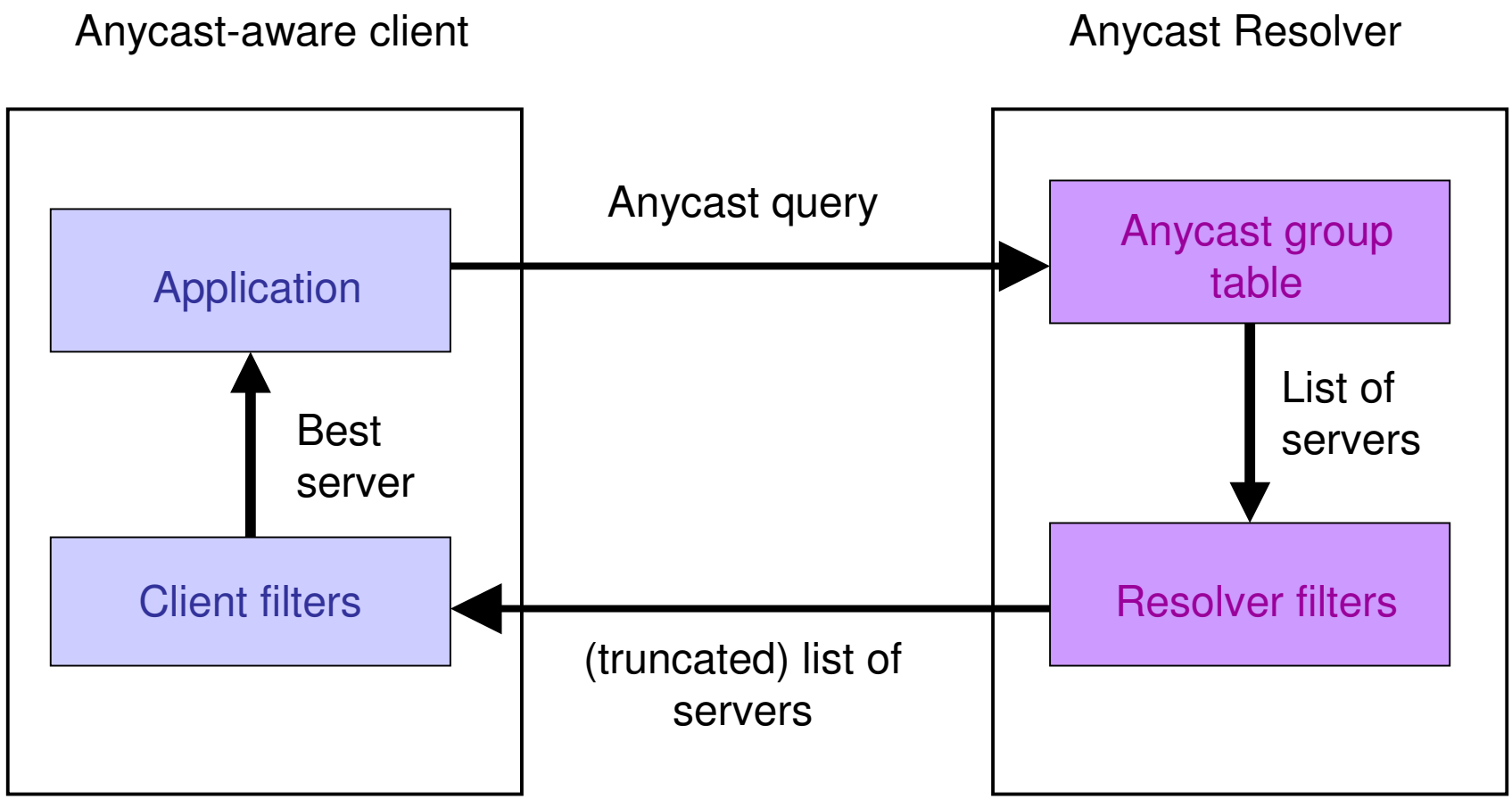
- Anycast identifier specifies a group of equivalent hosts
- Requests are sent to best host in the group


Anycast mechanism
resolves to one of
possible many
Request
Reply

Existing Anycast Solutions and Limitations

- Existing Solutions:
  - RFC 1546 Host Anycasting Service
    - Definition of anycasting communication paradigm
    - Implementation suggestions for the network layer
  - IETF Server Location Protocol – Still existing ?
  - AKAMAI and other commercial cache/CDNs
  - Cisco DistributedDirector
- Limitations
  - Global router support
  - Per diagram destination selection
  - Limited set of metrics
  - No option for user input in server selection
  - Allocation of IP address space for anycast address

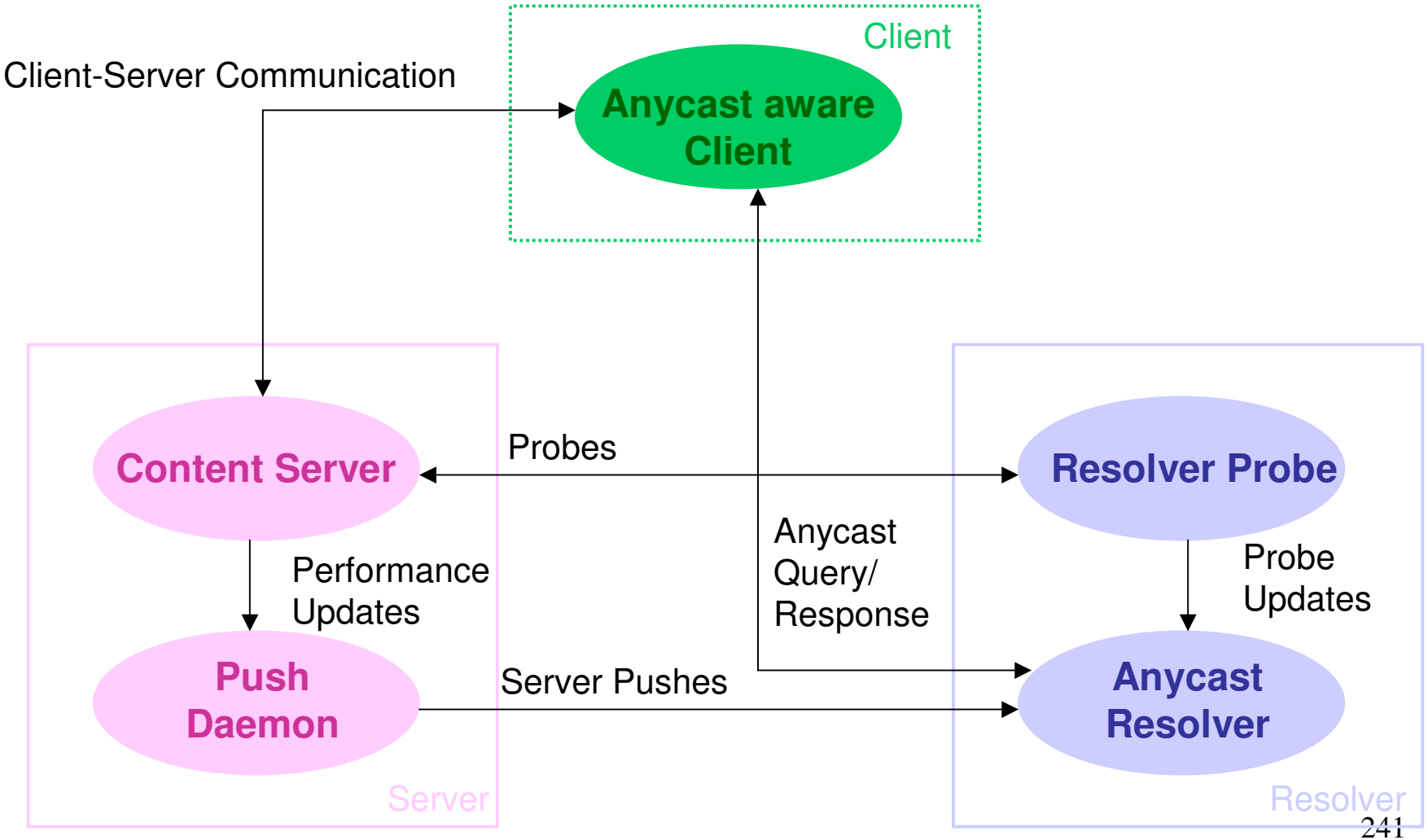
Application-Layer Anycasting



Filters

- Content-independent filters
  - E.g. Round-robin
- Metric-based filters
  - E.g. Minimum response time
- Policy-based filters
  - E.g. Minimum cost
- Filter Specification – Meteric Qualified ADN:
  - <Meteric_Service>%<Domain Name>
  - ServerLoad.Daily_News%cc.gatech.edu

Application-layer Anycasting Architecture

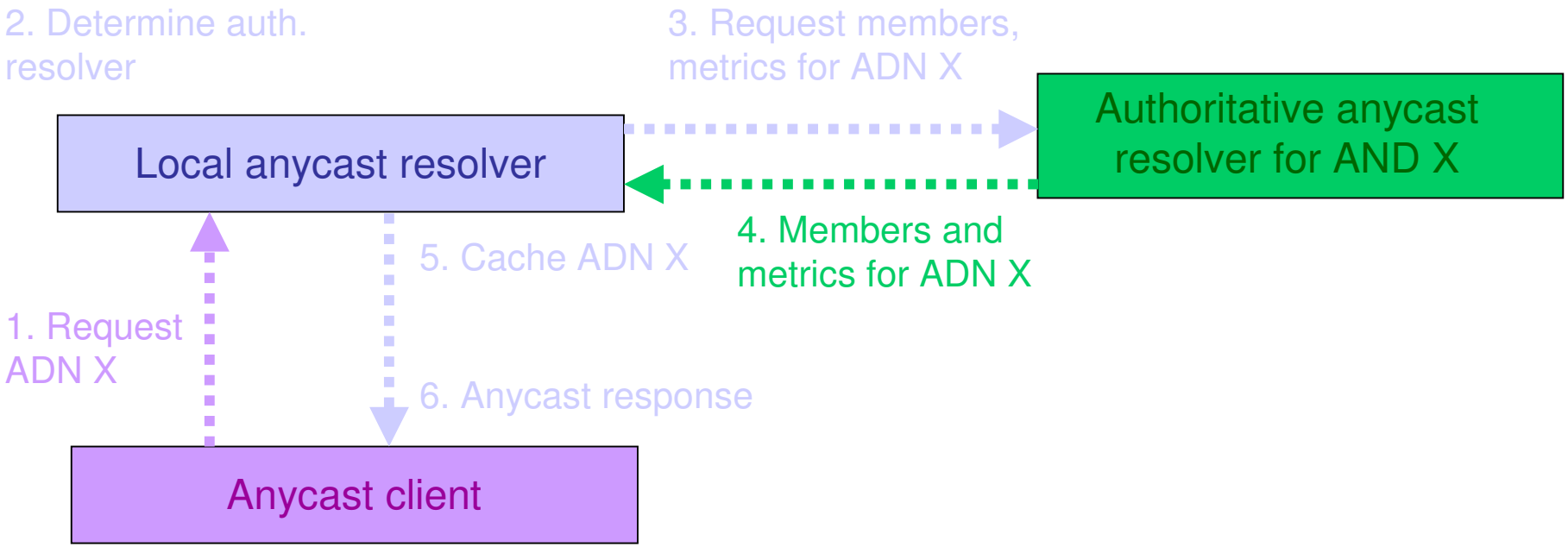


Anycast Groups

- Anycast groups consist of collection of IP addresses, domain names or aliases
- Group members provide equivalent service, e.g., mirrored FTP servers or web search engines
- Anycast groups identified by Anycast Domain Names
- Group membership an orthogonal issue architecture aliases

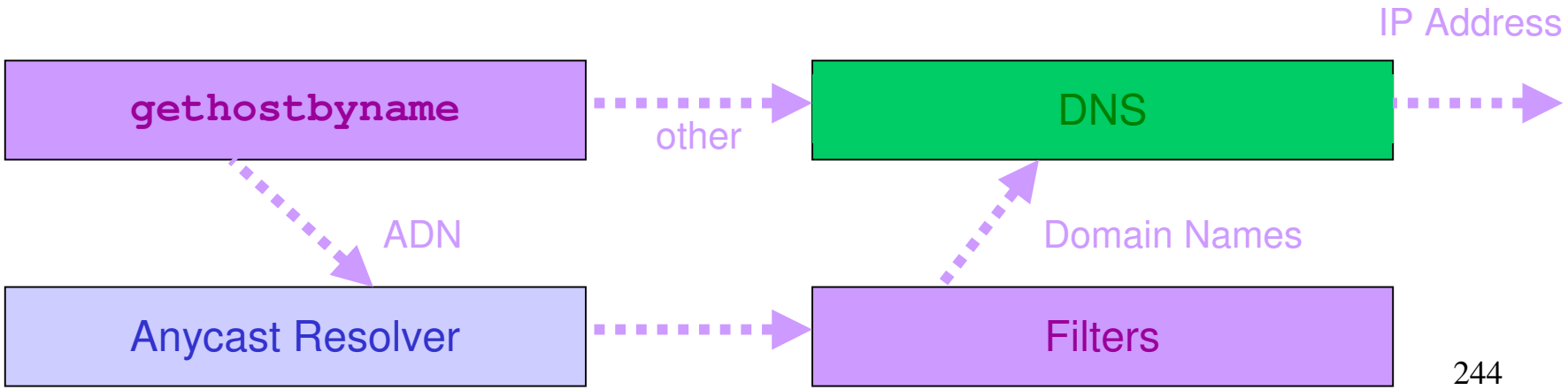
Anycast Domain Names

- Structure: <Service>%<Domain Name>
- Example:
  - Daily-News%cc.gatech.edu



Implementation

- Implementation using Metric Qualified ADNs
- Intercept calls to **`gethostbyname`**
- Transparent access to anycasting without modifying existing applications



Response Time Determination for Web Servers

- Response time:
  - Measured from time client issues request until receives last byte of file of network
  - Round trip path delays + server processing delays
- Overview of technique:
  - Resolver probes for path-dependent response time
  - Server measures and pushes path-independent processing time
  - Lighter-weight push more frequent than heavier-weght probe
  - Probe result used to calibrate pushed value

Performance Metric Determination

- Metric collection techniques
  - Server push algorithm
  - Agent probe mechanism
  - Hybrid push/probe technique

Server Push Process

- Typical server response cycle:

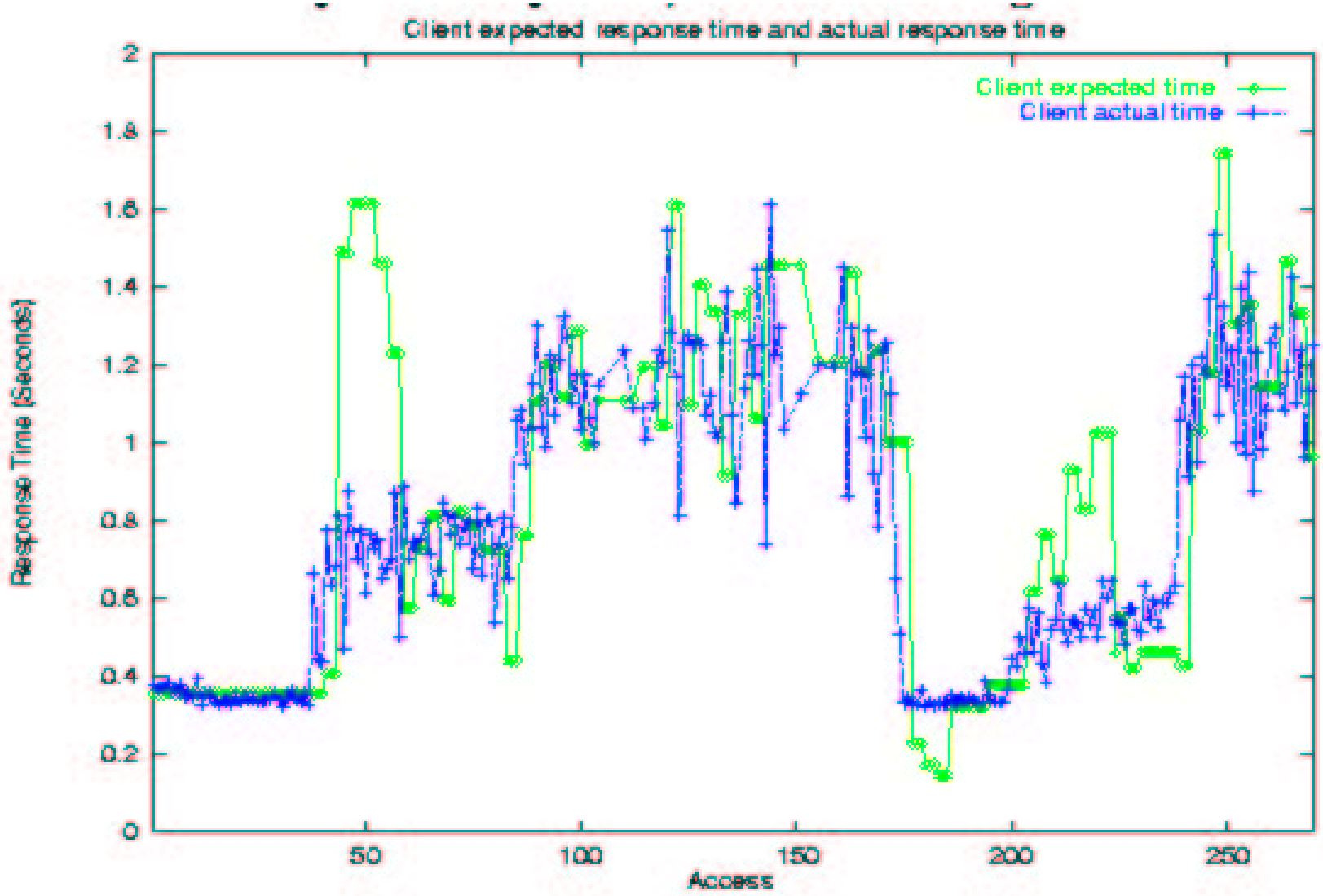
```
assign process to handle query
parse query
locate requested file
repeat until file is written
    read from file
    write to network
```

- Process:
  - Measure and smooth time until first read (TUFR)
  - Push if significant change

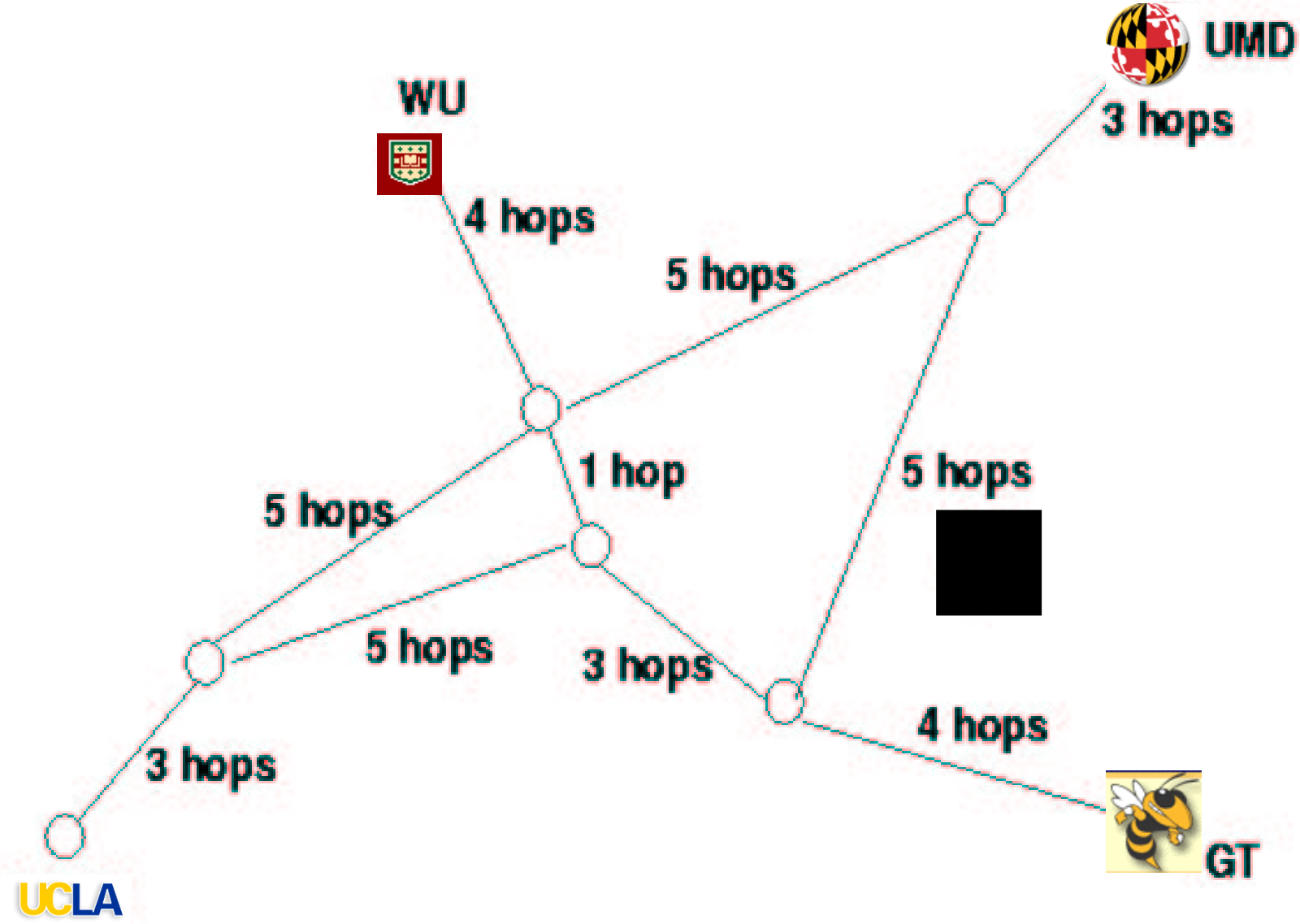
Resolver Process and Hybrid Technique

- Resolver probe process:
  - Request dummy file from server
  - Measure response time (RT)
- Hybrid push-probe technique
  - Dummy file contains most recent TUFR
  - Each probe: compute scaling factor SF = RT/TUFR
  - Each push: estimate RT = SF x TUFR

Performance of Hybrid Algorithm



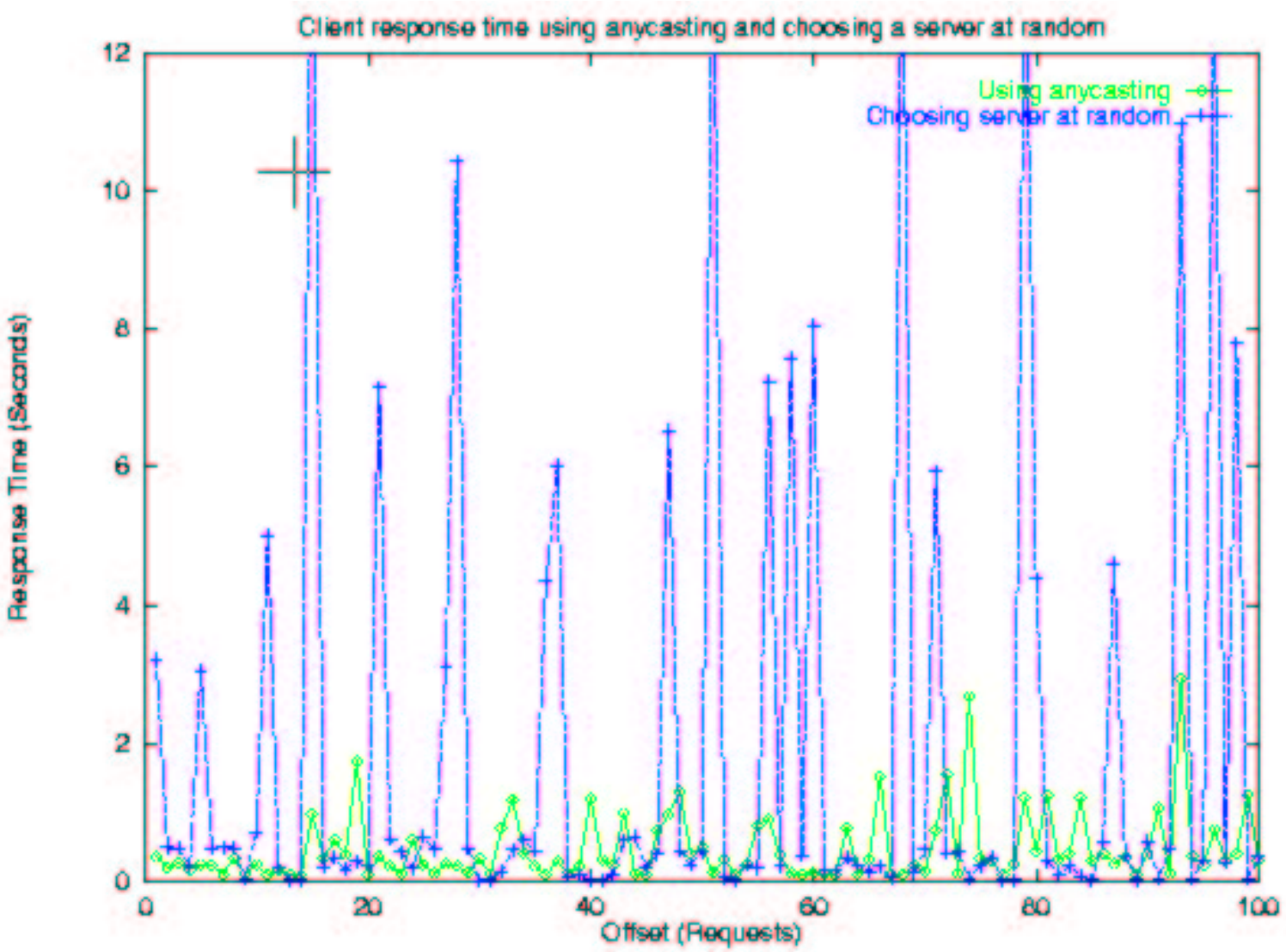
Wide-Area Experiment



Refinement

- Problem of oscillation among servers
- Set of Equivalent Servers (ES)
  - Subset of the replicated servers whose measured performance is within a threshold of best performance

Performance of Anycast vs Random Selection

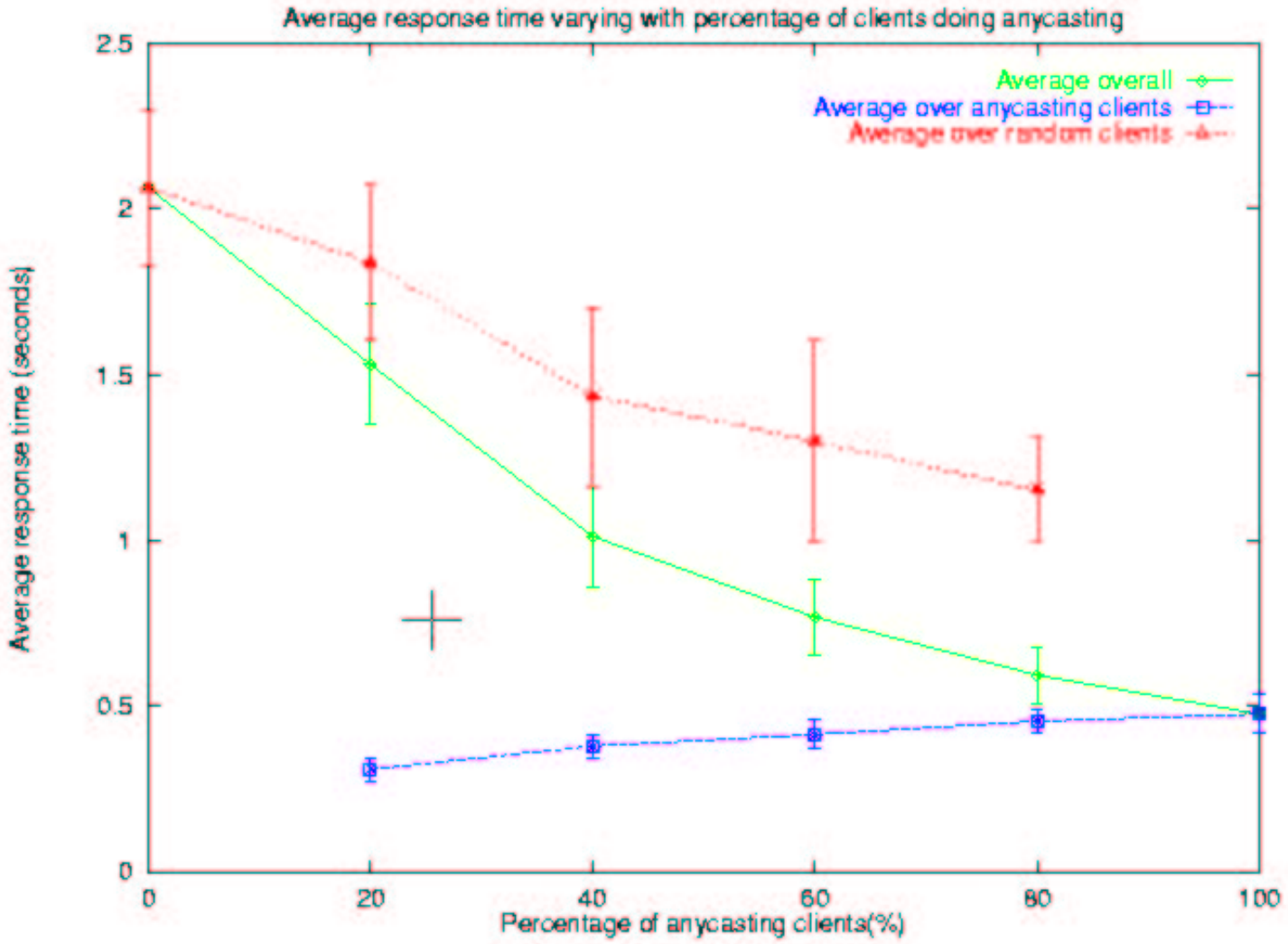


Performance of Server Location Schemes

Server Location Algorithm	Average Response Time (sec.)	Standard Deviation (sec.)
Anycasting	0.49	0.69
Nearest Server	1.12	2.47
Random	2.13	6.96

- 50% improvement using Nearest Server
- Another 50% improvement using Anycasting
- More predictable service

Performance as More Clients Anycast



Avoid Oscillations Among Servers

- Basic technique:
  - Estimate response time for each server
  - Indicate the best server when queried
- Identifying one best server can result in oscillations
- Use set of equivalent servers
- Choose randomly among equivalent servers

Effect of Technique on Server Load

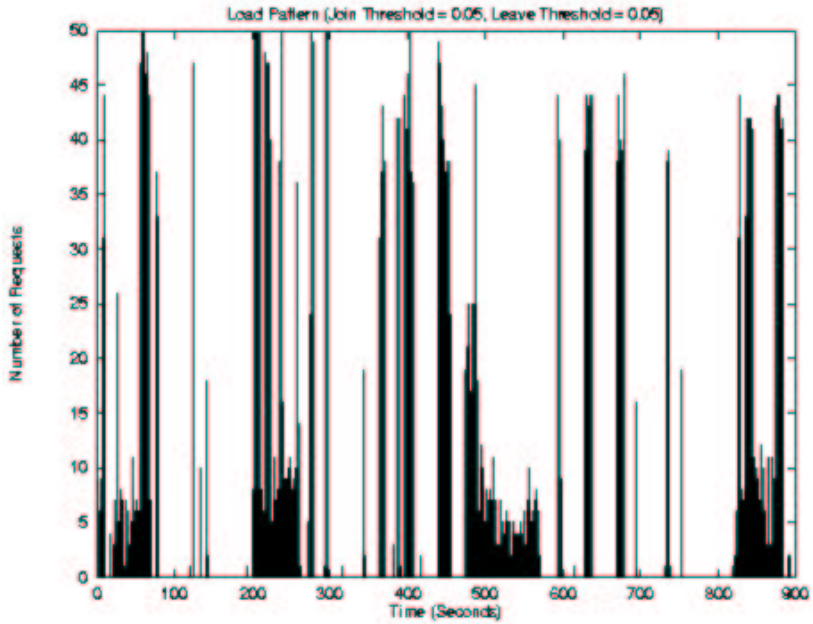


Figure 1. Low Threshold Values

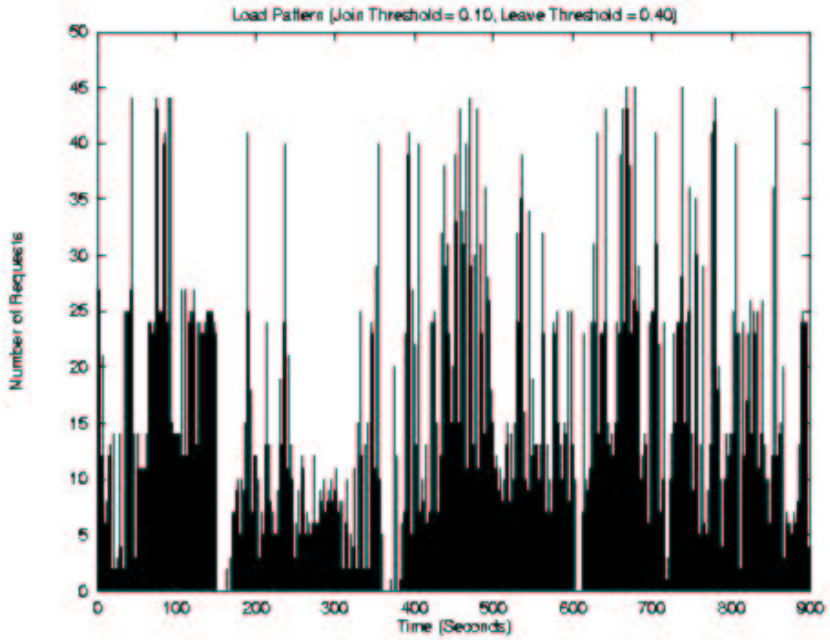


Figure 2: Higher Threshold Values

Scalability Techniques

- Server can multicast pushed data
- Server and resolver can control overhead
- System can limit number of anycast groups
- Resolver can track "most promising" servers
- Users can pay premium for service

Conclusions

- Summary
  - Server replication increasingly important – web services etc.
  - Application layer architecture that is scalable using replicated resolvers organized in a DNS like hierarchy
  - Web server performance can be tracked with reasonable relative accuracy. Techniques used can be generalized to other servers
  - A hybrid push-probe technique provides scalable monitoring. May be useful in other contexts
  - Application-layer anycasting gives significant improvement over other server selection techniques

Any problems (truth in advertising☺)

- Discussions
  - Was the study extensive enough ?
    - 4 Anycast-aware servers – UCLA (1), WUStL(1), Gatech (2)
    - Anycast resolvers – UMD, Gatech
    - 20 Anycast-aware clients – UMD (4), Gatech (16)
- Study of anycast vs random selection
  - Experiments done one after another – any performance difference due to cached content ?
- Would performance improve if network-support for path performance metrics included ?
  - Global IP-anycast [SIGCOMM00]

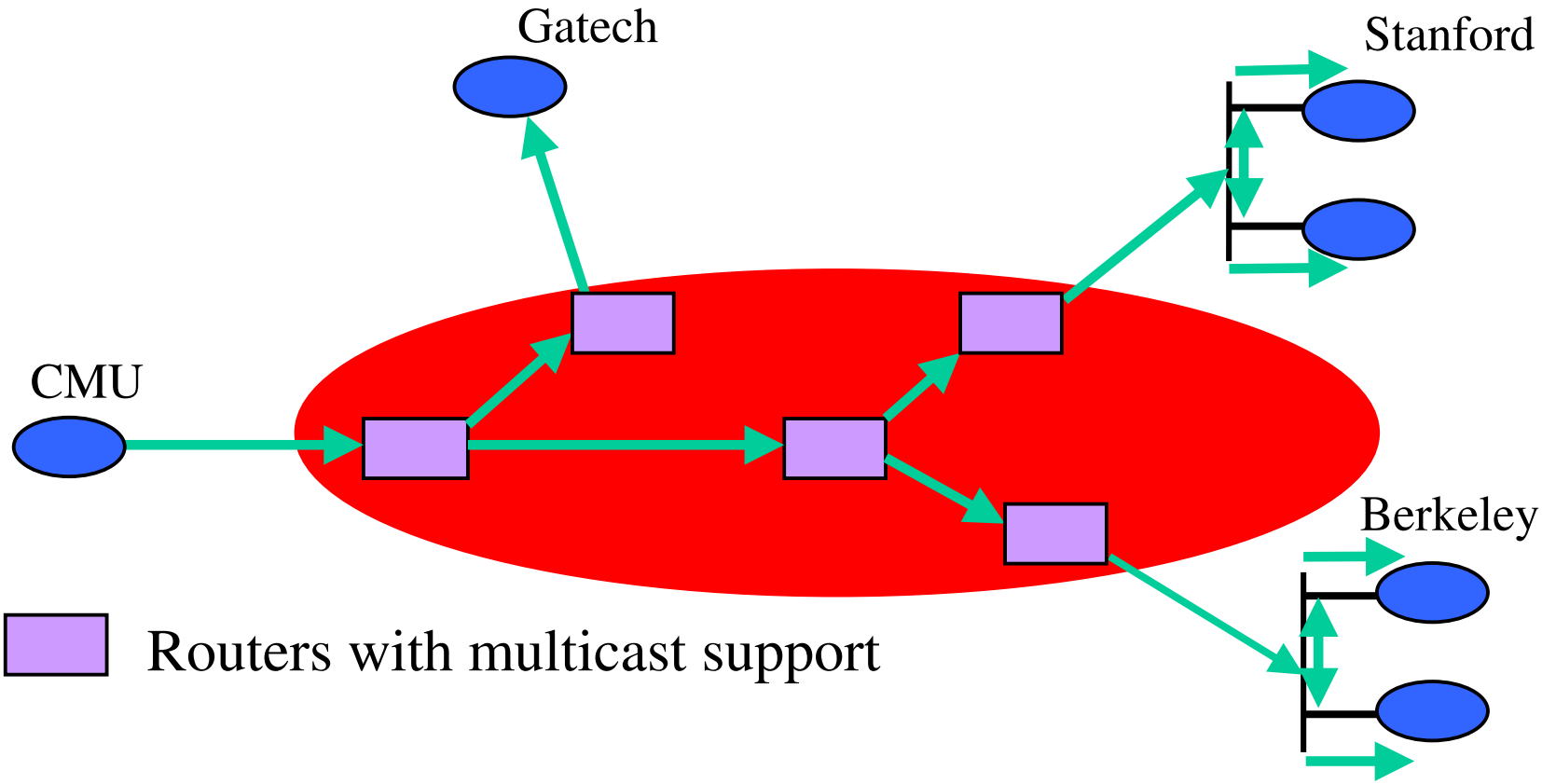
11. A Case For End System Multicast

Yang-hua Chu, Sanjay Rao and Hui Zhang
Carnegie Mellon University

Talk Outline

- Definition, potential benefits & problems
- Look at Narada approach specifically
- Performance in simulation
- Performance in network experiment
- Related work
- Discussion

IP Multicast



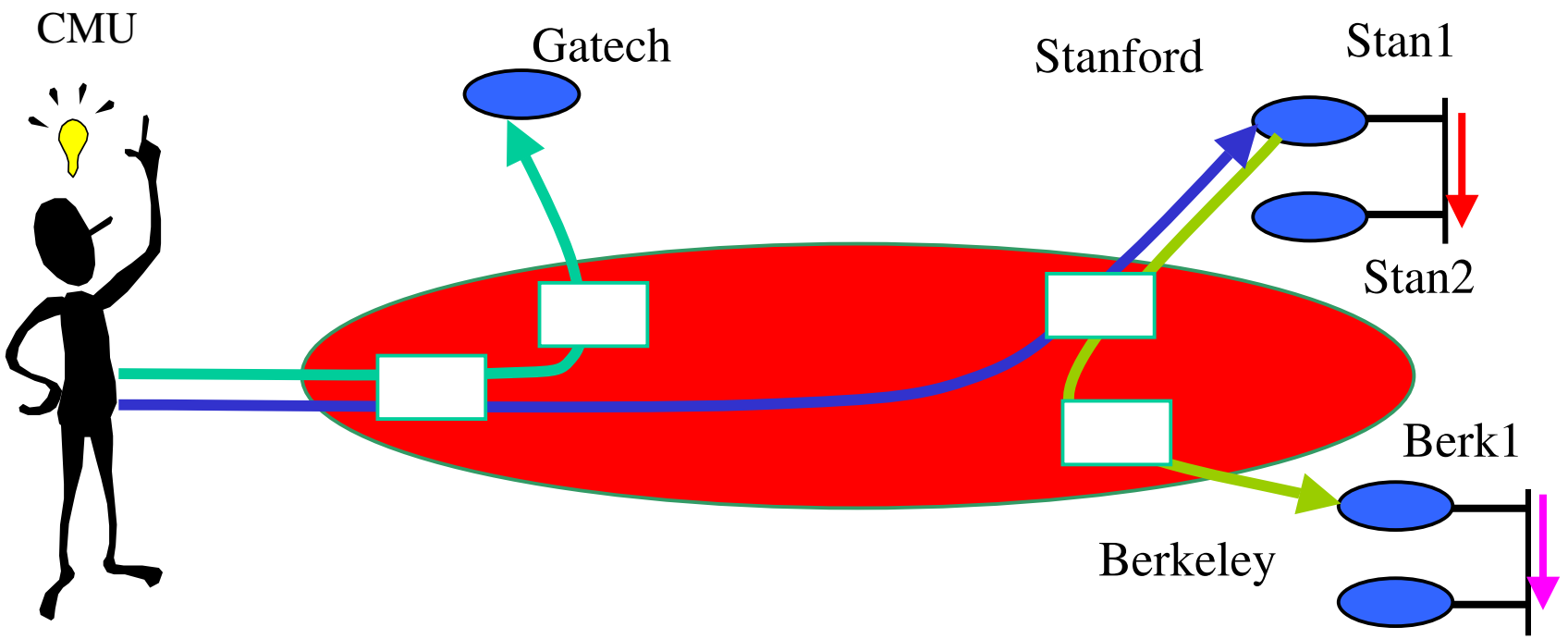
- No duplicate packets
- Highly efficient bandwidth usage

Key Architectural Decision: Add support for multicast in IP layer

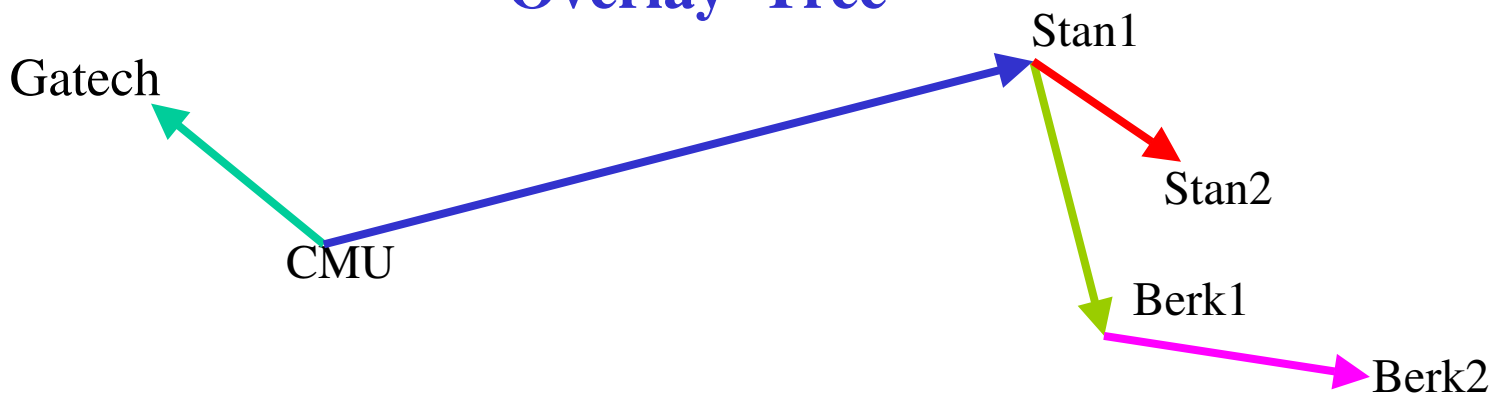
Key Concerns with IP Multicast

- Scalability with number of groups
  - Routers maintain per-group state
  - Analogous to per-flow state for QoS guarantees
  - Aggregation of multicast addresses is complicated
- Supporting higher level functionality is difficult
  - IP Multicast: best-effort multi-point delivery service
  - End systems responsible for handling higher level functionality
  - Reliability and congestion control for IP Multicast complicated
- *Inter-domain routing is hard.*
- *No management of flat address space.*
- Deployment is difficult and slow
  - ISP's reluctant to turn on IP Multicast

End System Multicast



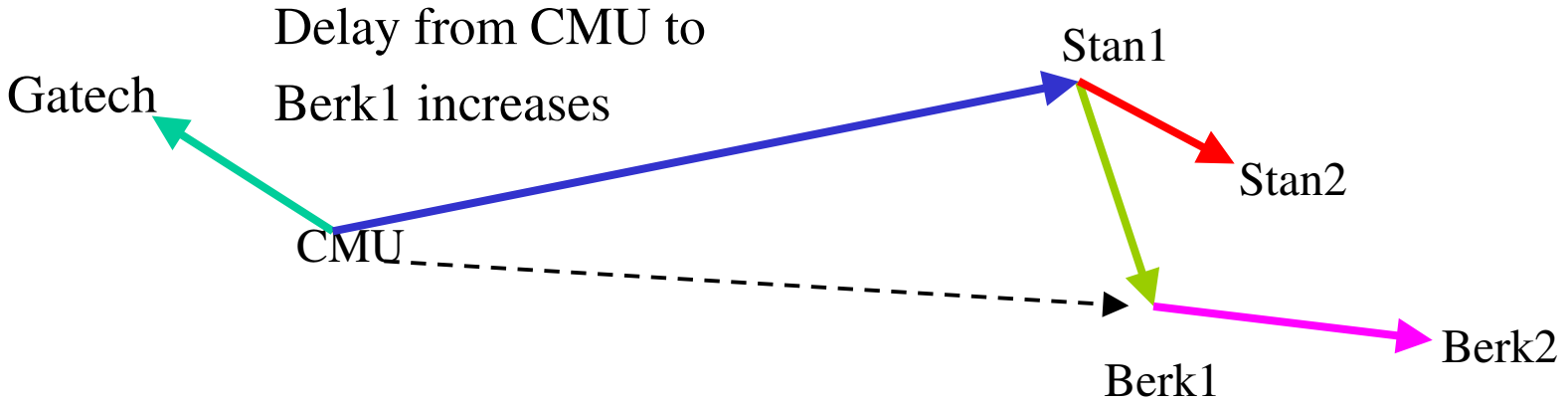
Overlay Tree

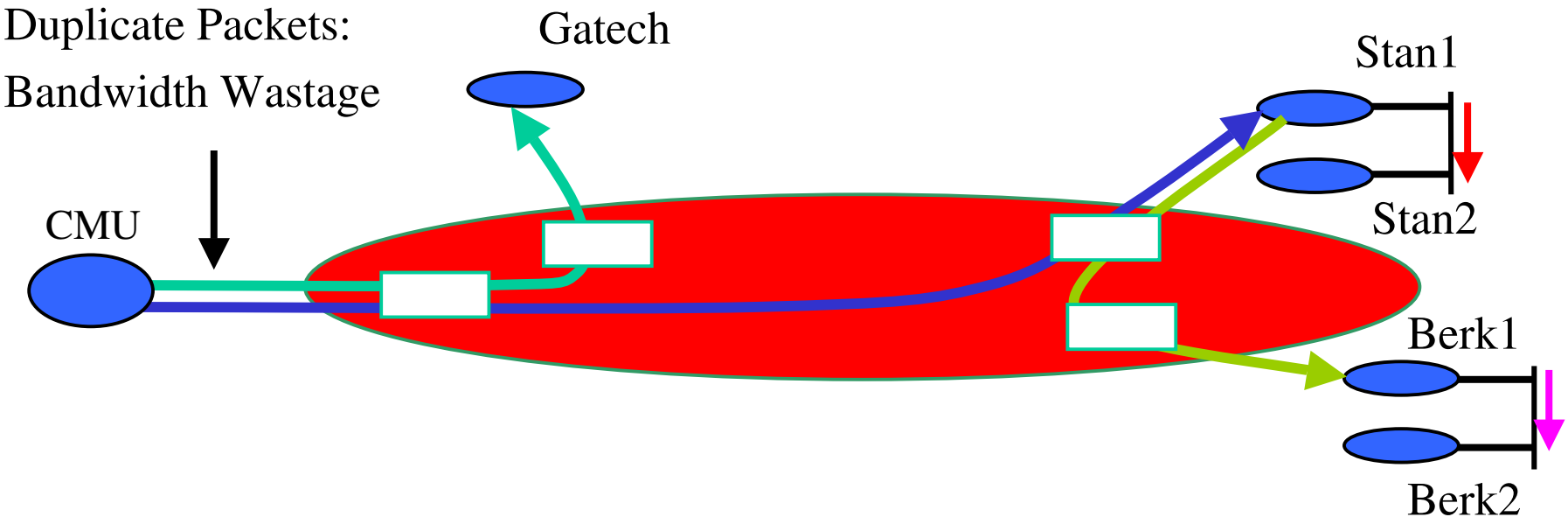


Potential Benefits

- Scalability *(number of sessions in the network)*
  - Routers do not maintain per-group state
  - End systems do, but they participate in very few groups
- Easier to deploy
- Potentially simplifies support for higher level functionality
  - Leverage computation and storage of end systems
  - For example, for buffering packets, transcoding, ACK aggregation
  - Leverage solutions for unicast congestion control and reliability

Performance Concerns



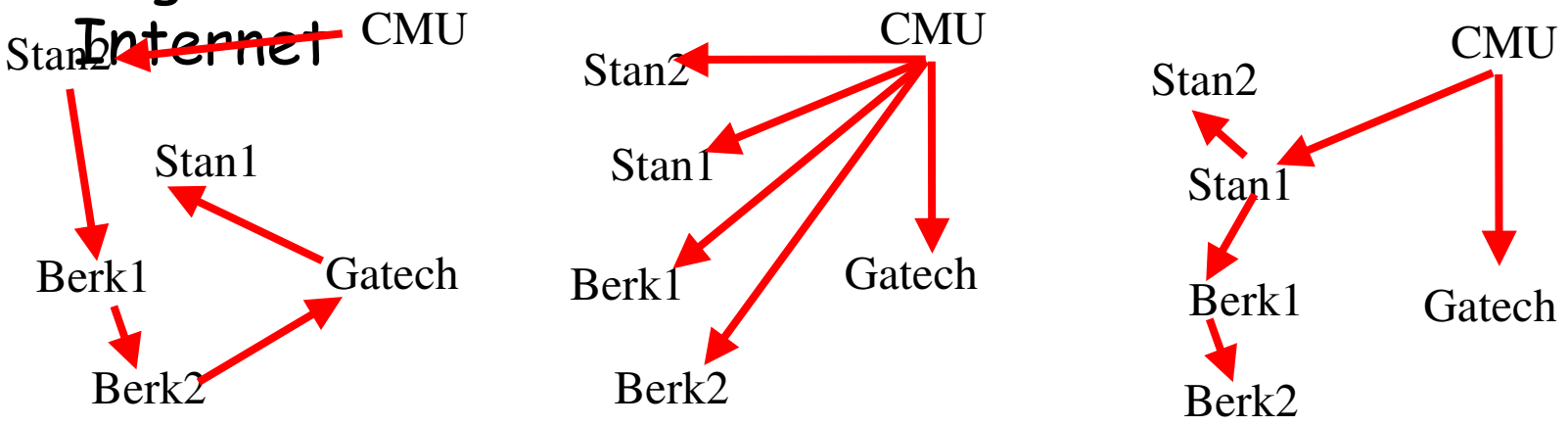


What is an efficient overlay tree?

- The delay between the source and receivers is small
- Ideally,
  - The number of redundant packets on any physical link is low

Heuristic we use:

- Every member in the tree has a small degree
- Degree chosen to reflect bandwidth of connection to Internet



High latency High degree (unicast) "Efficient" overlay

Why is self-organization hard?

- Dynamic changes in group membership
  - Members join and leave dynamically
  - Members may die
- Limited knowledge of network conditions
  - Members do not know delay to each other when they join
  - Members probe each other to learn network related information
  - Overlay must self-improve as more information available
- Dynamic changes in network conditions
  - Delay between members may vary over time due to congestion

Narada Design (1)

Step 0

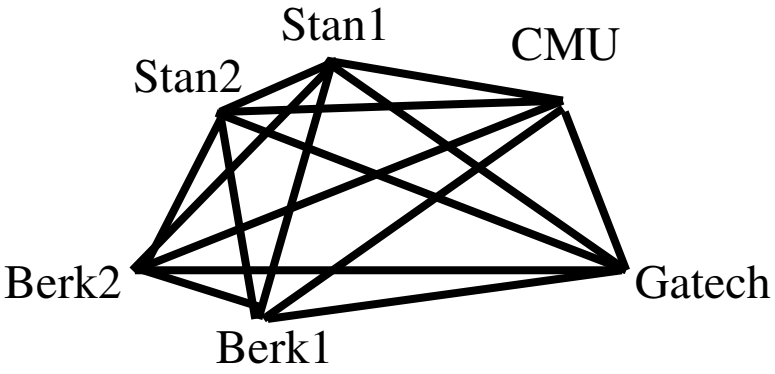
Maintain a complete overlay graph of all group members

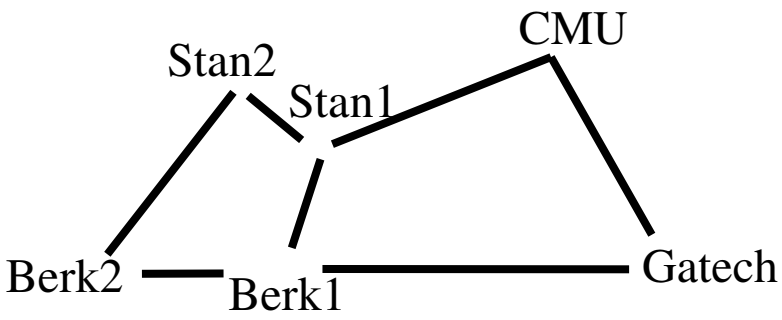
- *Links correspond to unicast paths*
- *Link costs maintained by polling*

Step 1

"Mesh": Subset of complete graph may have cycles and includes all group members

- Members have low degrees
- Shortest path delay between any pair of members along mesh is small

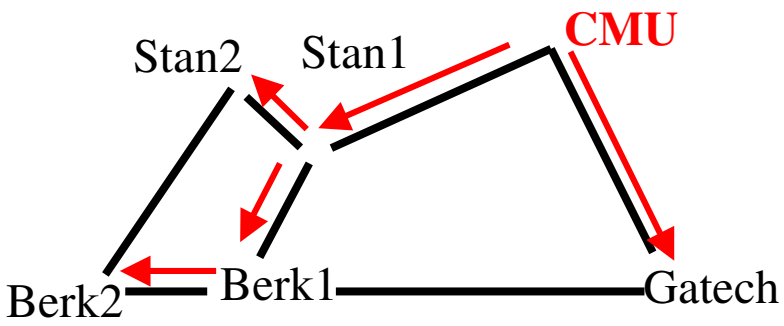




Narada Design (2)

Step 2

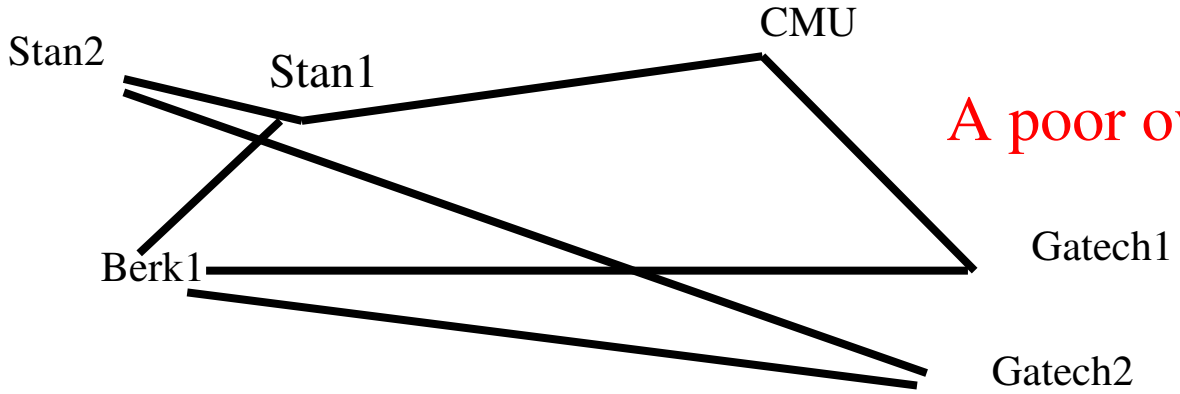
- Source rooted shortest delay spanning trees of mesh
- Constructed using well known routing algorithms
  - Members have low degrees
  - Small delay from source to receivers



Narada Components

- Mesh Management:
  - Ensures mesh remains connected in face of membership changes
- Mesh Optimization:
  - Distributed heuristics for ensuring shortest path delay between members along the mesh is small
- Spanning tree construction:
  - Routing algorithms for constructing data-delivery trees
  - Distance vector routing, and reverse path forwarding

Optimizing Mesh Quality



A poor overlay topology

- Members periodically probe other members at random
- New Link added if

  Utility Gain of adding link > Add Threshold
- Members periodically monitor existing links
- Existing Link dropped if

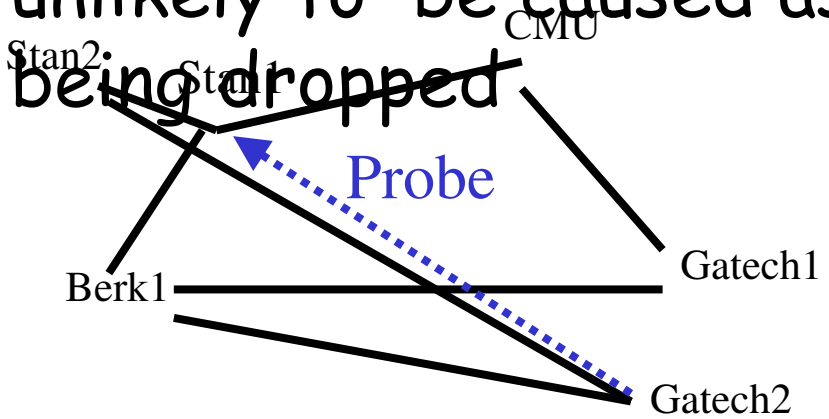
  Cost of dropping link < Drop Threshold

The terms defined

- Utility gain of adding a link based on
  - The number of members to which routing delay improves
  - How significant the improvement in delay to each member is
- Cost of dropping a link based on
  - The number of members to which routing delay increases, for either neighbor
- Add/Drop Thresholds are functions of:
  - Member's estimation of group size
  - Current and maximum degree of member in the mesh

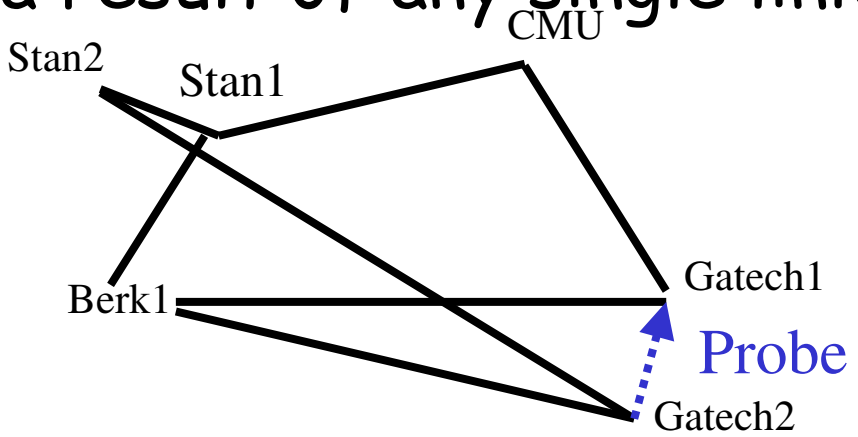
Desirable properties of heuristics

- Stability: A dropped link will not be immediately readded
- Partition Avoidance: A partition of the mesh is unlikely to be caused as a result of any single link being dropped



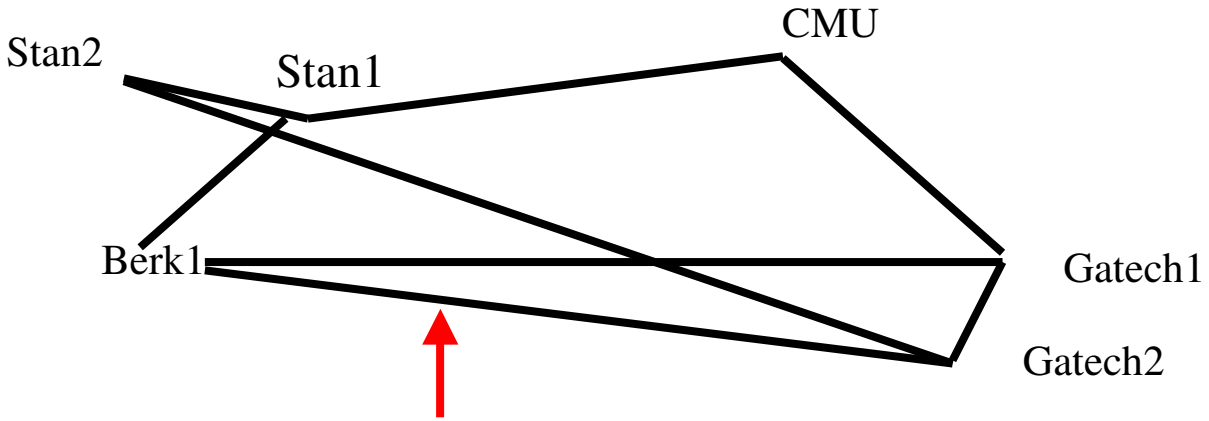
Delay improves to Stan1, CMU but marginally.

Do not add link!



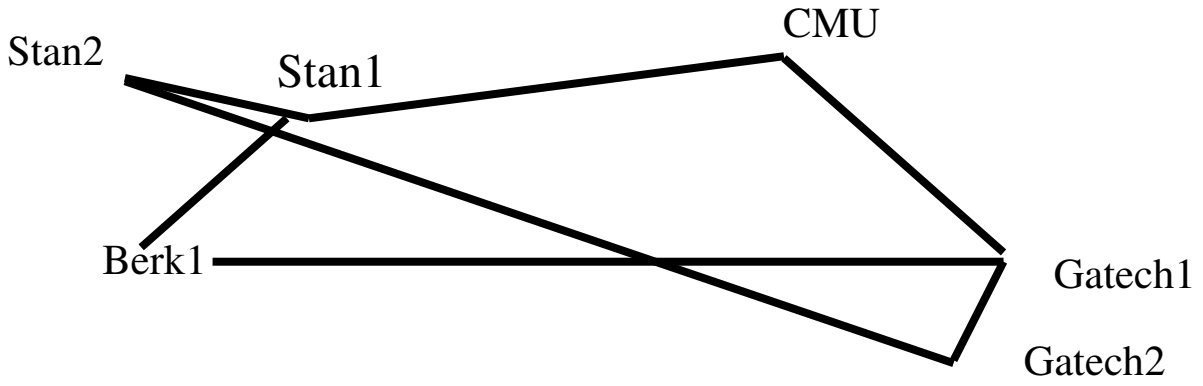
Delay improves to CMU, Gatech1 and significantly.

Add link!



Used by Berk1 to reach only Gatech2 and vice versa.

Drop!!



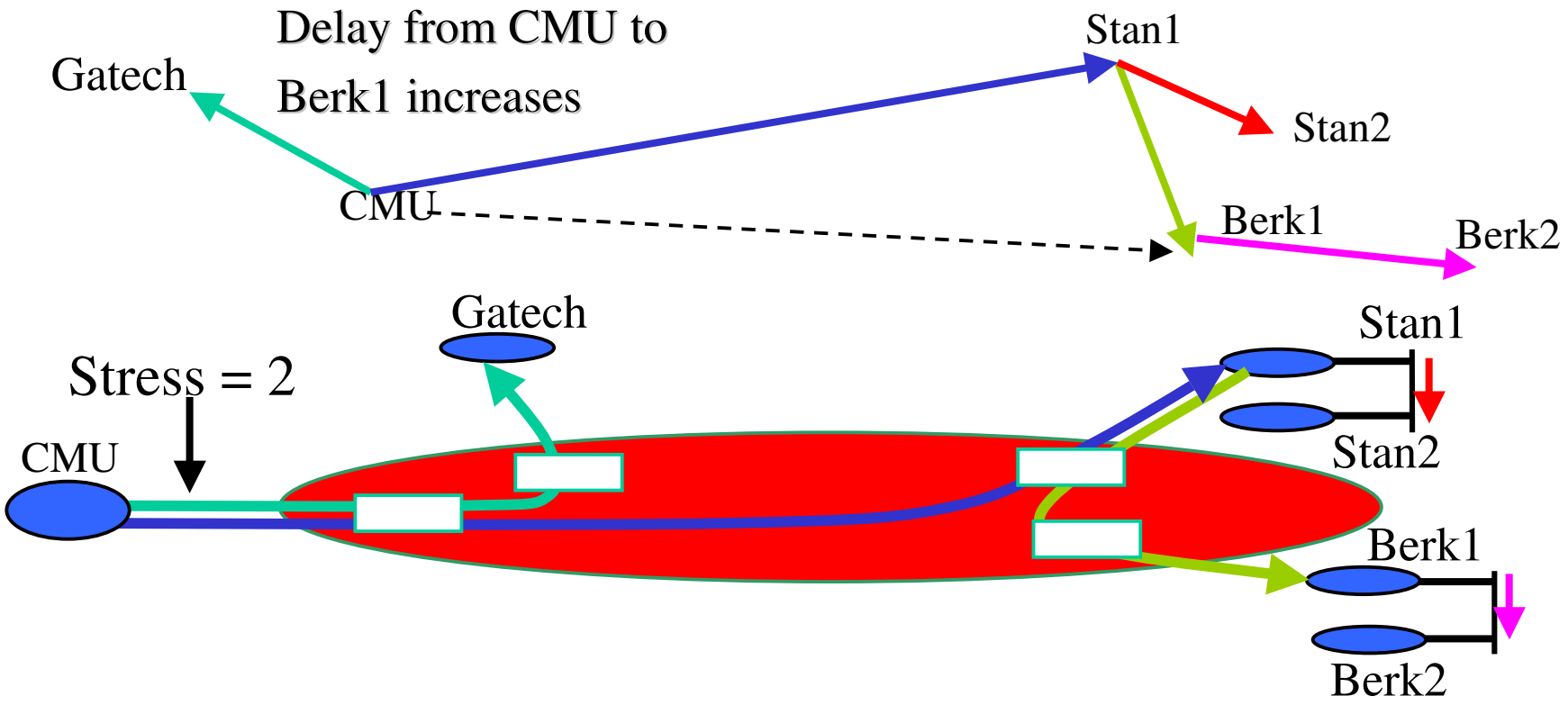
An improved mesh !!

Narada Evaluation

- Simulation experiments
- Evaluation of an implementation on the Internet

Performance Metrics

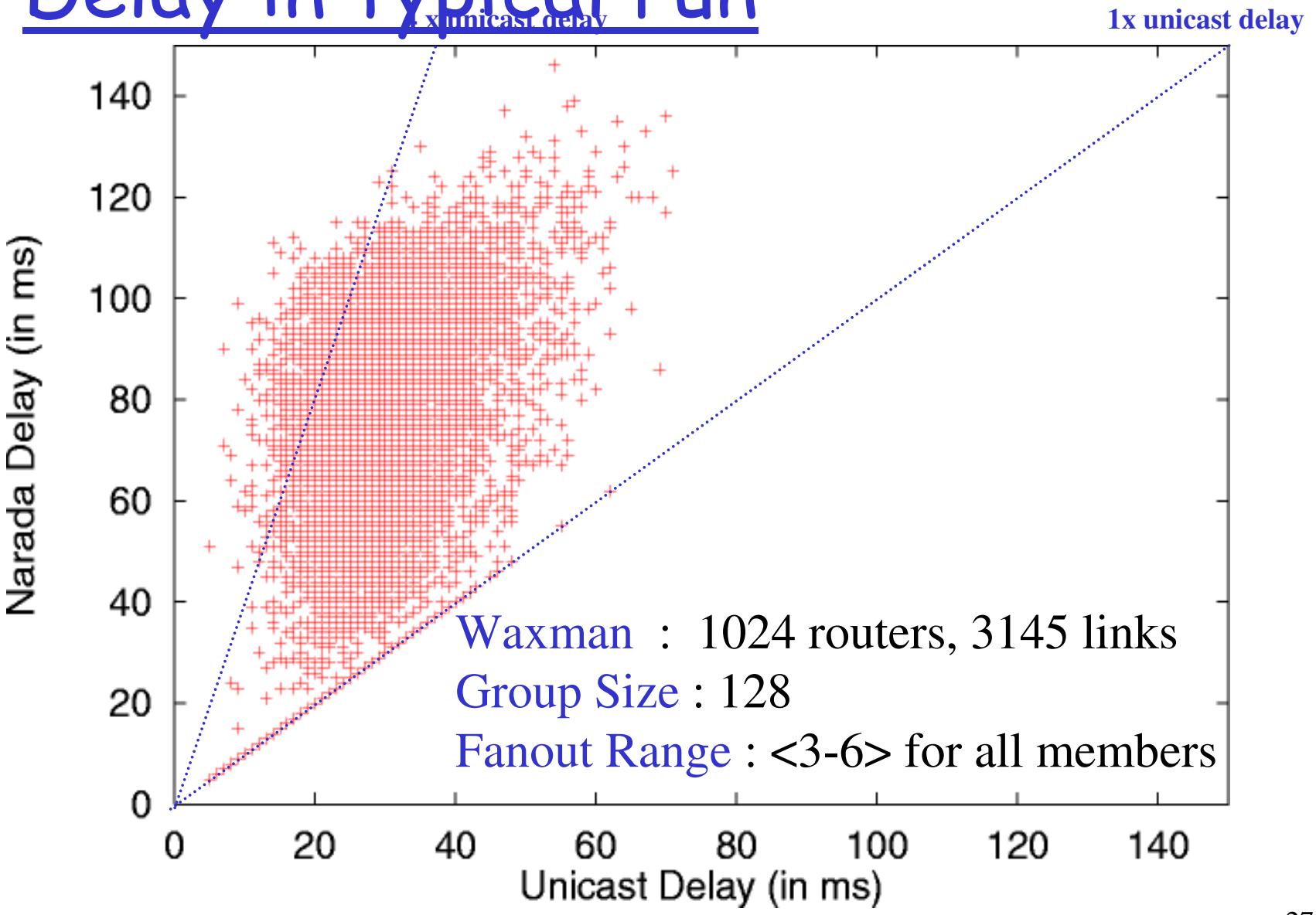
- Delay between members using Narada
- Stress, defined as the number of identical copies of a packet that traverse a physical link



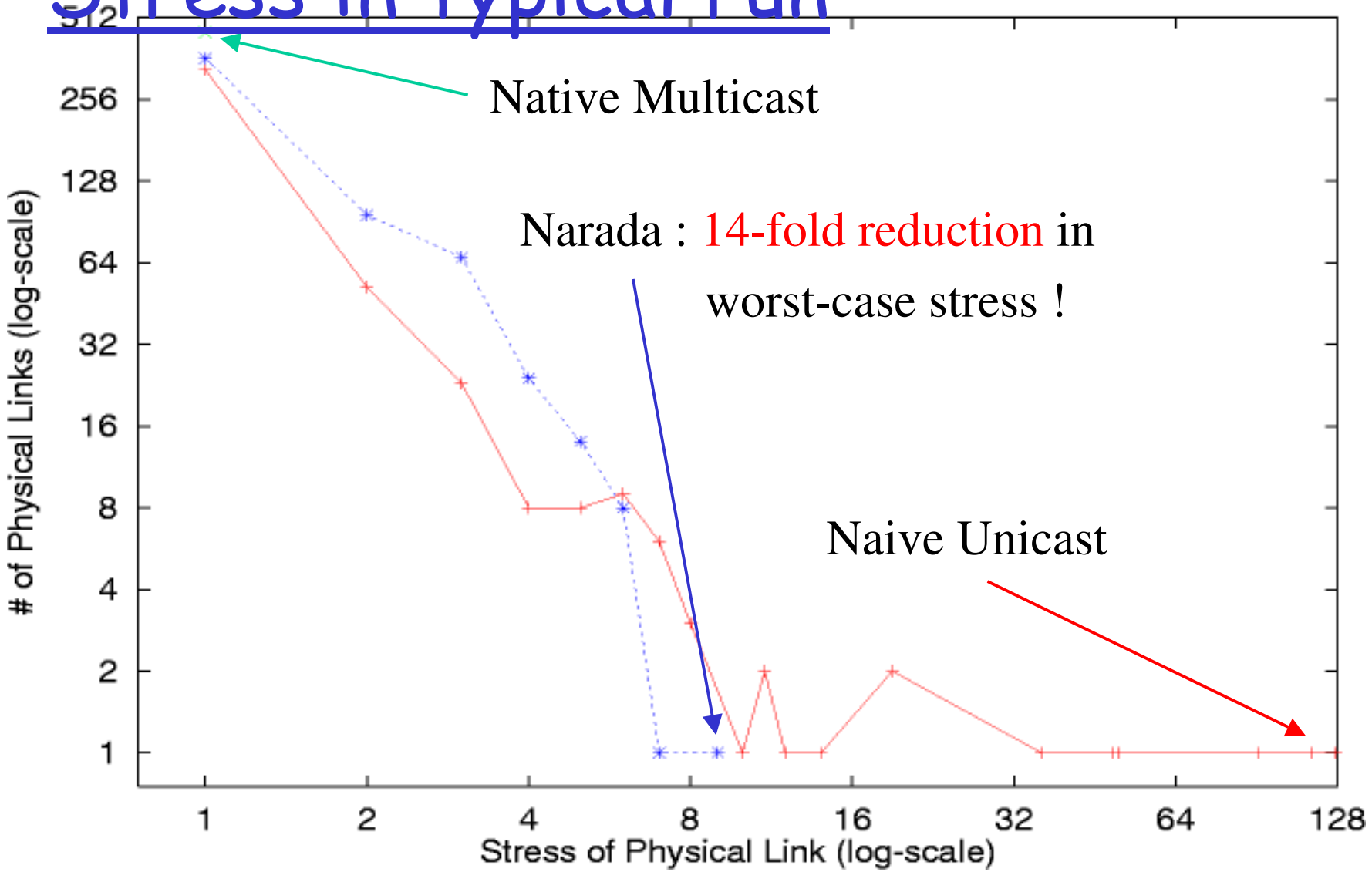
Factors affecting performance

- Topology Model
  - Waxman Variant
  - Mapnet: Connectivity modeled after several ISP backbones
  - ASMap: Based on inter-domain Internet connectivity
- Topology Size
  - Between 64 and 1024 routers
- Group Size
  - Between 16 and 256
- Fanout range
  - Number of neighbors each member tries to maintain in the mesh

Delay in typical run



Stress in typical run



Overhead

- Two sources
  - pairwise exchange of routing and control information
  - polling for mesh maintenance.
- Claim: Ratio of non-data to data traffic grows linearly with group size.
- Narada is targeted at small groups.

Related Work

- Yoid (Paul Francis, ACIRI)
  - More emphasis on architectural aspects, less on performance
  - Uses a shared tree among participating members
    - More susceptible to a central point of failure
    - Distributed heuristics for managing and optimizing a tree are more complicated as cycles must be avoided
- Scattercast (Chawathe et al, UC Berkeley)
  - Emphasis on infrastructural support and proxy-based multicast
    - To us, an end system includes the notion of proxies
  - Also uses a mesh, but differences in protocol details

Conclusions

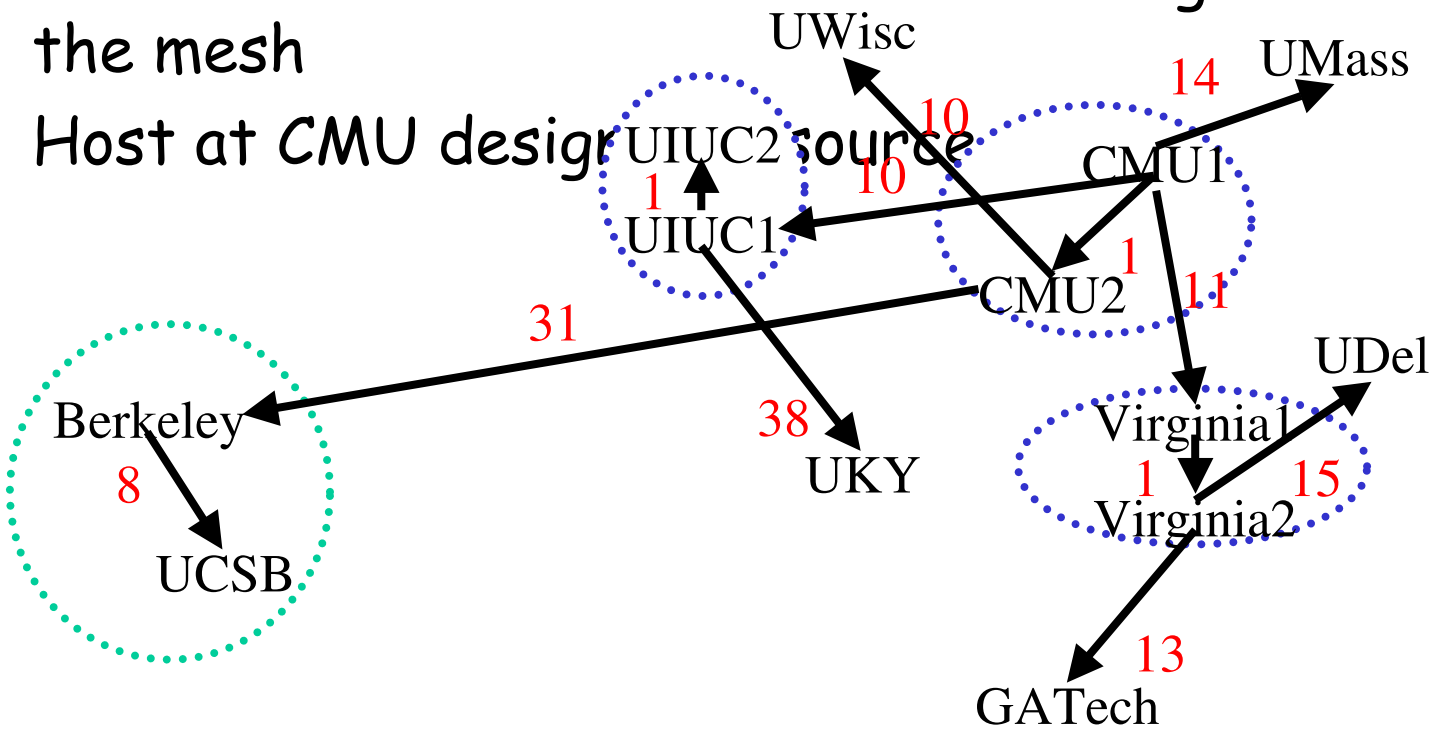
- Proposed in 1989, IP Multicast is not yet widely deployed
  - Per-group state, control state complexity and scaling concerns
  - Difficult to support higher layer functionality
  - Difficult to deploy, and get ISP's to turn on IP Multicast
- Is IP the right layer for supporting multicast functionality?
- For small-sized groups, an end-system overlay approach
  - is feasible
  - has a low performance penalty compared to IP Multicast
  - has the potential to simplify support for higher layer functionality
  - allows for application-specific customizations

Open Questions

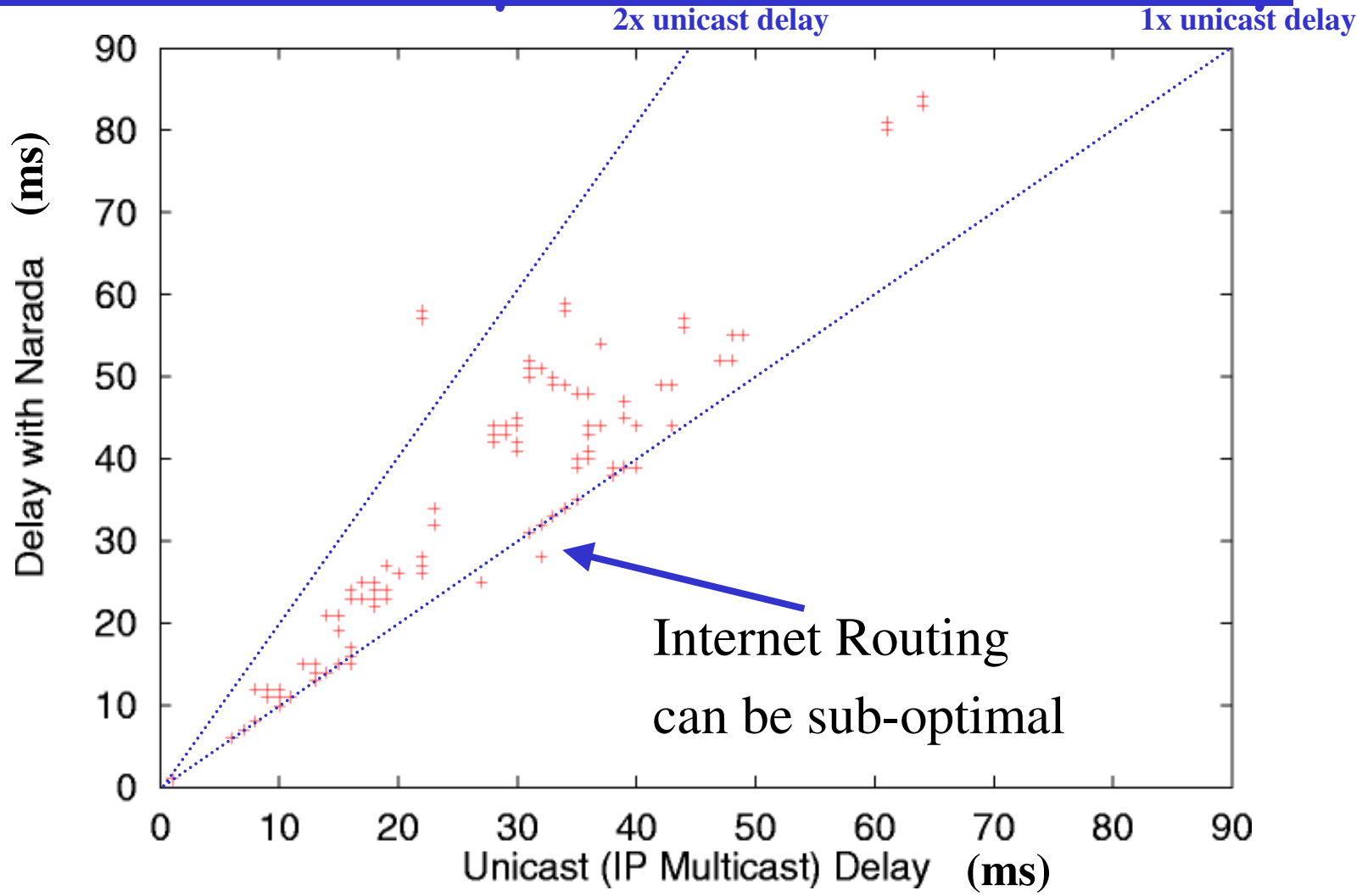
- Theoretical bounds on how close an ESM tree can come to IP multicast performance.
- Alternate approach: Work with complete graph but modify multicast routing protocol.
- Leveraging unicast reliability and congestion contol.
- Performance improvements: Reduce polling overhead.

Internet Evaluation

- 13 hosts, all join the group at about the same time
- No further change in group membership
- Each member tries to maintain 2-4 neighbors in the mesh
- Host at CMU designated source



Narada Delay Vs. Unicast Delay



12. Overcast: Reliable Multicasting with an Overlay Network

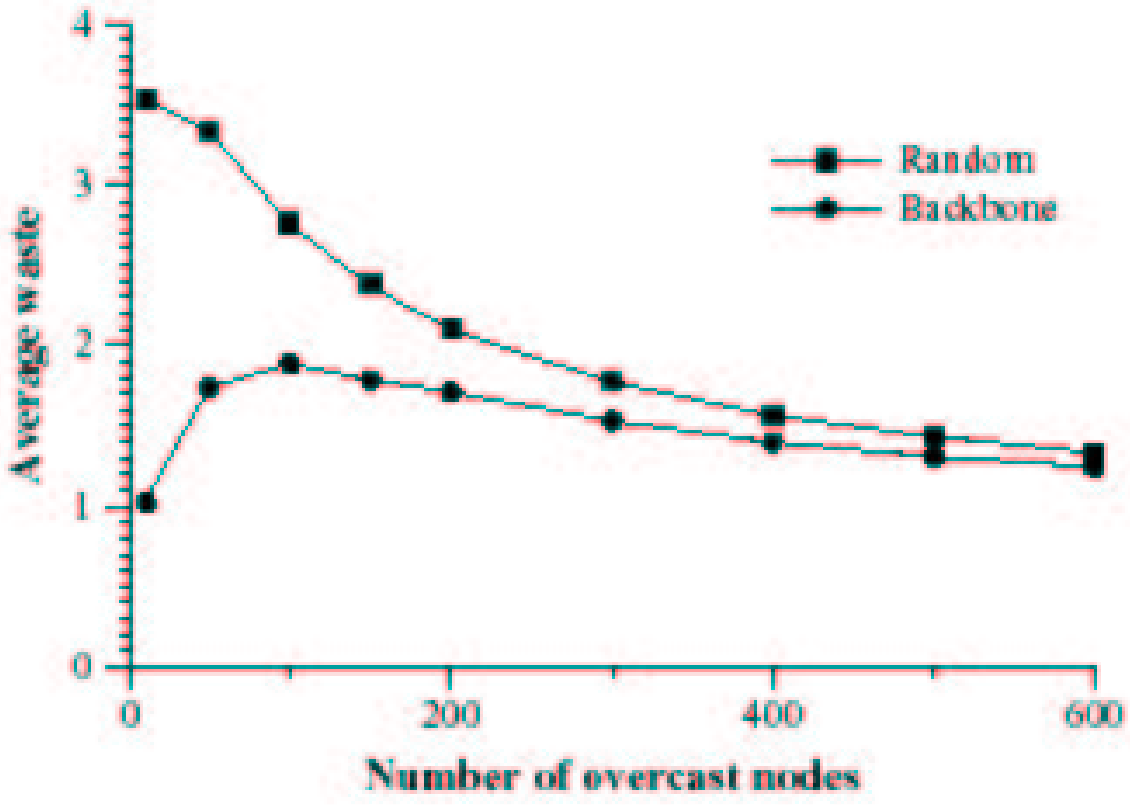
Paper authors: Jannotti, Gifford, Johnson, Kaashoek, O'Toole Jr.

Design Goals

- Provide application-level multicasting using already existing technology via **overcasting**
- Scalable, efficient, and reliable distribution of high quality video
- Compete well against **IP Multicasting**

Overcasting vs IP Multicast

Overcasting imposes about twice as much **network load** as IP multicast (for large networks). This difference is in O(n). (*Figure 4*)



Architecture of an Overcast system

- The entities which form the architecture of the Overcast system are called **nodes**.
- Nodes are connected via an organizational scheme **distribution tree**.
- Each **group** provides **content**, which is replicated at each of the nodes. A node can conceivably participate in more than one group.
- **Clients** are end-system consumers of content.

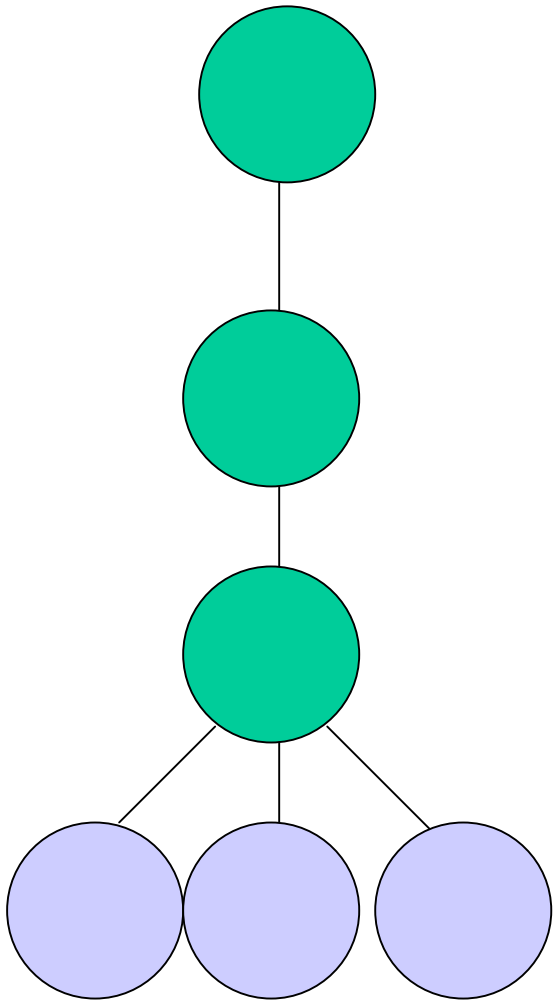
Root nodes (1)

- Content originates at the root node and is streamed down the distribution tree.
- The root is the administrational center of the group.
- When clients join the group, they go to a webpage at the root, which redirects them to the "best" overcast node
- Is the root a single point of failure?

Root nodes (2)

- **Linear roots** can alleviate this problem. For the source to fail, all roots must fail.

- Using **round robin IP resolution**, you can stop your content serving cluster from being ‘slashdotted’ (overloaded by sheer popularity) by client requests.



Distribution tree

- The distribution tree is built and maintained using a **self-organizing** algorithm.
- The primary heuristic of this algorithm is to maximize **bandwidth** from the root to an overcast node.
- **Backbone nodes** are nodes which are located on or close to a network backbone. Overcast performs better when these backbone nodes are located at the top of the tree (ie, they are switched on first)

Tree building protocol (1)

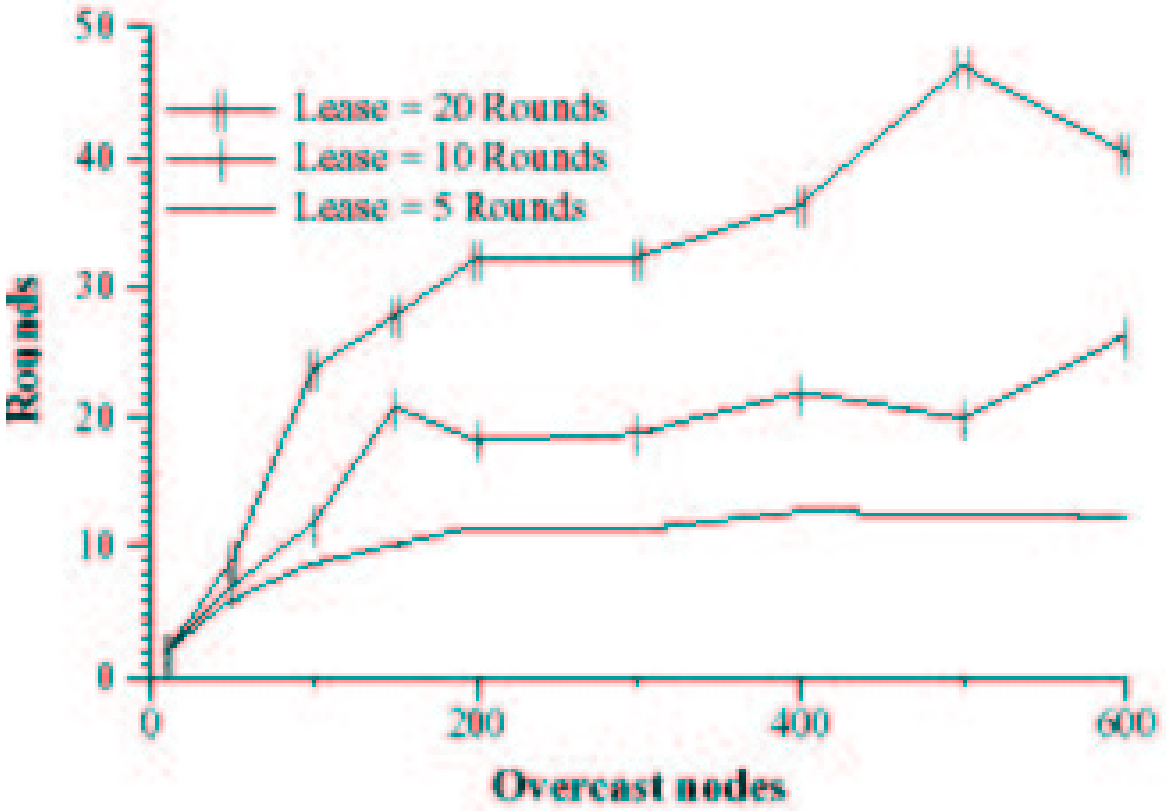
- A node **initializes** by booting up, obtaining its IP, and contacts a "global, well-known registry" (Possible point of failure?) with a unique **serial number**.
- **Registry** provides a list of groups to join.
- This node initially chooses the root as its **parent**. A series of **rounds** will begin in which the node decides where on the tree it should be.

Tree building protocol (2)

- For each round, evaluate the bandwidth we have to our parent. Also consider the bandwidth to the rest of our parent's children.
- If there is a tie (bandwidth differences between 2 or more nodes within 10%), break it by the number of hops reported by **traceroute**.
- The child selects the best of it's parents children as its new parent.
- Nodes maintain an **ancestor list** and can rejoin further up the tree when it's ancestors fail.

Reaching a stable state

Overcast nodes can still receive content from the root, even when the tree is not stabilized. A typical round period is about 1 to 2 seconds.



Tree building protocol (3)

- By having consecutive rounds of tree building, the distribution tree can overcome conditions occuring on the underlying network.
- The **up/down** protocol is used to maintain information about the status of nodes.
- Each node in the network maintains a table of information about all of it's descendants.
- After the tree stabilizes, nodes will continue to consider relocating after a **reevaluation period**.

Up/down protocol

- A parent that gets a new child gives its parent a **birth certificate**, which propagates to the root. The node's **sequence number** is incremented by 1. Sequence numbers are used to prevent a race condition.
- When a parent's child fails to report its status (called **checkin**), after a length of time called a **lease period**, the parent propagates a **death certificate** for its child up the tree.
- A child also presents any certificates or changes it has accumulated from its last checkin.

Problems

- A simple approach leads to a simple solution:
- Not appropriate for software (same as mirroring a file!)
- Not appropriate for games (latency is too high!)
- What about teleconferencing? Authors suggest that if a non-root node wishes to send to other nodes, the node should first **unicast** (send via normal TCP/IP to the root, which will then overcast it as normal). Is this a good solution?

Closing thoughts

- Simulated on a virtual network topology (Georgia Tech Internetwork Topology Models). Take results with a grain of salt (or two).
- Might work out well commercially. Consider a high demand for high definition video over the net, and corporations/entities willing to deliver it (CNN,Hollywood studios,Olympics). Overcast could be a premium service for ISP subscribers [like newsgroups,www hosting].

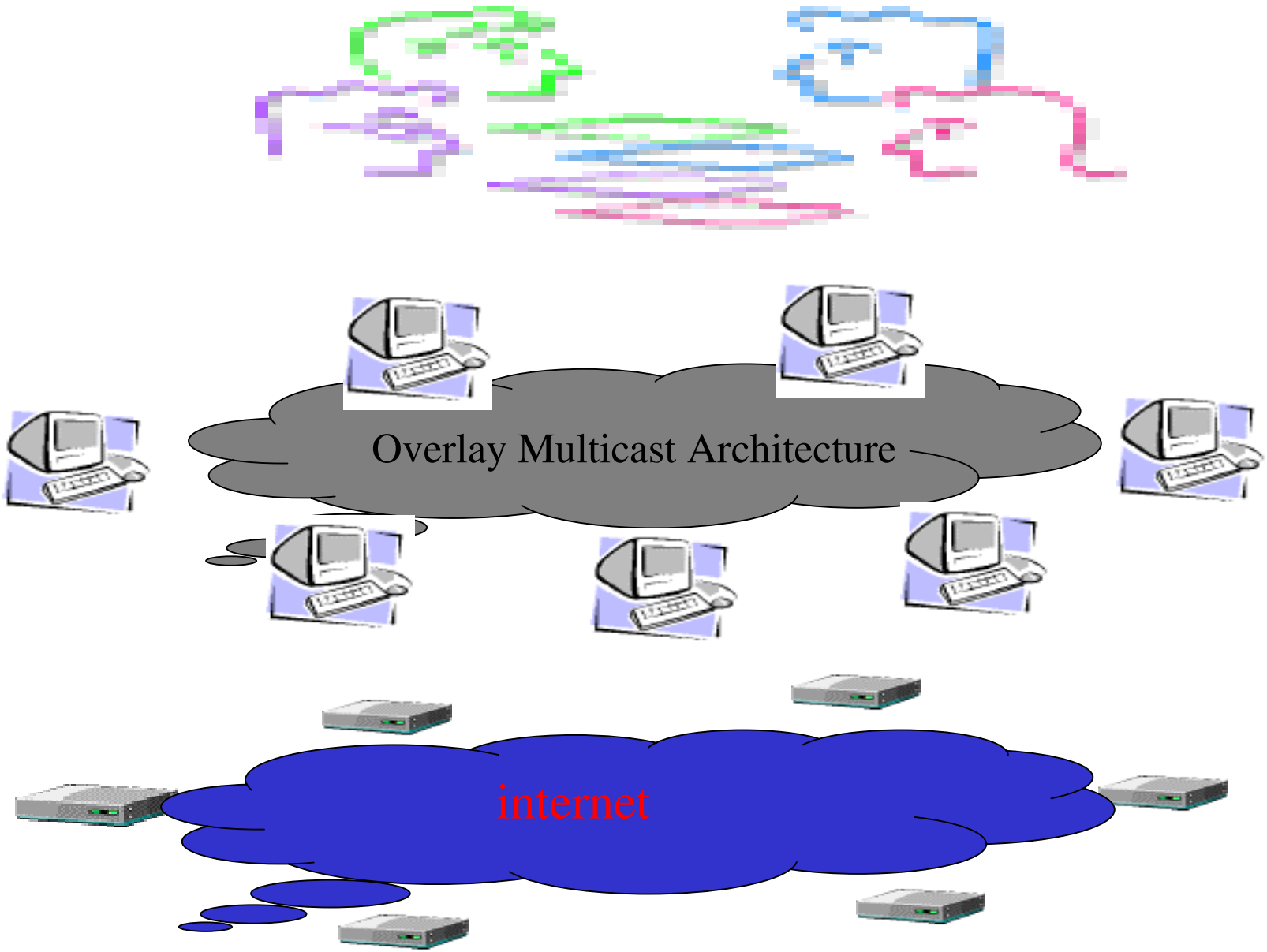
Lecture 5: Applications

- Real Time Conferencing
- Generalised File System
- Other…
- Conclusion of p2p 5 lectures…

13. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture

Yanghua Chu et al. (CMU)
SIGCOMM'01

most slides comes from authors presentation


Overlay Multicast Architecture
internet

Past Work

- Self-organizing protocols
  - Yoid (ACIRI), Narada (CMU), Scattercast (Berkeley), Overcast (CISCO), Bayeux (Berkeley), …
  - Construct overlay trees in distributed fashion
  - Self-improve with more network information

- Performance results showed promise, but…
  - Evaluation conducted in simulation
  - Did not consider impact of network dynamics on overlay performance

Focus of This Paper

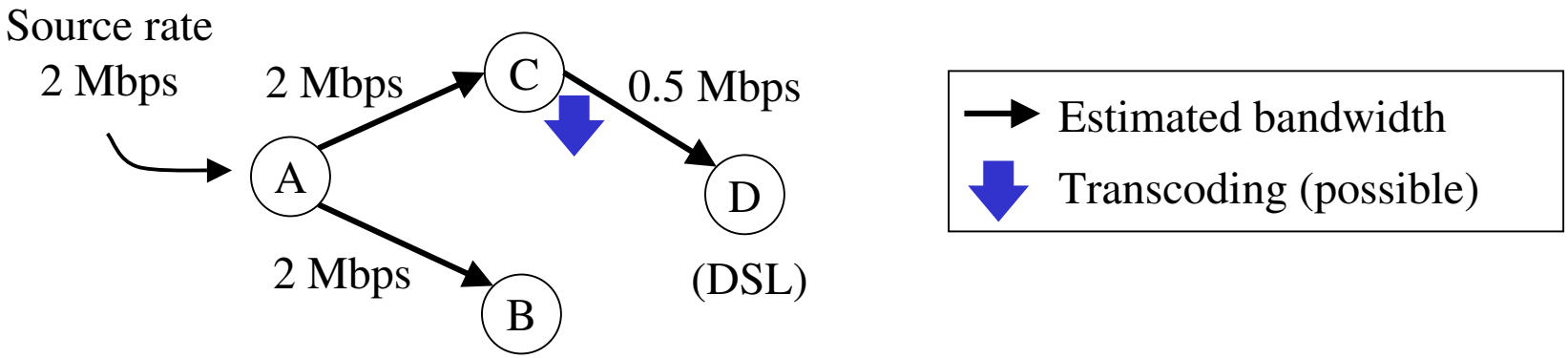
- Can End System Multicast support real-world applications on the Internet?
  - Study in context of conferencing applications
  - Show performance acceptable even in a dynamic and heterogeneous Internet environment

- First detailed Internet evaluation to show the feasibility of End System Multicast

Why Conferencing?

- Important and well-studied
  - Early goal and use of multicast (vic, vat)
- Stringent performance requirements
  - High bandwidth, low latency
- Representative of interactive applications
  - E.g., distance learning, on-line games

Supporting Conferencing in ESM



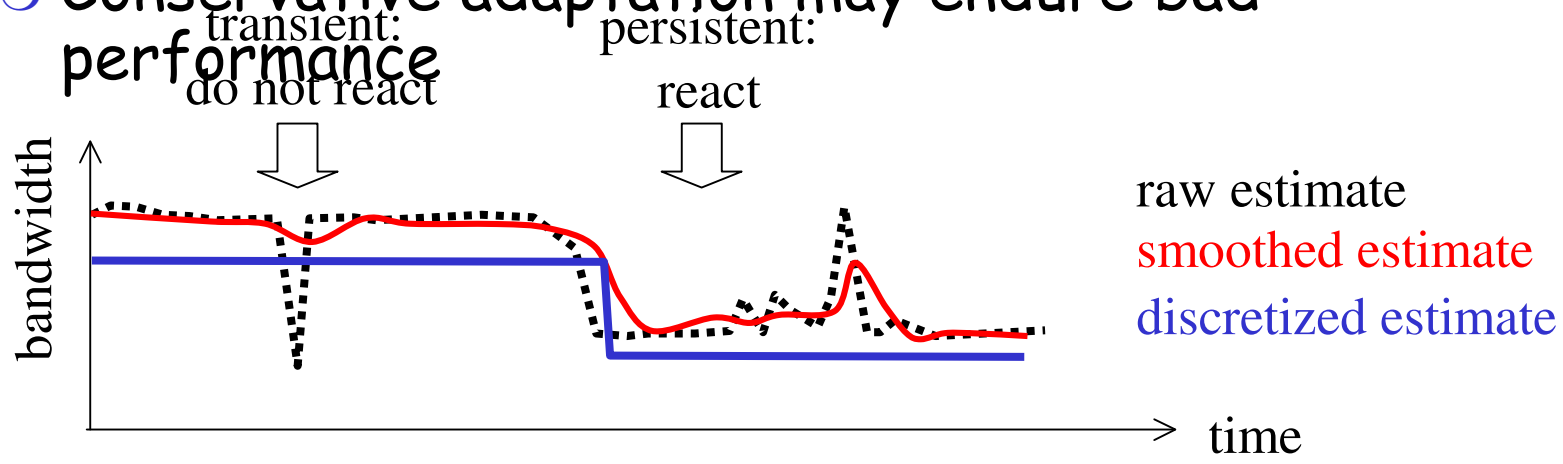
- Framework
  - Bandwidth estimation
  - Adapt data rate to bandwidth est. by packet dropping
- Objective
  - High bandwidth and low latency to all receivers along the overlay

Enhancements of Overlay Design

- Two new issues addressed
  - Dynamically adapt to changes in network conditions
  - Optimize overlays for multiple metrics
    - Latency and bandwidth
- Study in the context of the Narada protocol (Sigmetrics 2000)
  - Use Narada to define logical topology
  - Techniques presented apply to all self-organizing protocols

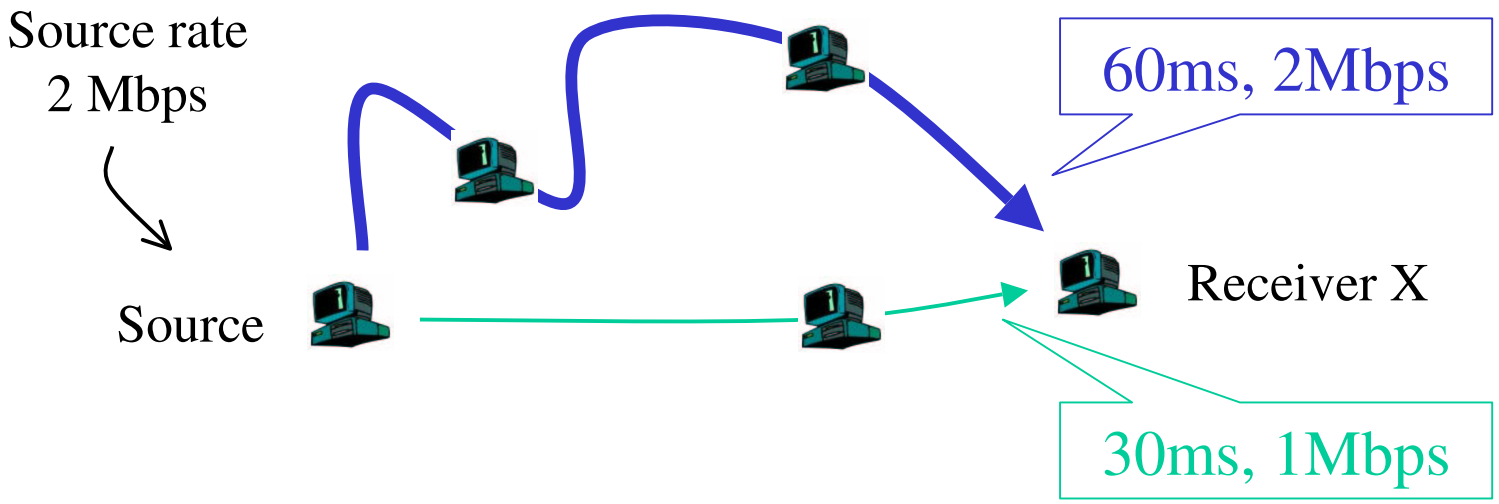
Adapt to Dynamic Metrics

- Adapt overlay trees to changes in network condition
  - Monitor bandwidth and latency of overlay links
- Link measurements can be noisy
  - Aggressive adaptation may cause overlay instability
  - Conservative adaptation may endure bad performance



- Capture the long term performance of a link
  - Exponential smoothing, Metric discretization

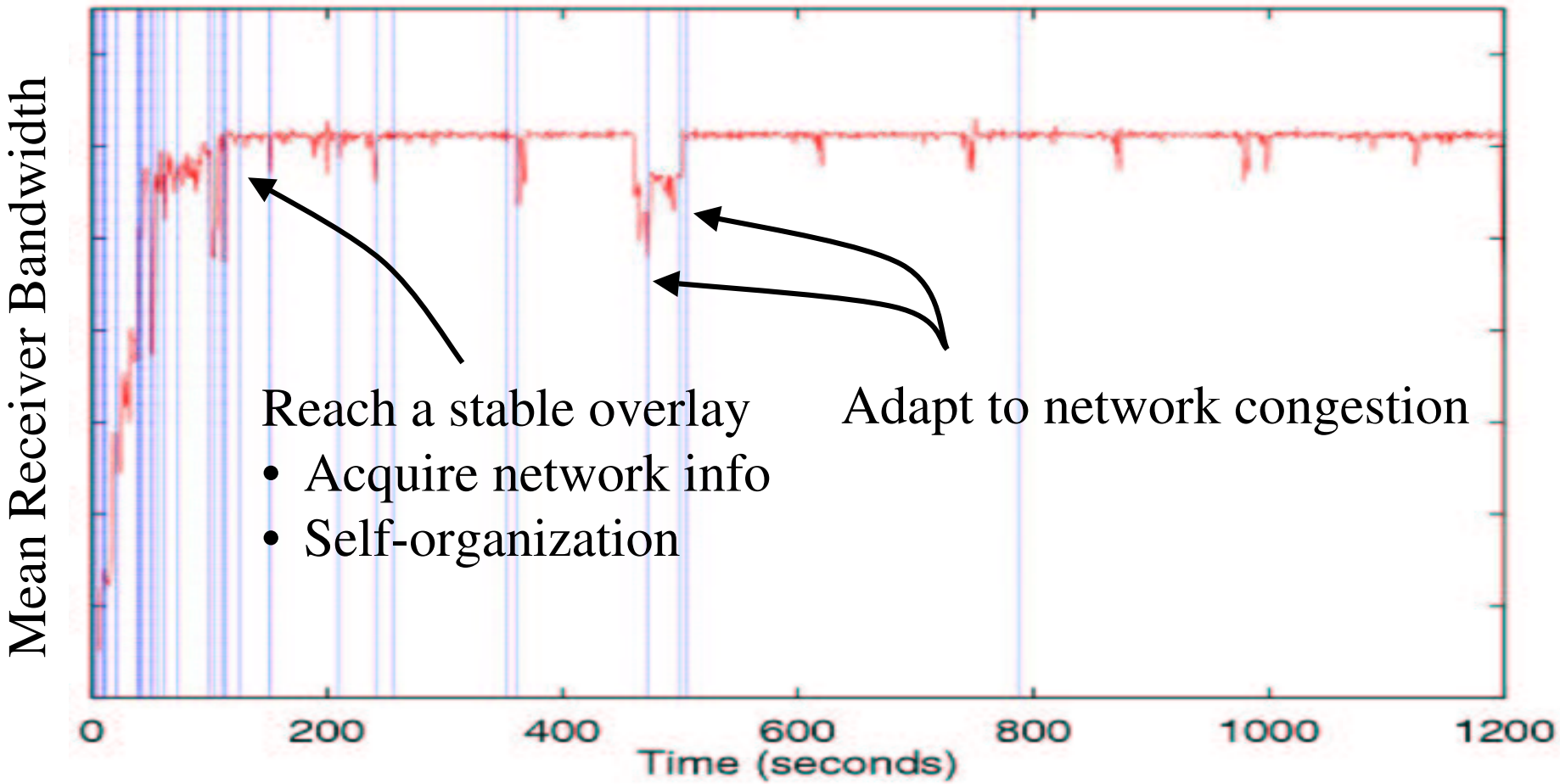
Optimize Overlays for Dual Metrics



- Prioritize bandwidth over latency
- Break tie with shorter latency

Example of Protocol Behavior

- All members join at time 0
- Single sender, CBR traffic



Evaluation Goals

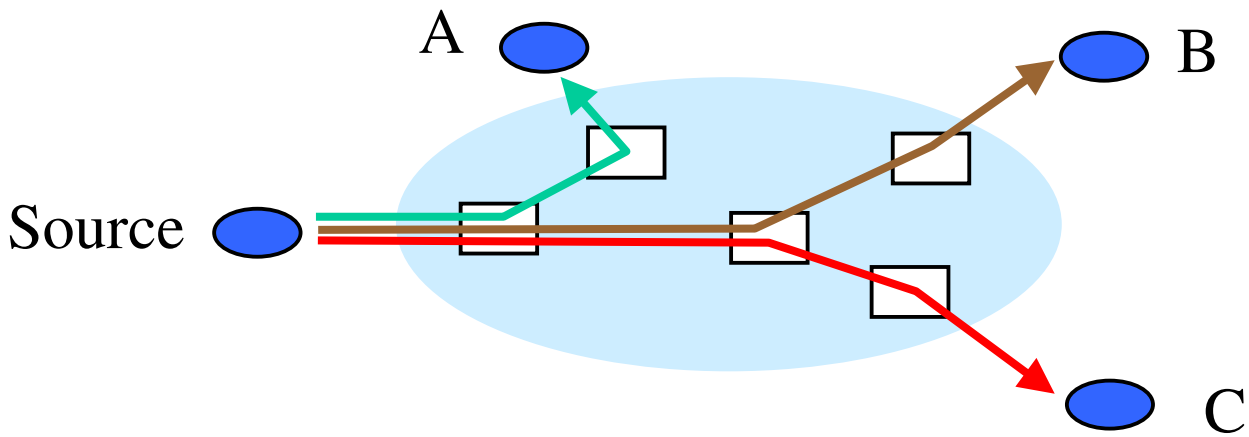
- Can ESM provide application level performance comparable to IP Multicast?
- What network metrics must be considered while constructing overlays?
- What is the network cost and overhead?

Evaluation Overview

- Compare performance of our scheme with
  - Benchmark (sequential unicast, mimicing IP Multicast)
  - Other overlay schemes that consider fewer network metrics
- Evaluate schemes in different scenarios
  - Vary host set, source rate
- Performance metrics
  - Application perspective: latency, bandwidth
  - Network perspective: resource usage, overhead

Benchmark Scheme

- IP Multicast not deployed
- Sequential Unicast: an approximation
  - Bandwidth and latency of unicast path from source to each receiver
  - Performance similar to IP Multicast with ubiquitous deployment



Overlay Schemes

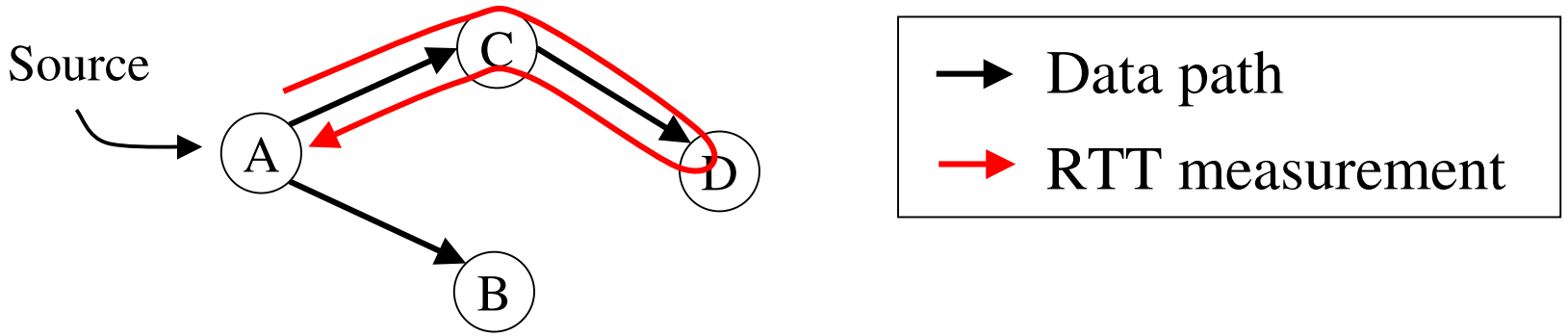
Overlay Scheme	Choice of Metrics	
	Bandwidth	Latency
Bandwidth-Latency	✓	✓
Bandwidth-Only	✓	✗
Latency-Only	✗	✓
Random	✗	✗

Experiment Methodology

- Compare different schemes on the Internet
  - Ideally: run different schemes concurrently
  - Interleave experiments of schemes
  - Repeat same experiments at different time of day
  - Average results over 10 experiments
- For each experiment
  - All members join at the same time
  - Single source, CBR traffic
  - Each experiment lasts for 20 minutes

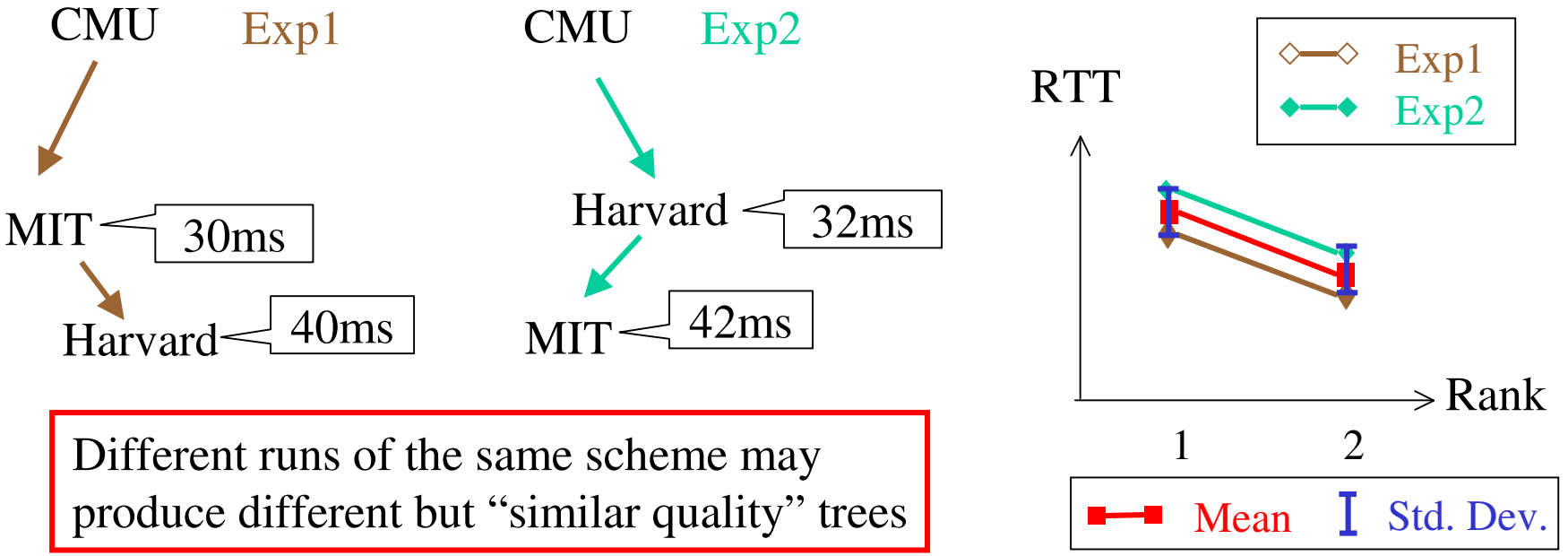
Application Level Metrics

- Bandwidth (throughput) observed by each receiver
- RTT between source and each receiver along overlay



These measurements include queueing and processing delays at end systems

Performance of Overlay Scheme



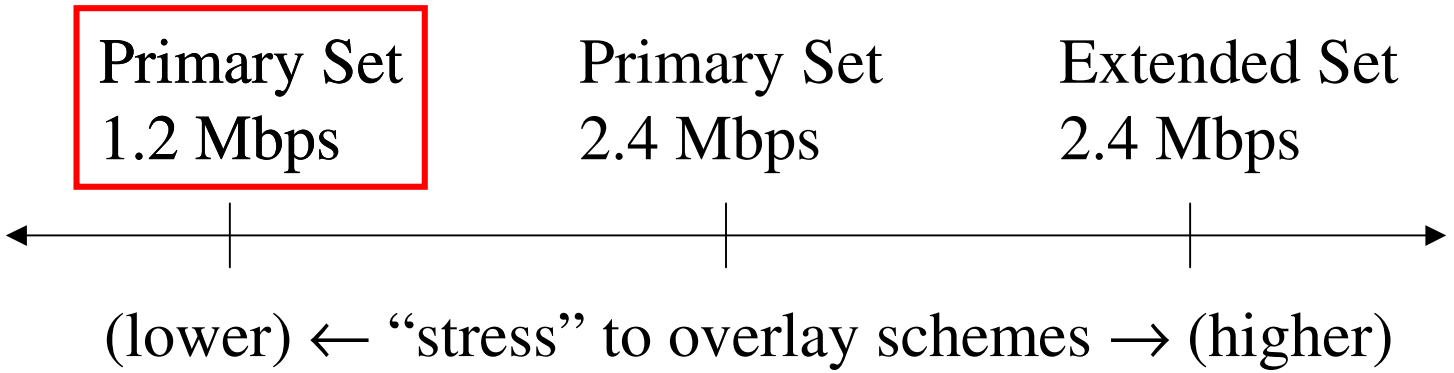
"Quality" of overlay tree produced by a scheme

- Sort ("rank") receivers based on performance
- Take mean and std. dev. on performance of same rank across multiple experiments
- Std. dev. shows variability of tree quality

Factors Affecting Performance

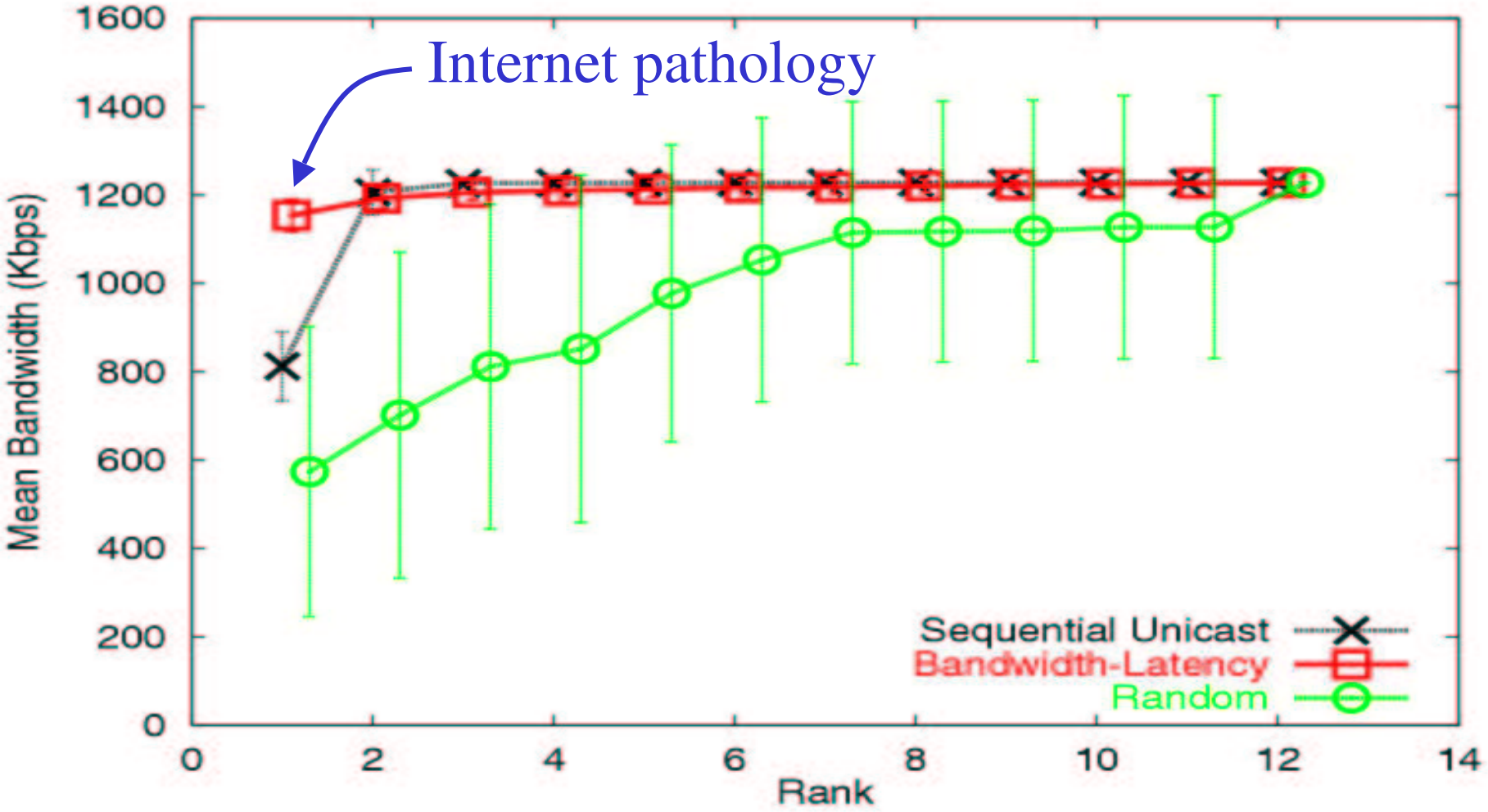
- Heterogeneity of host set
  - *Primary Set*: 13 university hosts in U.S. and Canada
  - *Extended Set*: 20 hosts, which includes hosts in *Primary Set* , Europe, `Asia, and behind ADSL
- Source rate
  - Fewer Internet paths can sustain higher source rate

Three Scenarios Considered



- Does ESM work in different scenarios?
- How do different schemes perform under various scenarios?

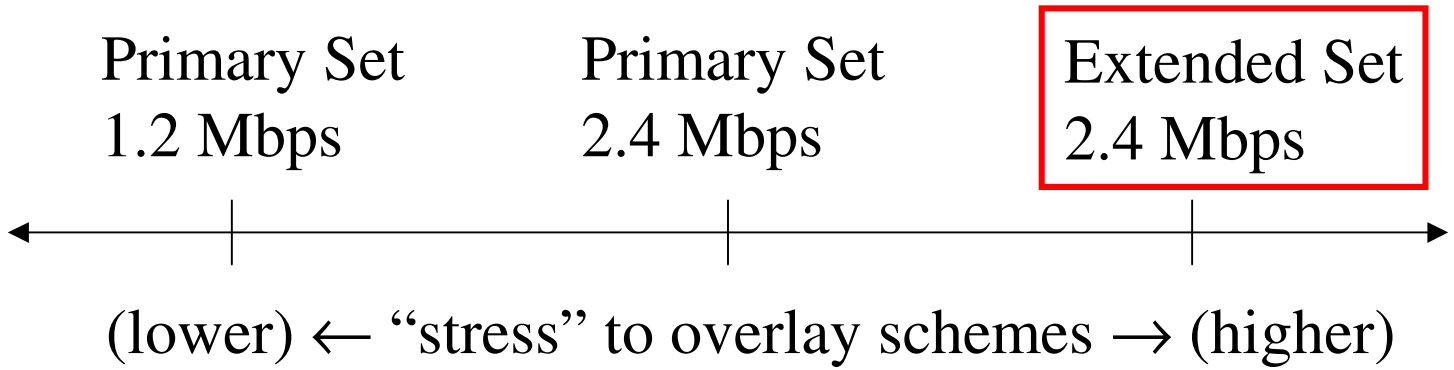
BW, Primary Set, 1.2 Mbps



Naïve scheme performs poorly even in a less "stressful" scenario

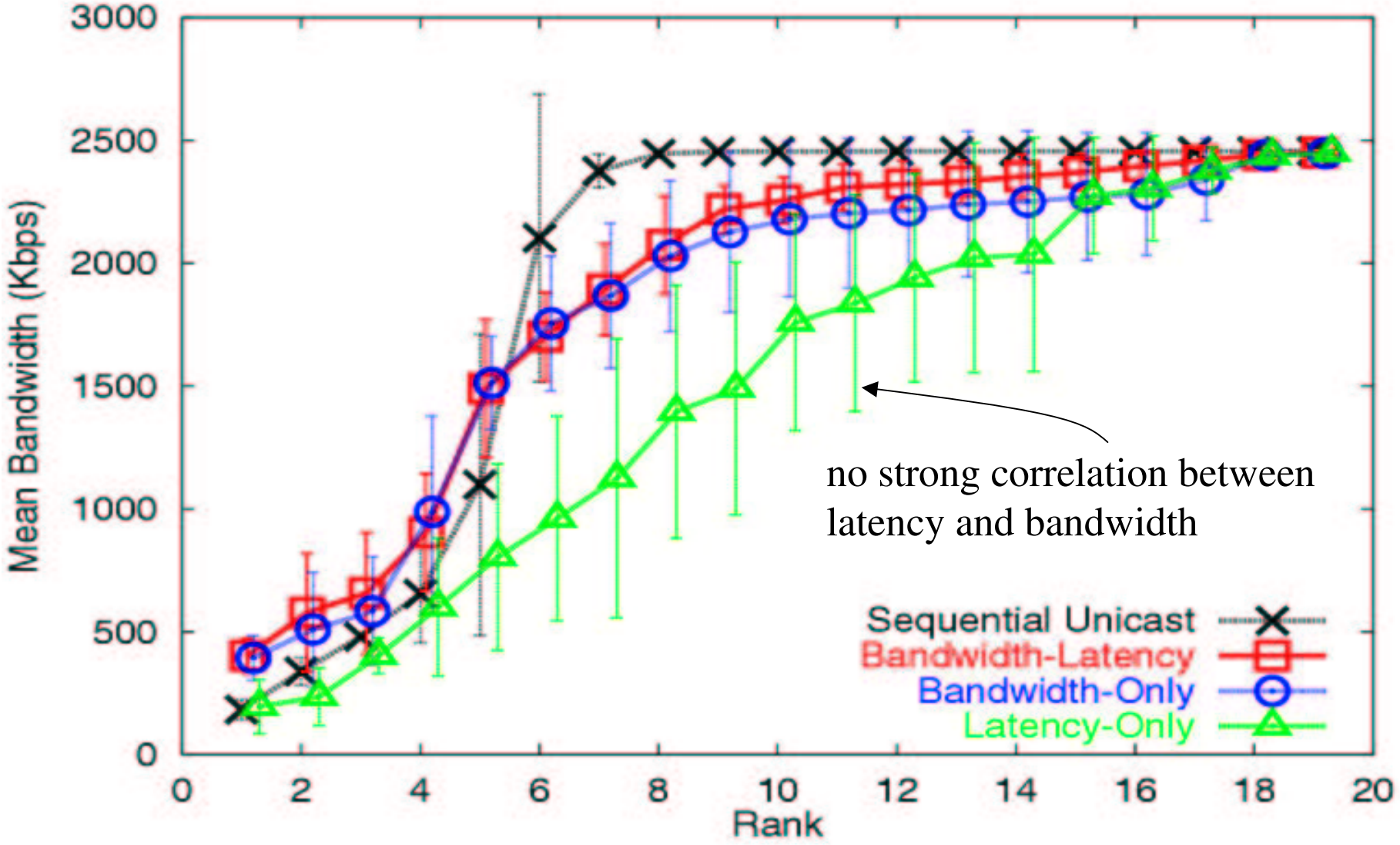
RTT results show similar trend

Scenarios Considered



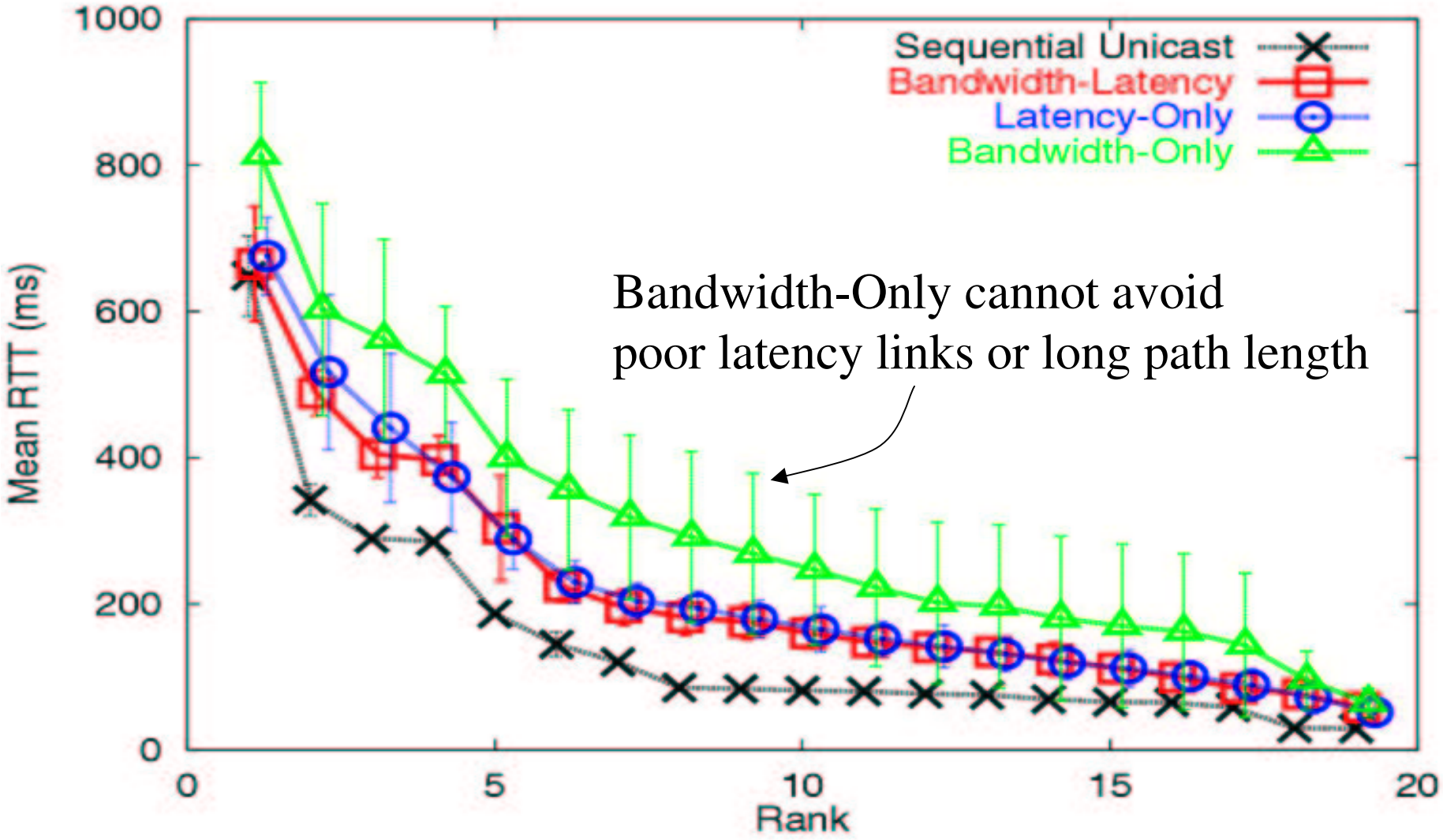
- Does an overlay approach continue to work under a more "stressful" scenario?
- Is it sufficient to consider just a single metric?
  - *Bandwidth-Only, Latency-Only*

BW, Extended Set, 2.4 Mbps



Optimizing only for latency has poor bandwidth performance

RTT, Extended Set, 2.4Mbps



Optimizing only for bandwidth has poor latency performance

Summary so far…

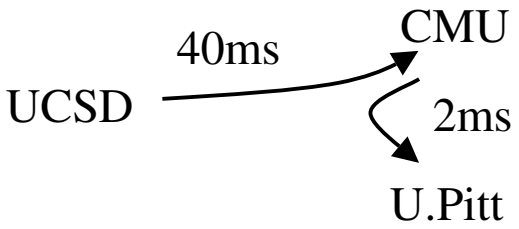
- For best application performance: adapt dynamically to both latency and bandwidth metrics
- *Bandwidth-Latency* performs comparably to IP Multicast (*Sequential-Unicast)*

- What is the network cost and overhead?

Resource Usage (RU)

Captures consumption of network resource of overlay tree

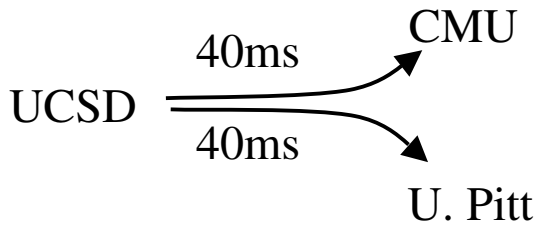
- Overlay link RU = propagation delay
- Tree RU = sum of link RU



Efficient (RU = 42ms)

Scenario: Primary Set, 1.2 Mbps
(normalized to IP Multicast RU)

IP Multicast	1.0
Bandwidth-Latency	1.49
Random	2.24
Naïve Unicast	2.62



Inefficient (RU = 80ms)

Protocol Overhead

$$\text{Protocol overhead} = \frac{\text{total non-data traffic (in bytes)}}{\text{total data traffic (in bytes)}}$$

- Results: Primary Set, 1.2 Mbps
  - Average overhead = 10.8%
  - 92.2% of overhead is due to bandwidnth probe
- Current scheme employs active probing for available bandwidth
  - Simple heuristics to eliminate unnecessary probes
  - Focus of our current research

Contribution

- First detailed Internet evaluation to show the feasibility of End System Multicast architecture
  - Study in context of a/v conferencing
  - Performance comparable to IP Multicast
- Impact of metrics on overlay performance
  - For best performance: use both latency and bandwidth
- More info: http://www.cs.cmu.edu/~narada

Discussion (1)

- Peer-to-peer versus proxy based architecture

Peer-to-peer	Proxy based
Distributed	Share network information , history across groups
Scalable to number of groups	Long term connection, more stable
	Manage resource allocation among groups

Discussion (2)

- Multipath framework where each recipient gets data from the source along multiple paths, with a fraction of the data flowing along any given path
  - Any individual path doesn't radically affect overall performance
  - Receive data while monitoring

Discussion (3)

- Rigorous change detection algorithm
  - On the Constancy of Internet Path Properties
    - www.aciri.org/vern/imw2001-papers/38.ps.gz
  - Idea of more formally identifying changes
    - www.variation.com/cpa/tech/changepoint.html

14. Fault-tolerant replication management in large-scale distributed storage systems

Richard Golding
Storage Systems Program, Hewlett Packard Labs
golding@hpl.hp.com

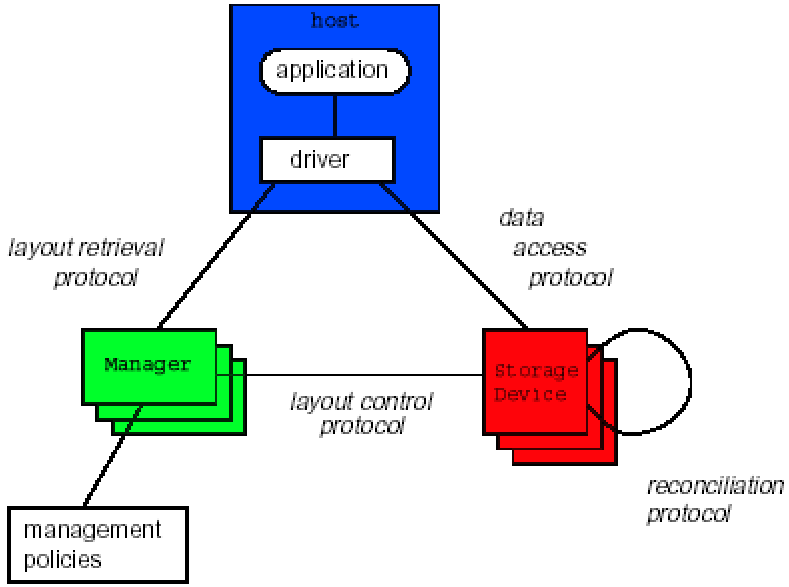
Elizabeth Borowsky
Computer Science Dept., Boston College
borowsky@cs.bc.edu

Introduction

- **Palladio -** solution for detecting, handling, and recovering from both small- and large-scale failures in a distributed storage system.
- **Palladio -** provides virtualized data storage services to applications via set of ***virtual stores***, which are structured as a logical array of bytes into which applications can write and read data. The store's ***layout*** maps each byte in its address space to an address on one or more devices.
- **Palladio -** ***storage devices*** take an active role in the recovery of the stores they are part of. ***Managers*** keep track of the virtual stores in the system, coordinating changes to their layout and handling recovery from failure.

•Provide robust read and write access to data in virtual stores.

- •Atomic and serialized read and write access.
- •Detect and recover from failure.
- •Accommodate layout changes.



Palladio implementation structure

Entities

Hosts
Stores
Managers
Management policies

Protocols

Layout Retrieval protocol
Data Access protocol
Reconciliation protocol
Layout Control protocol

Protocols

Access protocol allows hosts to read and write data on a storage device as long as there are no failures or layout changes for the virtual store. It must provide serialized, atomic writes that can span multiple devices.

Layout retrieval protocol allows hosts to obtain the current layout of a virtual store — the mapping from the virtual store's address space onto the devices that store parts of it.

Reconciliation protocol runs between pairs of devices to bring them back to consistency after a failure.

Layout control protocol runs between managers and devices — maintains consensus about the layout and failure status of the devices, and in doing so coordinates the other three protocols.

Layout Control Protocol

The layout control protocol tries to maintain agreement between a store's manager and the storage devices that hold the store.

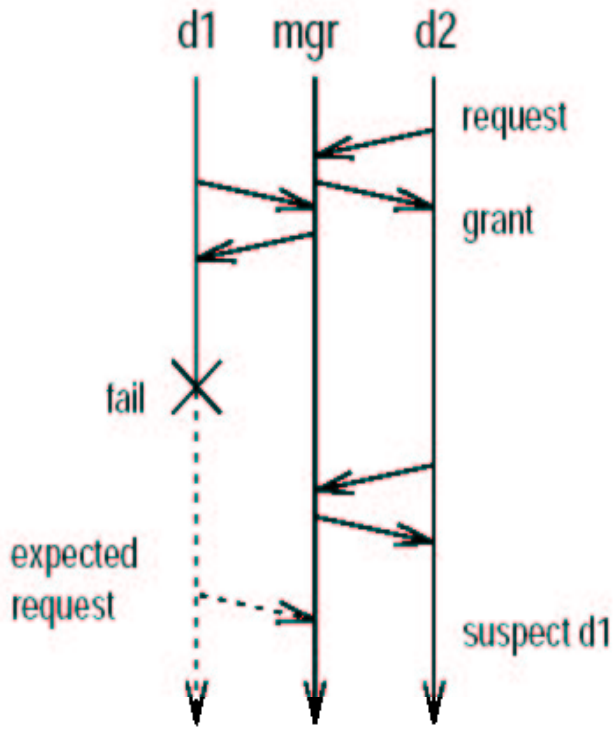
- The layout of data onto storage devices
- The identity of the store's *active manager*.

The notion of ***epochs***

- The layout and manager are fixed during each epoch
- Epochs are numbered
- Epoch transitions
- Device leases acquisition and renewal
- Device leases used to detect possible failure.

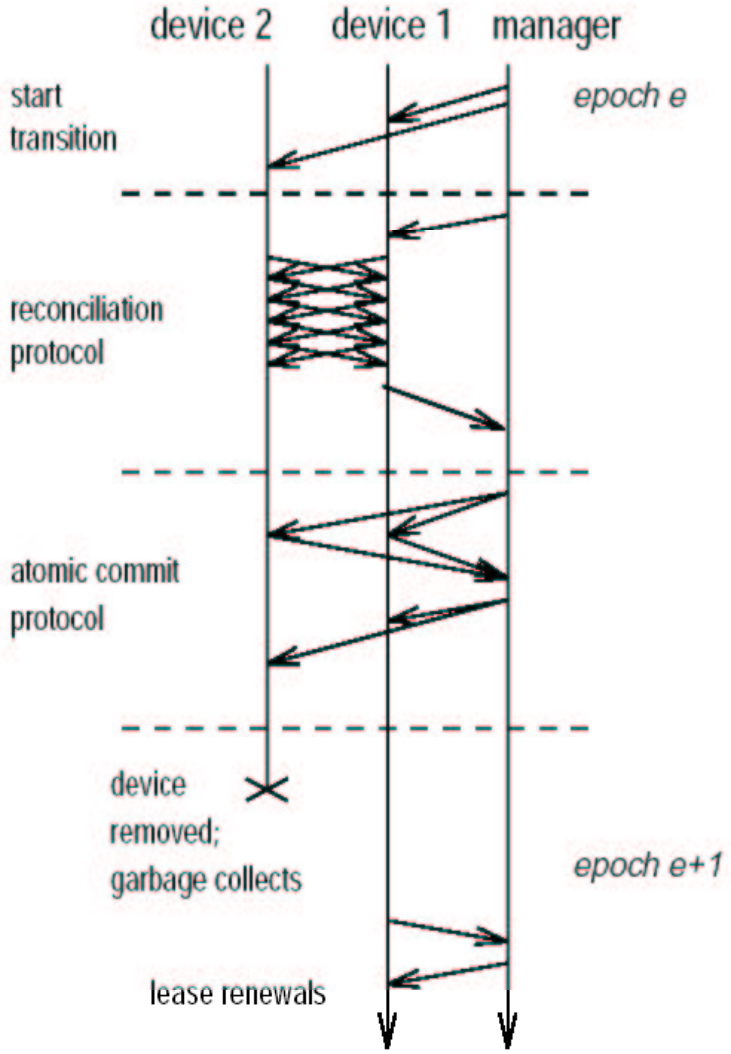
Operation during an epoch

- The manager has quorum and coverage of devices.
- Periodic lease renewal
  - »In case a device fails to report and try to renew its lease, the manager considers it failed
  - »In case the manager fails to renew the lease, the device considers the manager failed and starts a *manager recovery sequence*
- When the manager looses quorum or coverage the epoch ends and a state of epoch transition is entered.



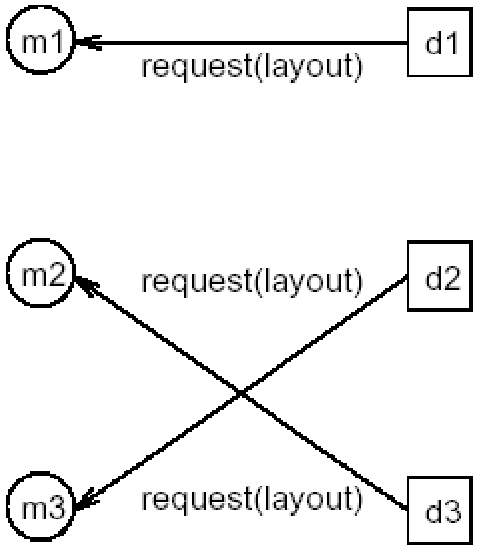
Epoch transition

- Transaction initiation
- Reconciliation
- Transaction commitment
- Garbage collection

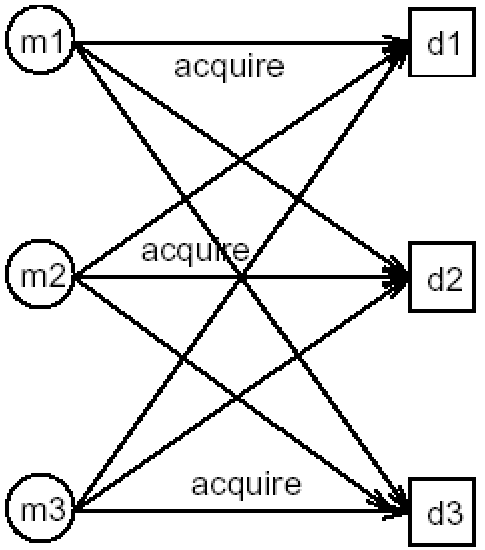


The recovery sequence

•*Initiation* - querying a recovery manager with the current layout and epoch number



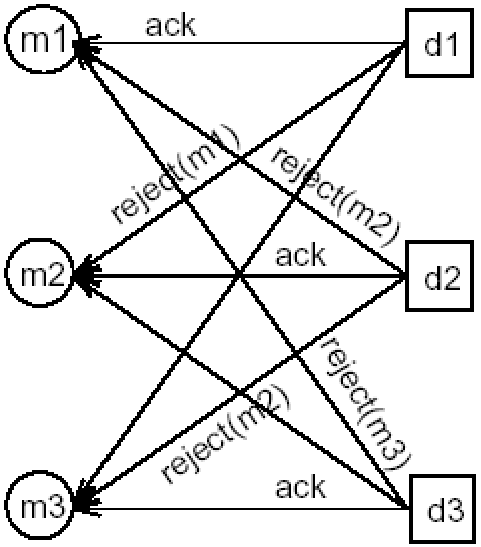
Step 1: devices request recovery



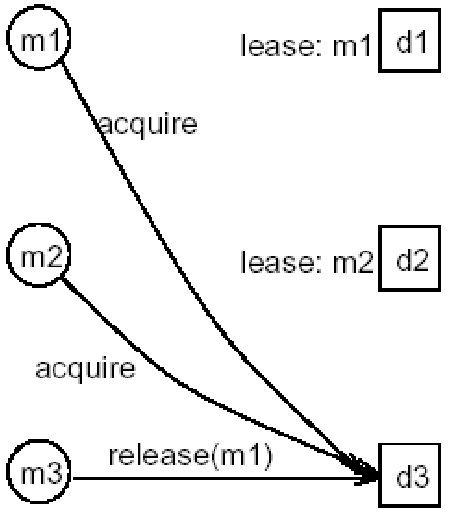
Step 2: managers try to acquire devices

The recovery sequence (continued)

•*Contention* - managers struggle to obtain quorum and coverage and to become active managers for the store - (recovery leases, acks and rejections)



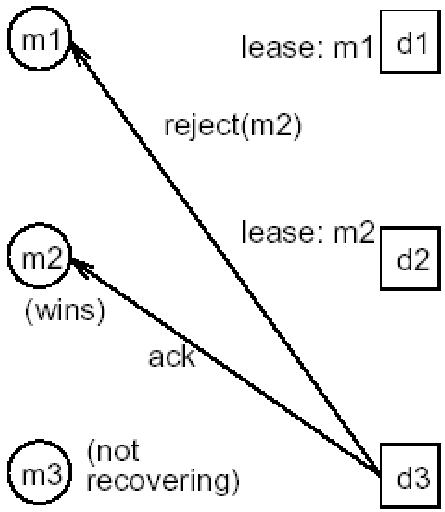
Step 3: devices accept first lease, reject others



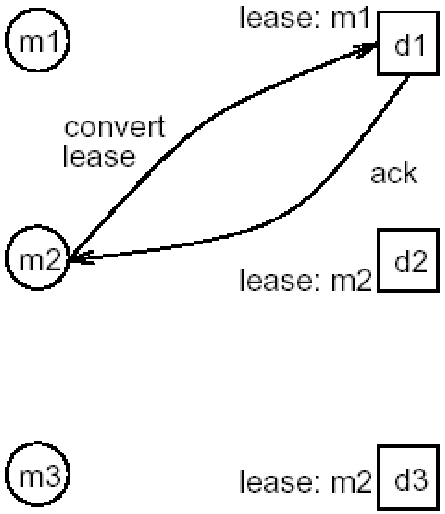
Step 4: least precedence manager gives up, others try again

The recovery sequence (continued)

- *Completion* - setting correct recovery leases & starting epoch transition
- *Failure* - failure of devices and managers during recovery



Step 5: remaining device accepts one lease, rejects other



Step 6: manager 2 wins contention, converts leases on other devices

Extensions

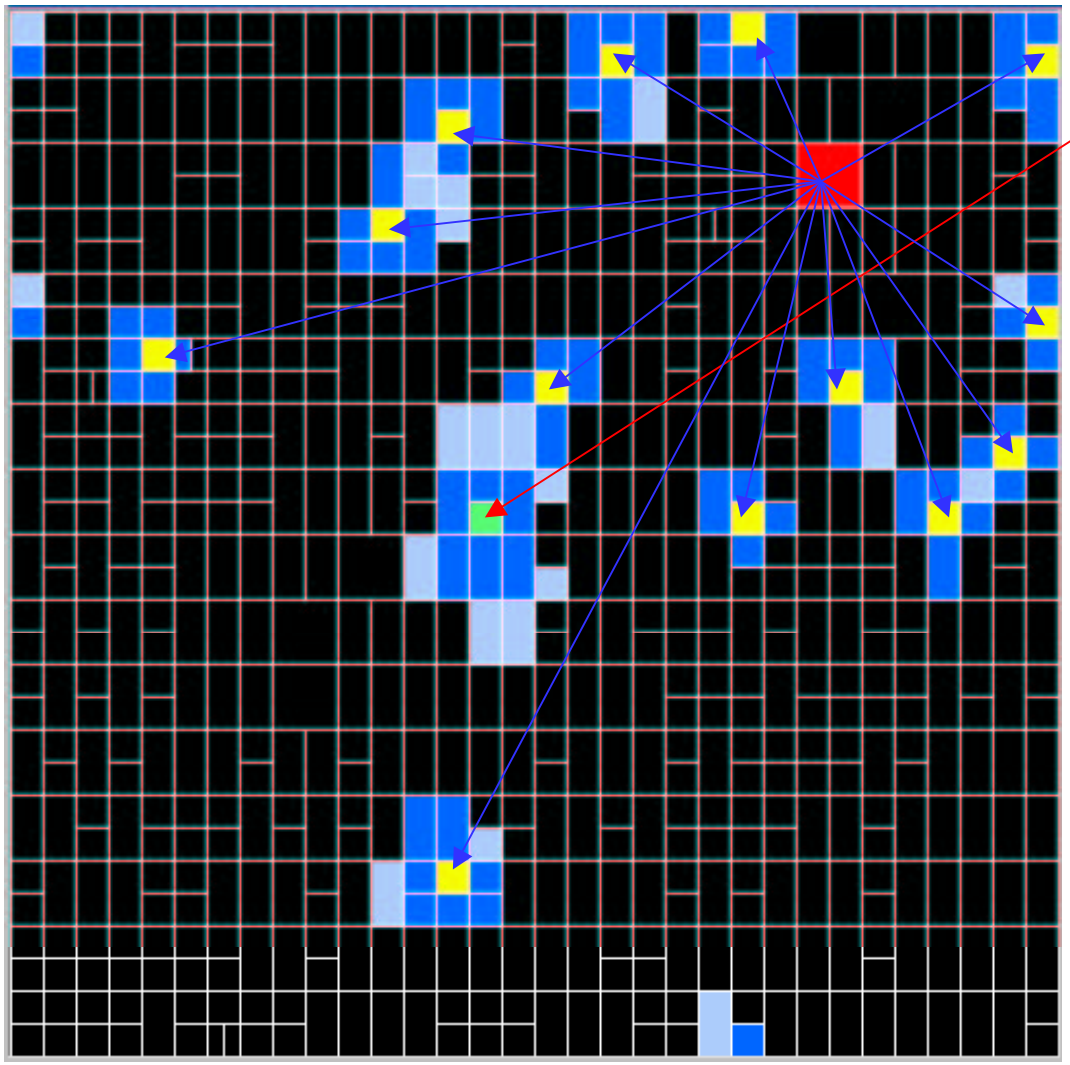
- *Single manager* **v.s.** *Multiple managers*
- *Whole devices* **v.s.** *Device parts (chunks)*
- *Reintegrating devices*
- *Synchrony model (future)*
- *Failure suspectors (future)*

Conclusions & recap

Palladio - Replication management system featuring

» *Modular protocol design*
» ***Active device participation***
» *Distributed management function*
» *Coverage and quorum condition*

Application example



Very popular content

Popularity indicator

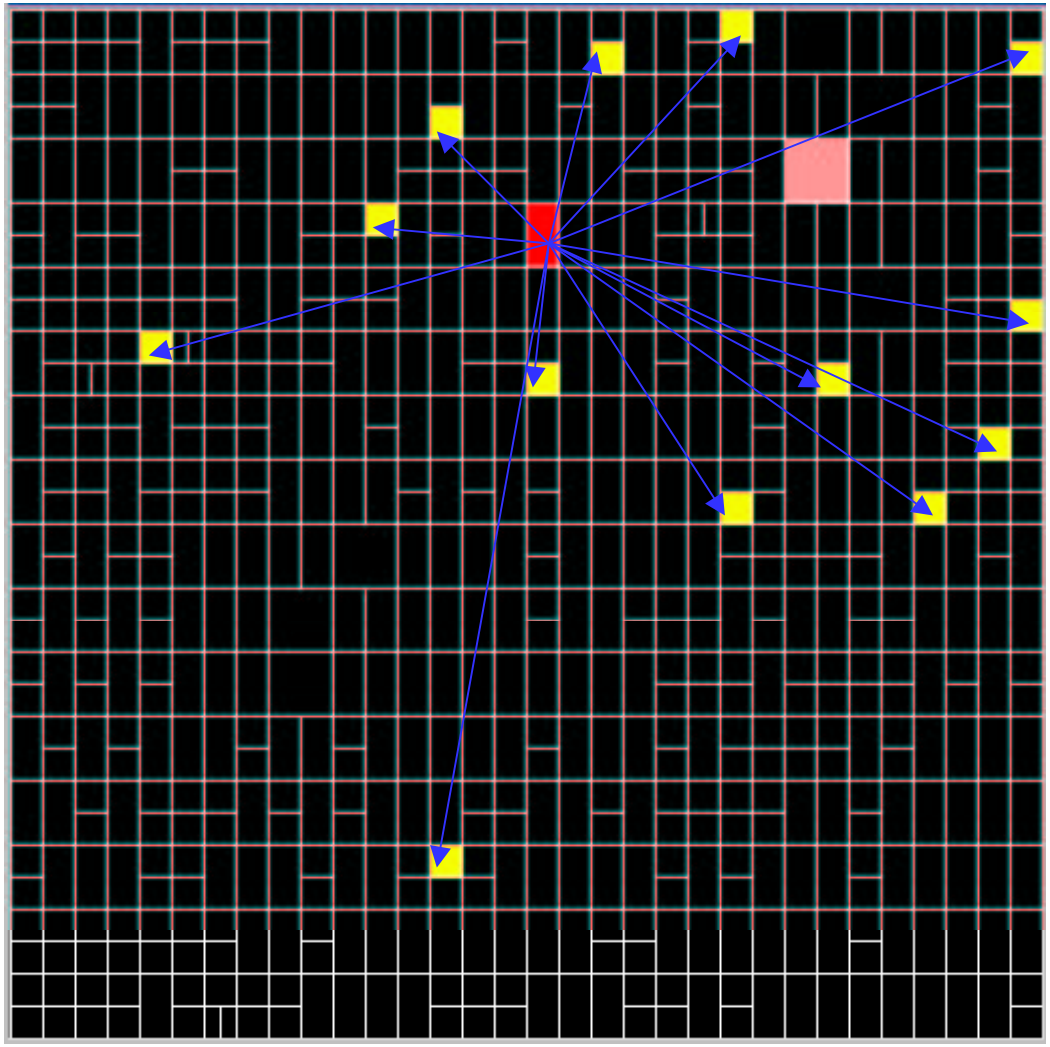
Manager node

ID=hash<FileID, MGR>

Storage nodes

ID=hash<FileID, STR, n>

Application example - benefits



Stable manager node

Stable storage nodes

- Self-manageable storage
- Increased availability
- Popularity is hard to fake
- Less per node load
- Could be applied recursively (?)

Wrapup discussion questions (1):

- What is a peer-peer network (what is not a peer-to-peer network?). Necessary:
  - *every* node is designed to (but may not by user choice) *provide some service* that helps other nodes in the network get service
  - no 1-N service providing
  - each node potentially has the same responsibility, functionality (maybe nodes can be polymorhpic)
    - corollary: by design, nothing (functionally) prevents two nodes from communicating directly
  - some applications (e.g., Napster) are a mix of peer-peer and centralized (lookup is centralized, file service is peer-peer) [recursive def. of peer-peer]
  - (logical connectivity rather than physical connectivity) routing will depend on service and data

Overlays?

- What is the relationship between peer-peer and application overlay networks?
  - Peer-peer and application overlays are different things. It is possible for an application level overlay to be built using peer-peer (or vice versa) but not always necessary
  - Overlay: in a wired net: if two nodes can communicate in the overlay using a path that is *not* the path the network level routing would define for them. Logical network on top of underlying network
    - source routing?
  - Wireless ad hoc nets – what commonality is there REALLY?

Wrapup discussion questions (2):

- What were the best p2p idea
- Vote now (and should it be a secret ballot usign Eternity☺

Wrapup discussion questions (3):

- Is ad hoc networking a peer-peer application?
  - Yes (30-1)
- Why peer-peer over client-server?
  - A well-deigned p2p provides better "scaability"
- Why client-server of peer-peer
  - peer-peer is harder to make reliable
  - availability different from client-server (p2p is more often at least partially "up")
  - more trust is required
- If all music were free in the future (and organized), would we have peer-peer.
  - Is there another app: ad hoc networking, any copyrighted data, peer-peer sensor data gathering and retrieval, simulation
- Evolution #101 – what can we learn about systems?