P561: Network Systems Week 7: Finding content Multicast

Tom Anderson
Ratul Mahajan

TA: Colin Dixon

Today

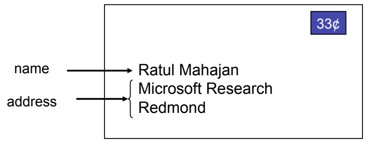
Finding content and services

- Infrastructure hosted (DNS)
- Peer-to-peer hosted (Napster, Gnutella, DHTs)

Multicast: one to many content dissemination

- Infrastructure (IP Multicast)
- Peer-to-peer (End-system Multicast, Scribe)

Names and addresses



Names: identifiers for objects/services (high level)
Addresses: locators for objects/services (low level)
Resolution: name → address

But addresses are really lower-level names

- e.g., NAT translation from a virtual IP address to physical IP, and IP address to MAC address

Naming in systems

Ubiquitous

- Files in filesystem, processes in OS, pages on the Web

Decouple identifier for object/service from location

- Hostnames provide a level of indirection for IP addresses

Naming greatly impacts system capabilities and performance

- Ethernet addresses are a flat 48 bits
  - flat → any address anywhere but large forwarding tables
- IP addresses are hierarchical 32/128 bits
  - hierarchy → smaller routing tables but constrained locations

Key considerations

For the namespace

- Structure

For the resolution mechanism

- Scalability
- Efficiency
- Expressiveness
- Robustness

Internet hostnames

Human-readable identifiers for end-systems

Based on an administrative hierarchy

- E.g., june.cs.washington.edu, www.yahoo.com
- **You** cannot name your computer foo.yahoo.com

In contrast, (public) IP addresses are a fixed-length binary encoding based on network position

- 128.95.1.4 is june's IP address, 209.131.36.158 is one of www.yahoo.com's IP addresses
- Yahoo cannot pick any address it wishes

Original hostname system

When the Internet was really young …

Flat namespace
- Simple (host, address) pairs

Centralized management
- Updates via a single master file called HOSTS.TXT
- Manually coordinated by the Network Information Center (NIC)

Resolution process
- Look up hostname in the HOSTS.TXT file
- Works even today: (c:/WINDOWS/system32/drivers)/etc/hosts

Problems with the original system

Coordination
- Between all users to avoid conflicts
- E.g., everyone likes a computer named Mars

Inconsistencies
- Between updated and old versions of file

Reliability
- Single point of failure

Performance
- Competition for centralized resources

Domain Name System (DNS)

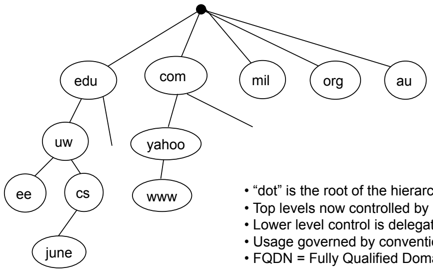
Developed by Mockapetris and Dunlap, mid-80's

Namespace is hierarchical
- Allows much better scaling of data structures
- e.g., *root* → edu → washington → cs → june

Namespace is distributed
- Decentralized administration and access
- e.g., june managed by cs.washington.edu

Resolution is by query/response
- With replicated servers for redundancy
- With heavy use of caching for performance

DNS Hierarchy



- "dot" is the root of the hierarchy
- Top levels now controlled by ICANN
- Lower level control is delegated
- Usage governed by conventions
- FQDN = Fully Qualified Domain Name

Name space delegation

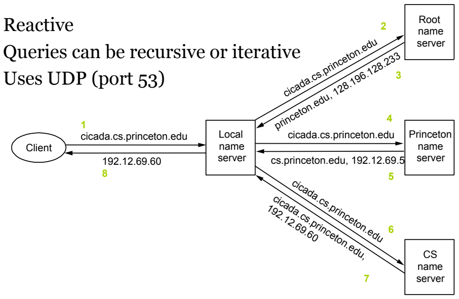
Each organization controls its own name space ("zone" = subtree of global tree)
- each organization has its own nameservers
  - replicated for availability
- nameservers translate names within their organization
  - client lookup proceeds step-by-step
- example: washington.edu
  - contains IP addresses for all its hosts (www.washington.edu)
  - contains pointer to its subdomains (cs.washington.edu)

DNS resolution

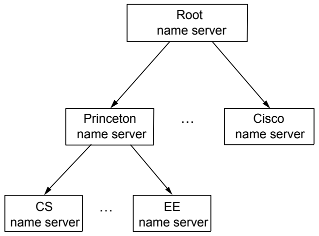
Reactive

Queries can be recursive or iterative

Uses UDP (port 53)



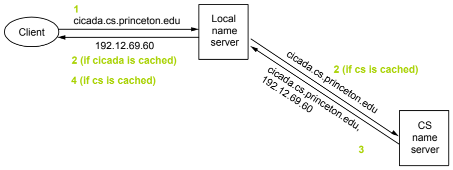
Hierarchy of nameservers



DNS performance: caching

DNS query results are cached at local proxy

- quick response for repeated translations
- lookups are the rare case
- vastly reduces load at the servers
- what if something new lands on slashdot?



DNS cache consistency

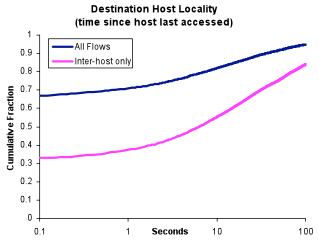
How do we keep cached copies up to date?

- DNS entries are modified from time to time
  - to change name → IP address mappings
  - to add/delete names

Cache entries invalidated periodically

- each DNS entry has time-to-live (TTL) field: how long can the local proxy can keep a copy
- if entry accessed after the timeout, get a fresh copy from the server
- how do you pick the TTL?
- how long after a change are all the copies updated?

DNS cache effectiveness



Traffic seen on UW's access link in 1999

Negative caching in DNS

Pro: traffic reduction

- Misspellings, old or non-existent names
- "Helpful" client features

Con: what if the host appears?

Status:

- Optional in original design
- Mandatory since 1998

DNS traffic in the wide-area

Study	% of DNS packets
Danzig, 1990	14%
Danzig, 1992	8%
Frazer, 1995	5%
Thomson, 1997	3%

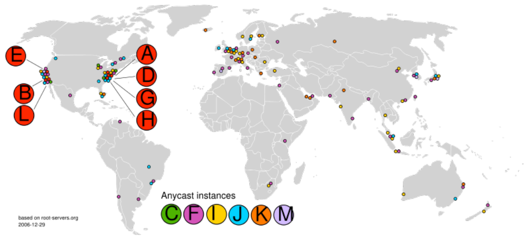
DNS bootstrapping

Need to know IP addresses of root servers before we can make any queries

Addresses for 13 root servers ([a-m].root-servers.net) handled via initial configuration

- Cannot have more than 13 root server IP addresses

DNS root servers



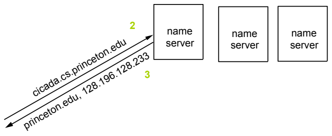
123 servers as of Dec 2006

DNS availability

What happens if DNS service is not working?

DNS servers are replicated

- name service available if at least one replica is working
- queries load balanced between replicas



Building on the DNS

Email: ratul@microsoft.com

- DNS record for ratul in the domain microsoft.com, specifying where to deliver the email

Uniform Resource Locator (URL) names for Web pages

- e.g., www.cs.washington.edu/homes/ratul
- Use domain name to identify a Web server
- Use "/" separated string for file name (or script) on the server

DNS evolution

Static host to IP mapping

- What about mobility (Mobile IP) and dynamic address assignment (DHCP)?
- Dynamic DNS

Location-insensitive queries

- Many servers are geographically replicated
- E.g., Yahoo.com doesn't refer to a single machine or even a single location; want closest server
- Next week

Security (DNSSec)

Internationalization

DNS properties (summary)

Nature of the namespace	Hierarchical; flat at each level
Scalability of resolution	High
Efficiency of resolution	Moderate
Expressiveness of queries	Exact matches
Robustness to failures	Moderate

Peer-to-peer content sharing

Want to share content among large number of users; each serves a subset of files

- need to locate which user has which files

Question: Would DNS be a good solution for this?

Napster (directory-based)

Centralized directory of all users offering each file

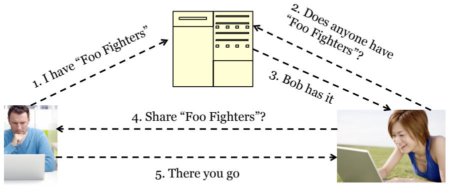
Users register their files

Users make requests to Napster central

Napster returns list of users hosting requested file

Direct user-to-user communication to download files

Naptser illustration



Naptser vs. DNS

	Napster	DNS
Nature of the namespace	Multi-dimensional	Hierarchical; flat at each level
Scalability	Moderate	High
Efficiency of resolution	High	Moderate
Expressiveness of queries	High	Exact matches
Robustness to failures	Low	Moderate

Gnutella (crawl-based)

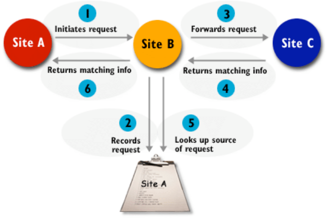
Can we locate files without a centralized directory?

- for legal and privacy reasons

Gnutella

- organize users into ad hoc graph
- flood query to all users, in breadth first search
  - use hop count to control depth
- if found, server replies back through path of servers
- client makes direct connection to server to get file

Gnutella illustration



Gnutella vs. DNS

	Gnutella	DNS
Nature of the namespace	Multi-dimensional	Hierarchical; flat at each level
Scalability	Low	High
Efficiency of resolution	Low	Moderate
Expressiveness of queries	High	Exact matches
Robustness to failures	Moderate	Moderate

Content is not indexed in Gnutella
Trade-off between exhaustiveness and efficiency

Distributed hash tables (DHTs)

Can we locate files without an exhaustive search?

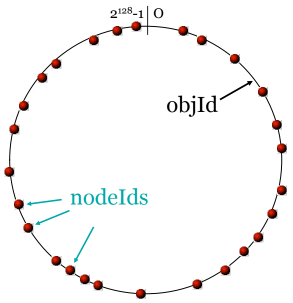
- want to scale to thousands of servers

DHTs (Pastry, Chord, etc.)

- Map servers and objects into an coordinate space
- Objects/info stored based on its key
- Organize servers into a predefined topology (e.g., a ring or a k-dimensional hypercube)
- Route over this topology to find objects

We'll talk about Pastry (with some slides stolen from Peter Druschel)

Pastry: Id space



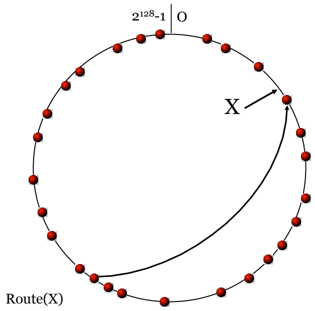
128 bit circular id space

nodeIds (uniform random)

objIds (uniform random)

Invariant: node with numerically closest nodeId maintains object

Pastry: Object insertion/lookup



Msg with key *X* is routed to live node with nodeId closest to X

Problem: complete routing table not feasible

Pastry: Routing

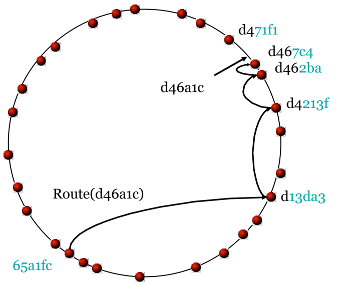
Tradeoff

O(*log N*) routing table size
O(*log N*) message forwarding steps

Pastry: Routing table (# 65a1fc*x*)

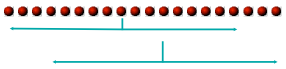
Row 0	*0 x*	*1 x*	*2 x*	*3 x*	*4 x*	*5 x*		*7 x*	*8 x*	*9 x*	*a x*	*b x*	*c x*	*d x*	*e x*	*f x*
Row 1	*6 0 x*	*6 1 x*	*6 2 x*	*6 3 x*	*6 4 x*		*6 6 x*	*6 7 x*	*6 8 x*	*6 9 x*	*6 a x*	*6 b x*	*6 c x*	*6 d x*	*6 e x*	*6 f x*
Row 2	*6 5 0 x*	*6 5 1 x*	*6 5 2 x*	*6 5 3 x*	*6 5 4 x*	*6 5 5 x*	*6 5 6 x*	*6 5 7 x*	*6 5 8 x*	*6 5 9 x*		*6 5 b x*	*6 5 c x*	*6 5 d x*	*6 5 e x*	*6 5 f x*
Row 3	*6 5 a 0 x*		*6 5 a 2 x*	*6 5 a 3 x*	*6 5 a 4 x*	*6 5 a 5 x*	*6 5 a 6 x*	*6 5 a 7 x*	*6 5 a 8 x*	*6 5 a 9 x*	*6 5 a a x*	*6 5 a b x*	*6 5 a c x*	*6 5 a d x*	*6 5 a e x*	*6 5 a f x*

Pastry: Routing



Properties
$\log_{16}$ N steps
O(*log N*) state

Pastry: Leaf sets



Each node maintains IP addresses of the nodes with the L/2 numerically closest larger and smaller nodeIds, respectively.

- routing efficiency/robustness
- fault detection (keep-alive)
- application-specific local coordination

Pastry: Routing procedure

```
if (destination is within range of our leaf set)
        forward to numerically closest member
else
        let $l$ = length of shared prefix
        let $d$ = value of $l$-th digit in $D$'s address
        if ($R_l^d$ exists)
                forward to $R_l^d$
        else
                forward to a known node that
                (a) shares at least as long a prefix
                (b) is numerically closer than this
node
```

Pastry: Performance

Integrity of overlay/ message delivery:

guaranteed unless *L/2* simultaneous failures of nodes with adjacent nodeIds

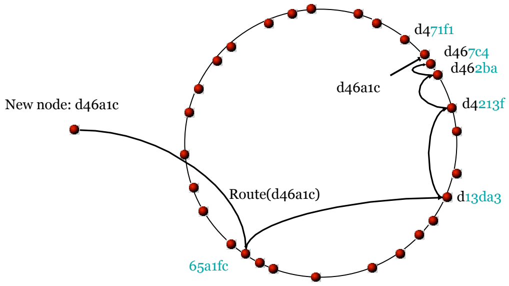
Number of routing hops:

No failures: < $log_{16} N$ expected, 128/4 + 1 max

During failure recovery:

- *O(N)* worst case, average case much better

Pastry: Node addition

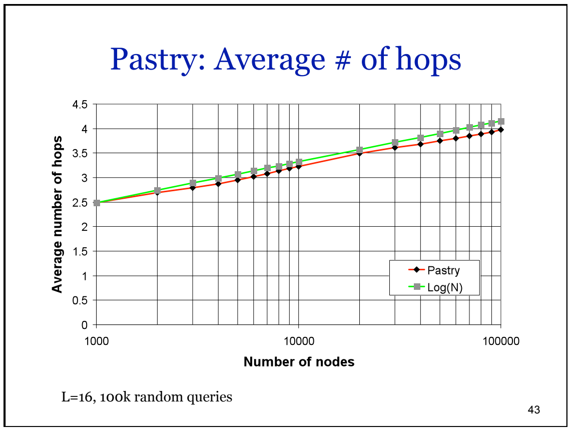


Node departure (failure)

Leaf set members exchange keep-alive messages

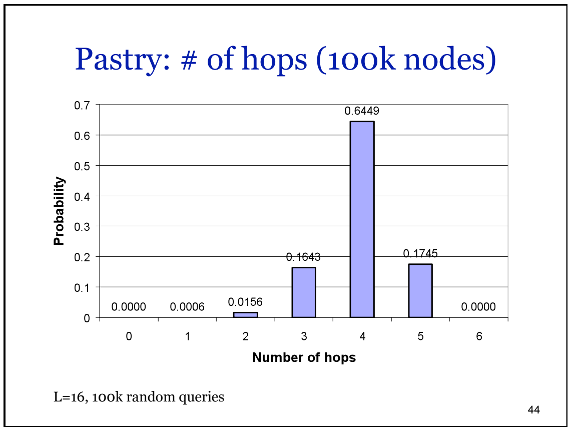
Leaf set repair (eager): request set from farthest live node in set

Routing table repair (lazy): get table from peers in the same row, then higher rows



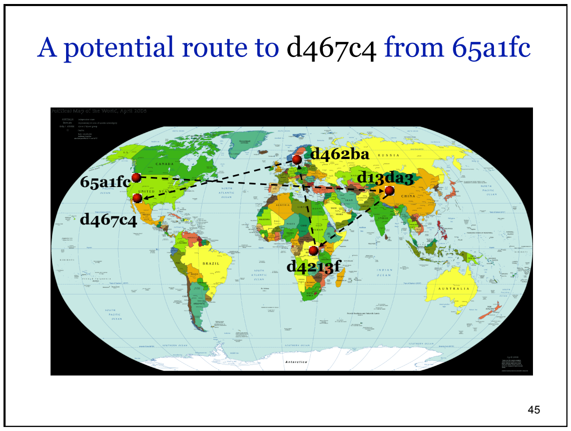
Pastry: Average # of hops

L=16, 100k random queries



Pastry: # of hops (100k nodes)

L=16, 100k random queries



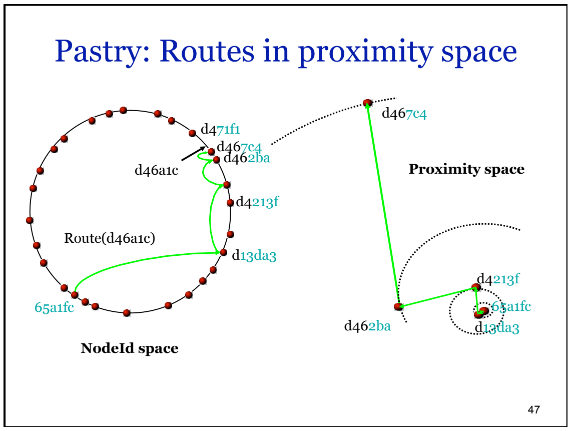
A potential route to d467c4 from 65a1fc

Pastry: Proximity routing

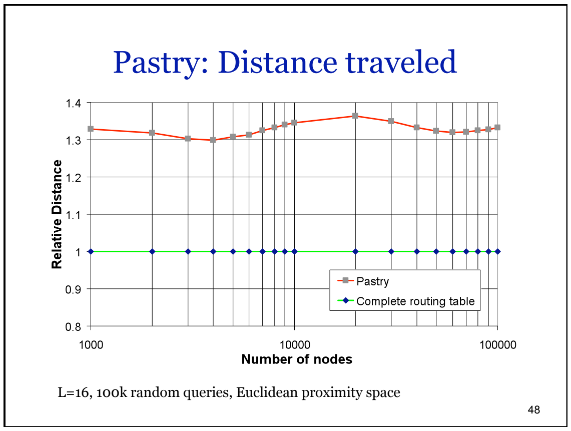
Assumption: scalar proximity metric, e.g. ping delay, # IP hops; a node can probe distance to any other node

Proximity invariant: Each routing table entry refers to a node close to the local node (in the proximity space), among all nodes with the appropriate nodeId prefix.

Locality-related route qualities: Distance traveled, likelihood of locating the nearest replica

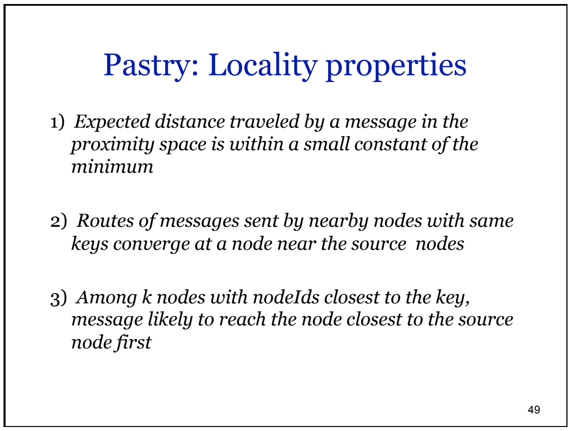


Pastry: Routes in proximity space



Pastry: Distance traveled

L=16, 100k random queries, Euclidean proximity space

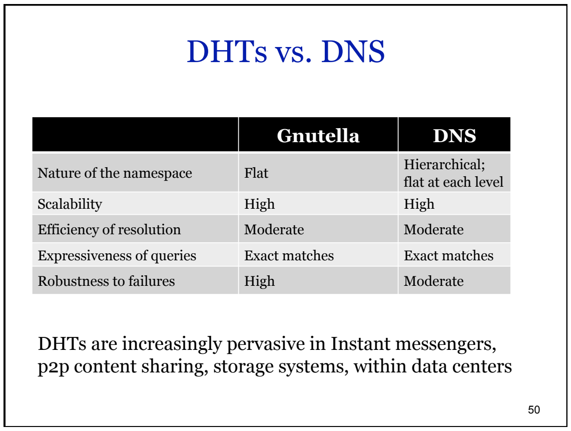


Pastry: Locality properties

1) *Expected distance traveled by a message in the proximity space is within a small constant of the minimum*

2) *Routes of messages sent by nearby nodes with same keys converge at a node near the source nodes*

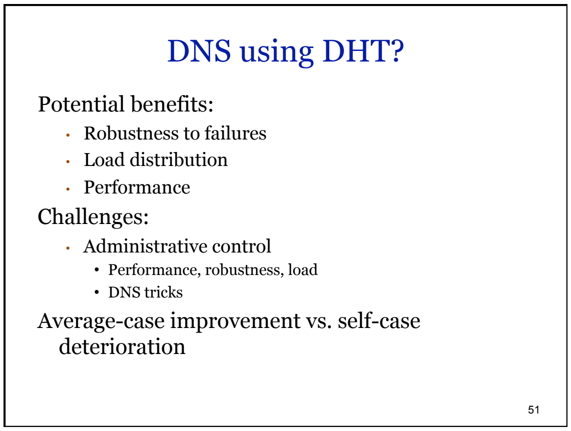
3) *Among k nodes with nodeIds closest to the key, message likely to reach the node closest to the source node first*



DHTs vs. DNS

	Gnutella	DNS
Nature of the namespace	Flat	Hierarchical; flat at each level
Scalability	High	High
Efficiency of resolution	Moderate	Moderate
Expressiveness of queries	Exact matches	Exact matches
Robustness to failures	High	Moderate

DHTs are increasingly pervasive in Instant messengers, p2p content sharing, storage systems, within data centers



DNS using DHT?

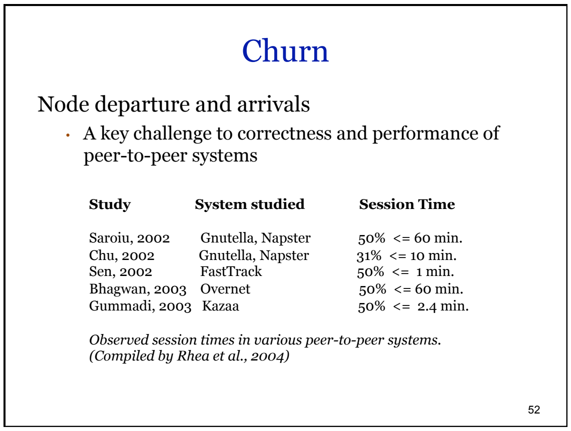
Potential benefits:

- Robustness to failures
- Load distribution
- Performance

Challenges:

- Administrative control
  - Performance, robustness, load
  - DNS tricks

Average-case improvement vs. self-case deterioration



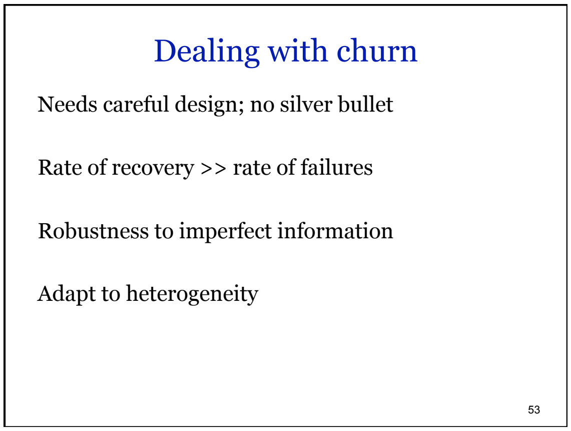
Churn

Node departure and arrivals

- A key challenge to correctness and performance of peer-to-peer systems

Study	System studied	Session Time
Saroiu, 2002	Gnutella, Napster	50% <= 60 min.
Chu, 2002	Gnutella, Napster	31% <= 10 min.
Sen, 2002	FastTrack	50% <= 1 min.
Bhagwan, 2003	Overnet	50% <= 60 min.
Gummadi, 2003	Kazaa	50% <= 2.4 min.

Observed session times in various peer-to-peer systems. (Compiled by Rhea et al., 2004)



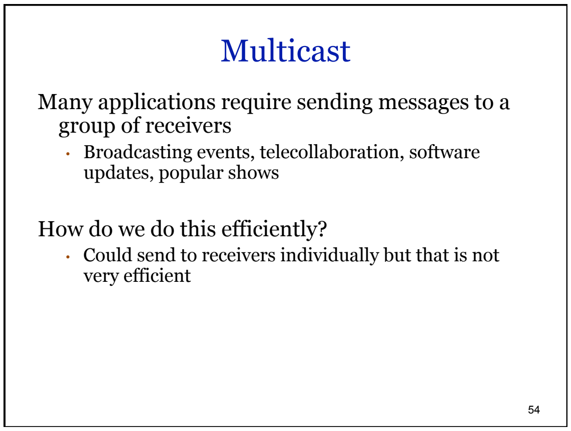
Dealing with churn

Needs careful design; no silver bullet

Rate of recovery >> rate of failures

Robustness to imperfect information

Adapt to heterogeneity



Multicast

Many applications require sending messages to a group of receivers

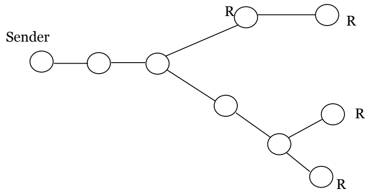
- Broadcasting events, telecollaboration, software updates, popular shows

How do we do this efficiently?

- Could send to receivers individually but that is not very efficient

Multicast efficiency

Send data only once along a link shared by paths to multiple receivers



Two options for implementing multicast

IP multicast

- special IP addresses to represent groups of receivers
- receivers subscribe to specific channels
- modify routers to support multicast sends

Overlay network

- PC routers, forward multicast traffic by tunneling over Internet
- Works on existing Internet, with no router modifications

IP multicast

How to distribute packets across thousands of LANs?

- Each router responsible for its attached LAN
- Hosts declare interest to their routers

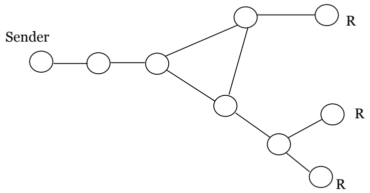
Reduces to:

- How do we forward packets to all interested routers? (DVMRP, M-OSPF, MBone)

Why not simple flooding?

If haven't seen a packet before, forward it on every link but incoming

- routers need to remember each pkt!
- every router gets every packet!



Distance vector multicast

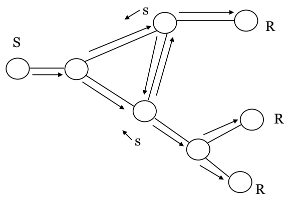
Intuition: unicast routing tables form inverse tree from senders to destination

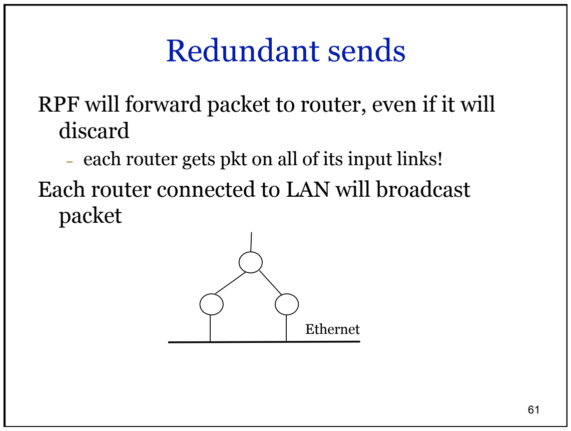
- why not use backwards for multicast?
- Various refinements to eliminate useless transfers

Implemented in DVMRP (Distance Vector Multicast Routing Protocol)

Reverse Path Flooding (RPF)

Router forwards packet from S iff packet came via shortest path back to S



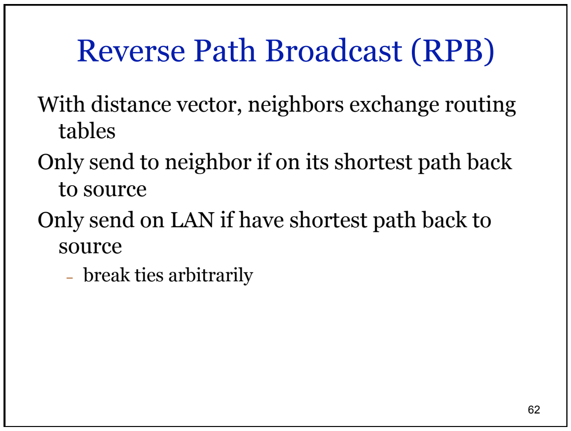


Redundant sends

RPF will forward packet to router, even if it will discard

- each router gets pkt on all of its input links!

Each router connected to LAN will broadcast packet



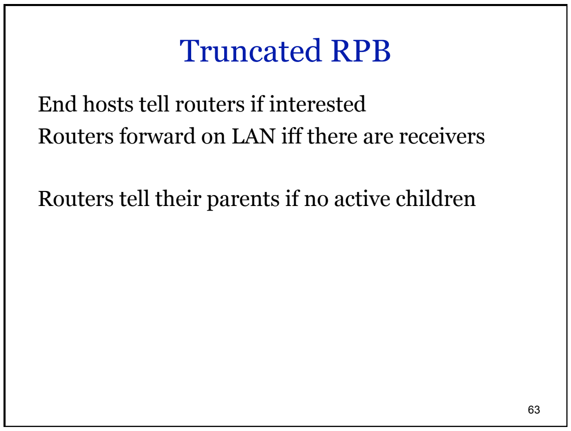
Reverse Path Broadcast (RPB)

With distance vector, neighbors exchange routing tables

Only send to neighbor if on its shortest path back to source

Only send on LAN if have shortest path back to source

- break ties arbitrarily

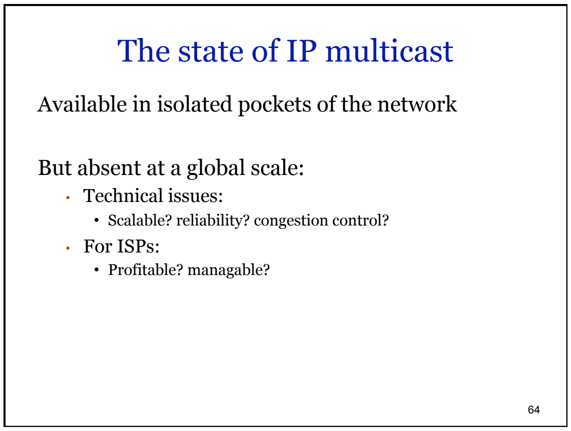


Truncated RPB

End hosts tell routers if interested

Routers forward on LAN iff there are receivers

Routers tell their parents if no active children

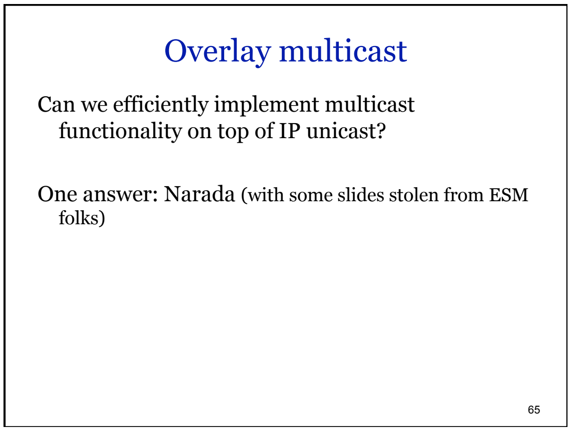


The state of IP multicast

Available in isolated pockets of the network

But absent at a global scale:

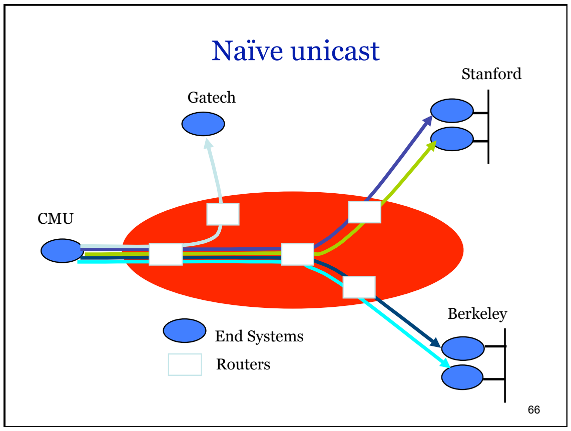
- Technical issues:
  - Scalable? reliability? congestion control?
- For ISPs:
  - Profitable? managable?



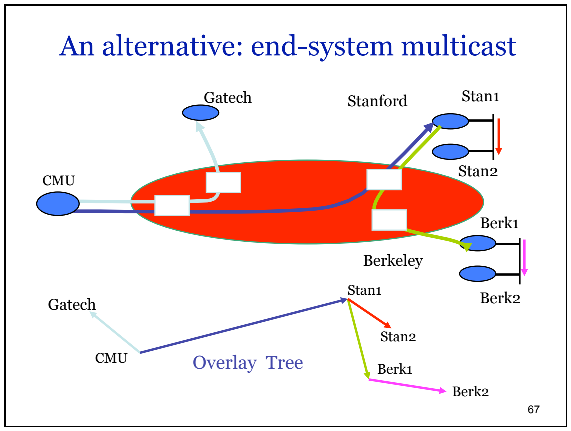
Overlay multicast

Can we efficiently implement multicast functionality on top of IP unicast?

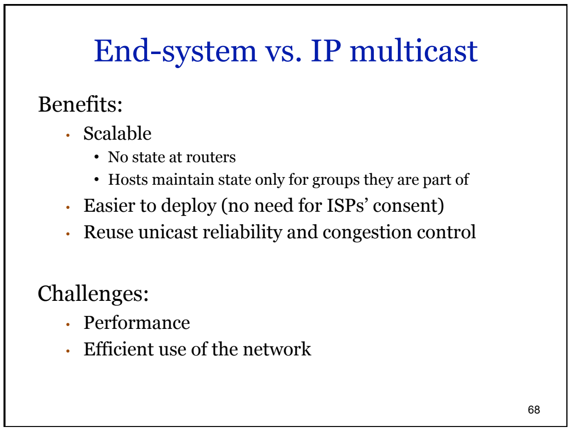
One answer: Narada (with some slides stolen from ESM folks)



Naïve unicast



An alternative: end-system multicast



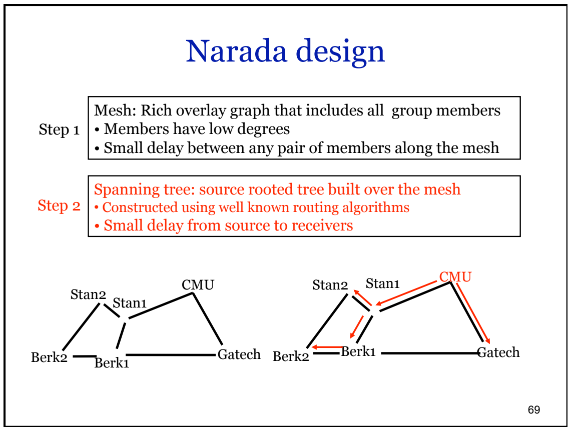
End-system vs. IP multicast

Benefits:

- Scalable
  - No state at routers
  - Hosts maintain state only for groups they are part of
- Easier to deploy (no need for ISPs' consent)
- Reuse unicast reliability and congestion control

Challenges:

- Performance
- Efficient use of the network



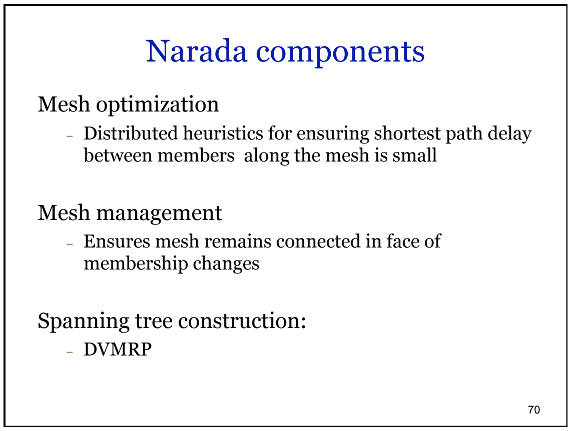
Narada design

Step 1

Mesh: Rich overlay graph that includes all group members
- Members have low degrees
- Small delay between any pair of members along the mesh

Step 2

Spanning tree: source rooted tree built over the mesh
- Constructed using well known routing algorithms
- Small delay from source to receivers



Narada components

Mesh optimization

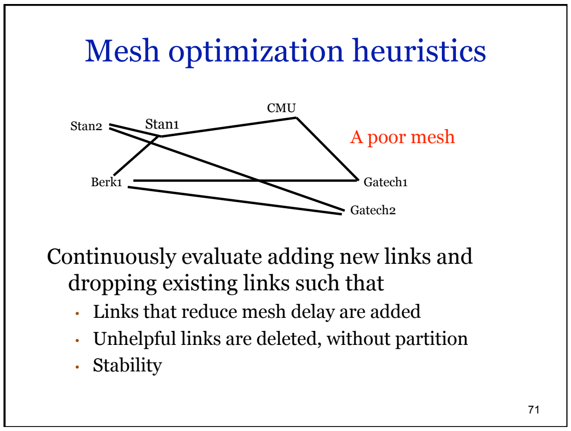
- Distributed heuristics for ensuring shortest path delay between members along the mesh is small

Mesh management

- Ensures mesh remains connected in face of membership changes

Spanning tree construction:

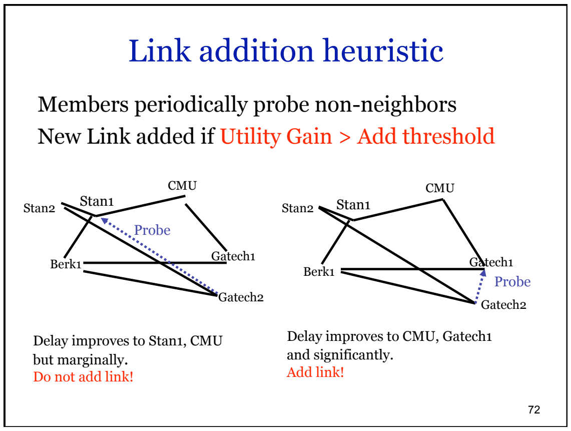
- DVMRP



Mesh optimization heuristics

Continuously evaluate adding new links and dropping existing links such that

- Links that reduce mesh delay are added
- Unhelpful links are deleted, without partition
- Stability



Link addition heuristic

Members periodically probe non-neighbors

New Link added if Utility Gain > Add threshold

Delay improves to Stan1, CMU but marginally.
Do not add link!

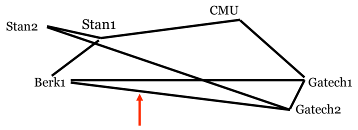
Delay improves to CMU, Gatech1 and significantly.
Add link!

Link deletion heuristic

Members periodically monitor existing links

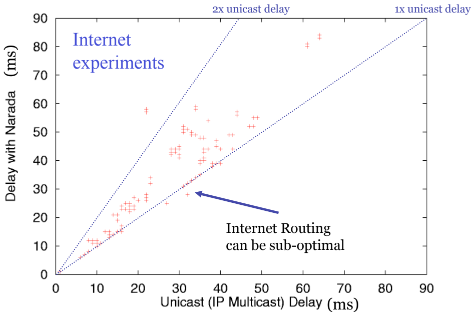
Link dropped if Cost of dropping < Drop threshold

Cost computation and drop threshold is chosen with stability and partitions in mind

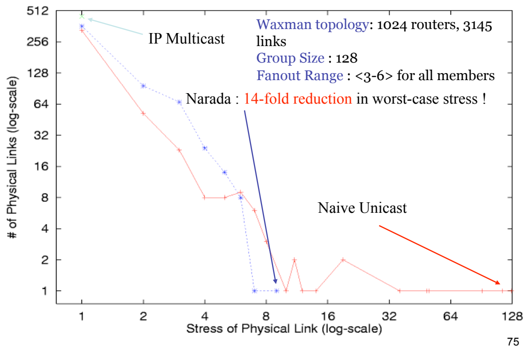


Used by Berk1 to reach only Gatech2 and vice versa.
Drop!!

Narada delay (performance)



Narada stress (efficiency)



Scalable overlay multicast

Can we design an overlay multicast system that scales to very large groups?

One answer: Scribe (with some slides stolen from Kasper Egdø and Morten Bjerre)

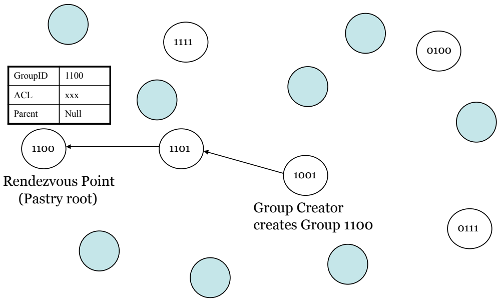
Scribe

Built on top of a DHT (Pastry)

Key ideas:

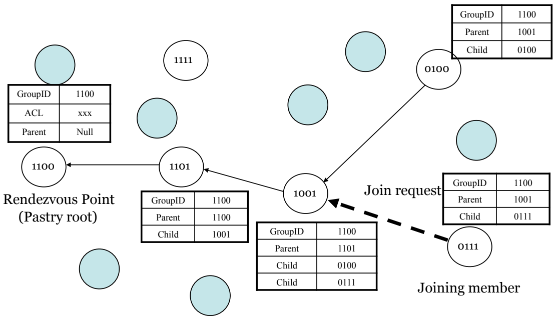
- Treat the multicast group name as a key into the DHT
- Publish info to the key owner, called the Rendezvous point
- Paths from subscribers to the RP form the multicast tree

Creating a group (1100)


1111
0100
1100
1101
1001
0111
Rendezvous Point (Pastry root)
Group Creator creates Group 1100

GroupID	1100
ACL	xxx
Parent	Null

Joining a group



GroupID	1100
Parent	1001
Child	0100

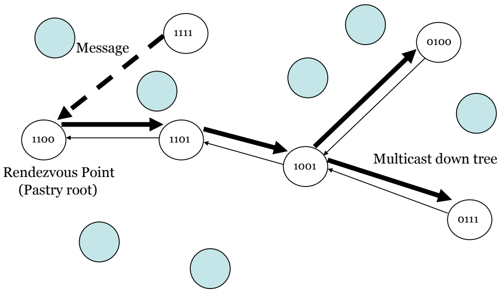
GroupID	1100
ACL	xxx
Parent	Null

GroupID	1100
Parent	1100
Child	1001

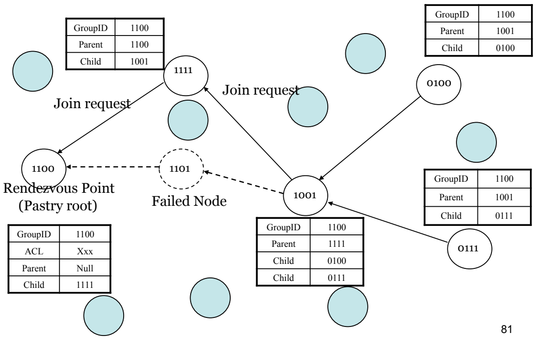
GroupID	1100
Parent	1101
Child	0100
Child	0111

GroupID	1100
Parent	1001
Child	0111

Multicasting



Repairing failures



GroupID	1100
Parent	1100
Child	1001

GroupID	1100
Parent	1001
Child	0100

GroupID	1100
Parent	1001
Child	0111

GroupID	1100
ACL	Xxx
Parent	Null
Child	1111

GroupID	1100
Parent	1111
Child	0100
Child	0111

Next week

Building scalable services

- CDNs, BitTorrent, caching, replication, load balancing, prefetching, …