P2P Applications

Niels Olof Bouvin

Purpose

- Demonstrate the use of P2P techniques in ‘real’ applications
- Show how a well designed P2P framework can be extended to support wildly different uses

Overview

- YouServ
- YouSearch
- PAST
- SCRIBE

YouServ

- What is YouServ and how does it differ form other personal web servers?
- Central concepts within YouServ
- Making content available
- Accessing content
- Replication
- Firewall traversal
- Summary

What is YouServ?

- A (rather elegant) distributed approach to handling multitudes of small Web servers within an organisation
- Rather than distributing files in email, data is kept close to their originator
- Transience of user machines is handled through transparent replication and firewall tunnelling
- Very modest hardware requirements (2900 users with a couple of (very) standard PCs as central co-ordinators)

Use-Case:
File sharing in companies

- **Email distribution**
  - inefficient (how many identical copies? how many different versions?)
  - no control (recipients can forward email at will)
- **Central Web servers**
  - cumbersome publishing
  - single point of failure
  - access controls?
- **P2P file sharing**
  - requires special software
  - bad image
  - access control?
- **Shared file systems**
  - hard to maintain structure
  - bogged down by rules and regulations :-)
- **Cloud services**
  - control? NSA? Industrial espionage?

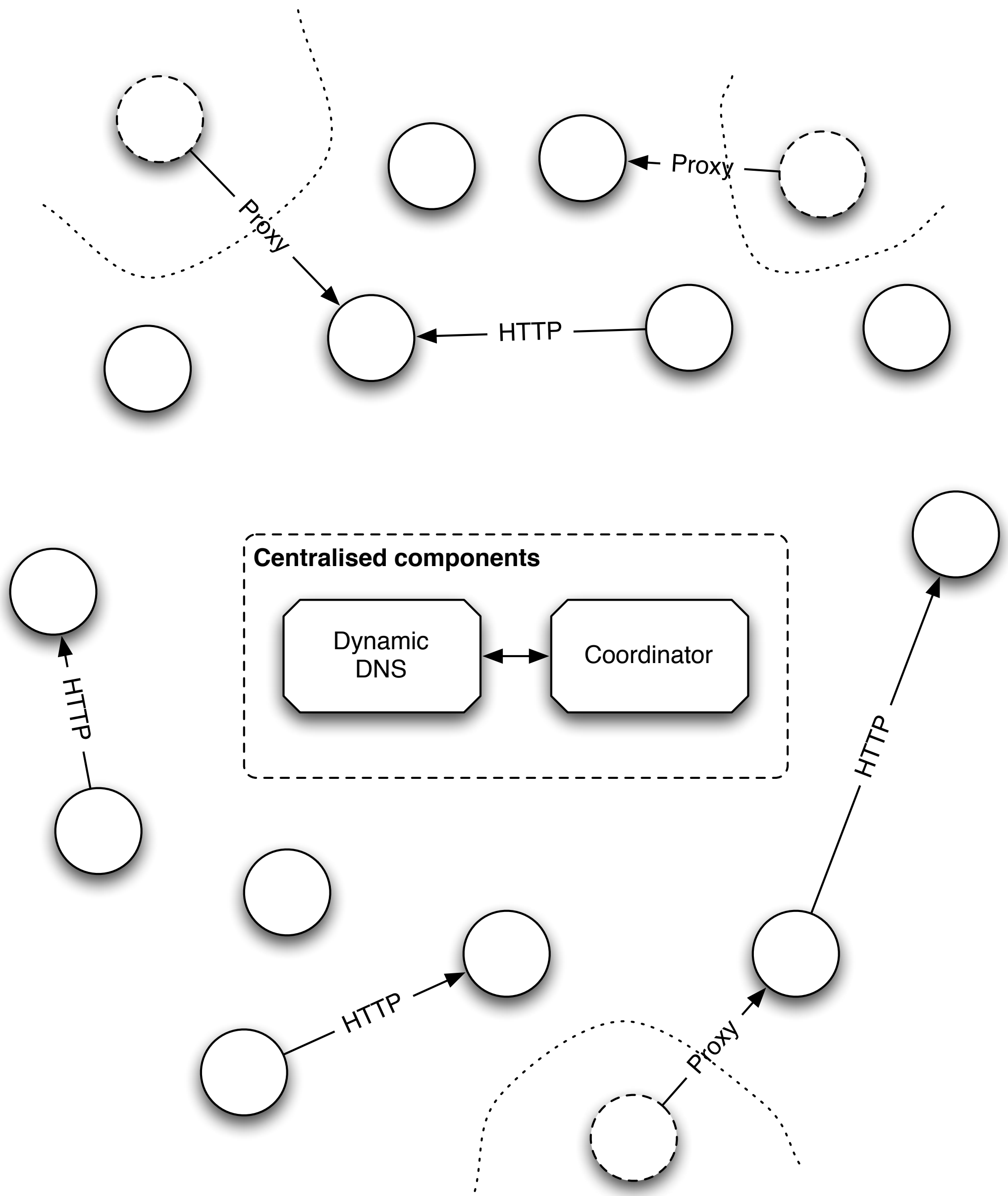
Locating the Right Web Server

- **Data stored on Web servers is easily browsable with ordinary Web browsers**
  - But how is a particular Web server located, if there are thousands in the organisation?
  - Who can remember IP numbers, and what if the machine is upgraded?
- **Solution: personal uServ Web servers are named after the user hosting them, i.e., bayardo.userv.ibm.com and this information is stored in a dynamic DNS server**

Central concepts within YouServ

- **YouServ peer nodes**
  - client running on user’s machine
  - Web server + special YouServ protocols
  - unique name based on user's company identity
- **Dynamic DNS server**
  - rapidly updated DNS entries directing users to current address of the YouServ peer
- **YouServ coordinator**
  - user authentication
  - registers proxying and replication
  - checks availability
  - but no heavy lifting!

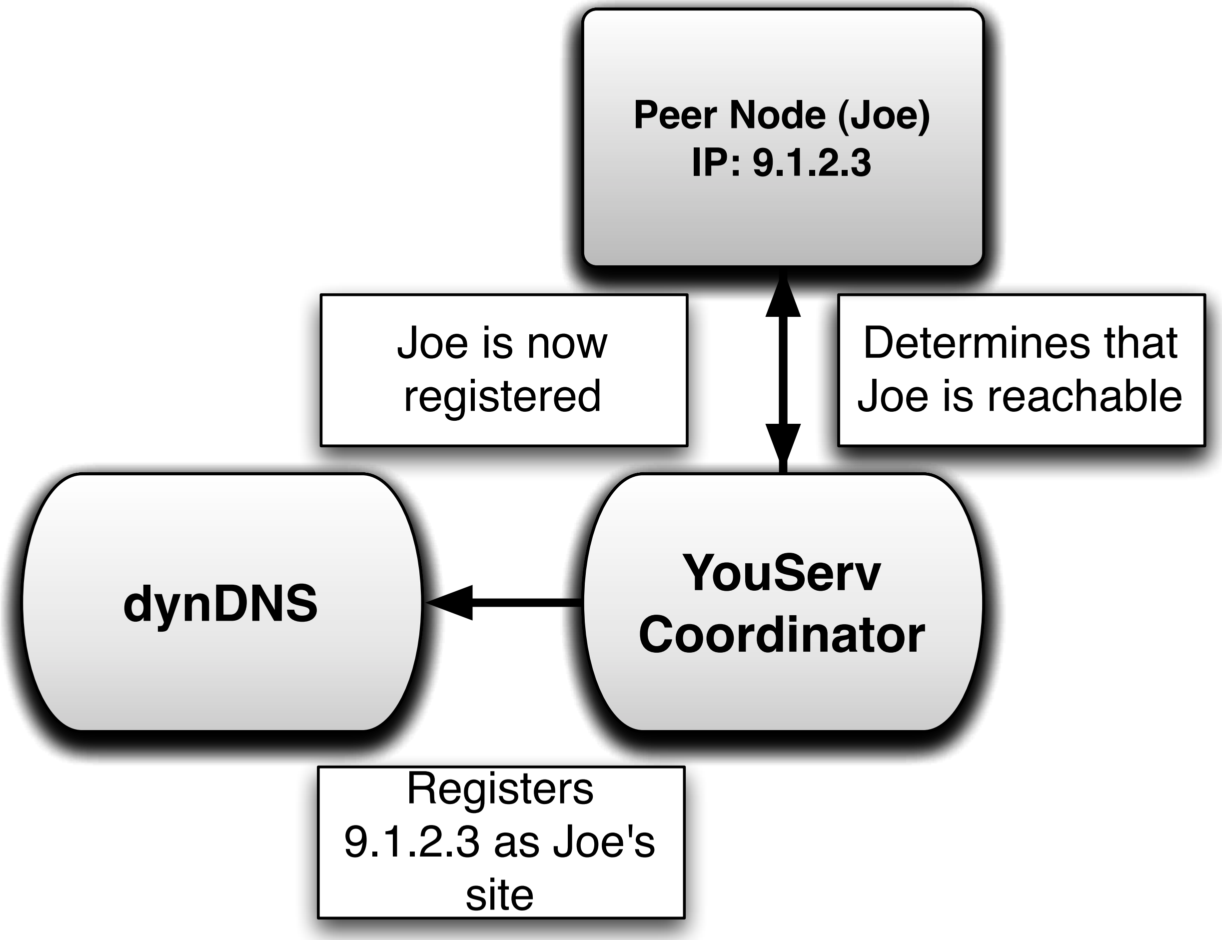
Network overview



Screen shot

Bookmarks
Location: http://bayardo.userv.ibm.com/stuff/demo/
w3
bayardo@us.ibm.com's files
IBM.
Welcome! Please sign in
w3 Home
My Home
uServ Home
uServ Users Online
uServ FAQ
home
stuff
demo
download all (as zip)
my_photos
a_presentation.prz 155KB
photo.jpg 74KB
test.java 2KB
uServ.jar 327KB
unzip.exe 84KB
download all (as zip)
Document: Done

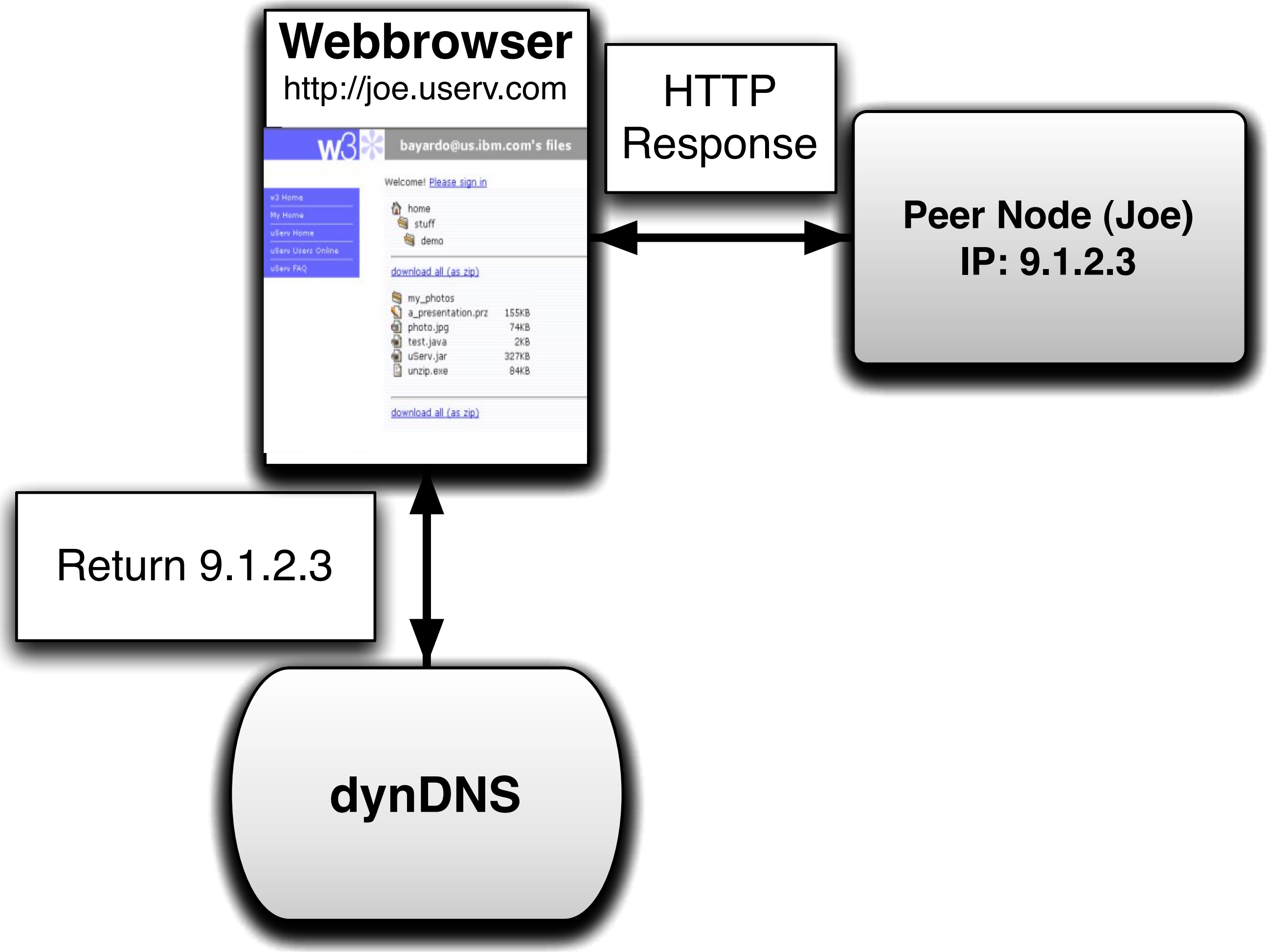
Making content available



Accessing content

- **Data is kept at users' machines running small Web servers**
- **Data is accessed by other users (who need not run any special software) with ordinary Web browsers**
  - they only need to know the name of the user

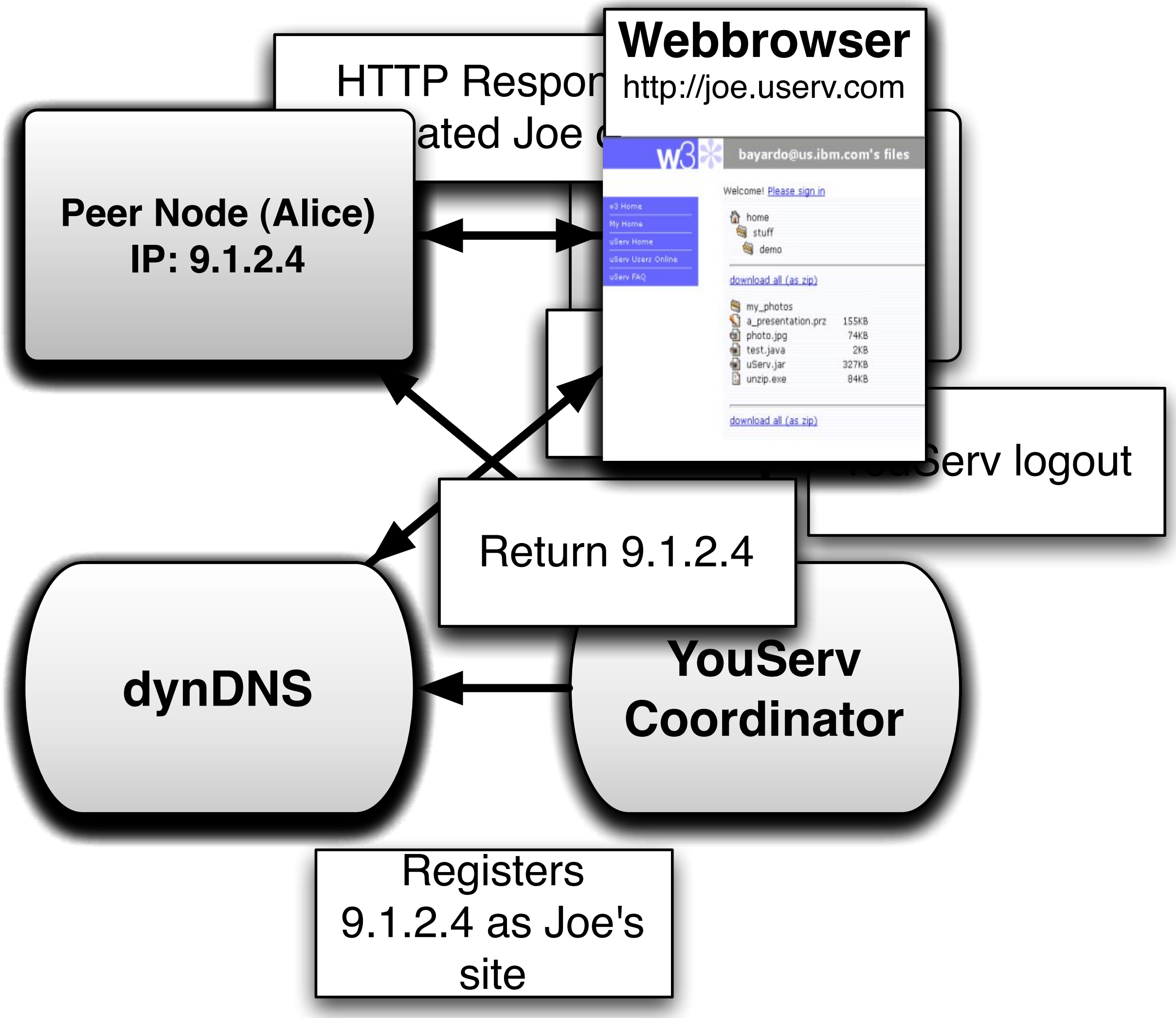
Accessing content



Replication

- **Problem: User machines are transient**
  - on and off
  - laptop computers
- **Solution: Replication**
  - peers (designated manually) replicate data
  - peers maintain summaries of files and synchronise every 3 minutes
    - this distributes availability checking to the peers
- **Replication registered with YouServ Coordinator**
  - upon unavailability, DNS is updated to point at most current replica
  - original target still in HTTP HOST header as with virtual hosting

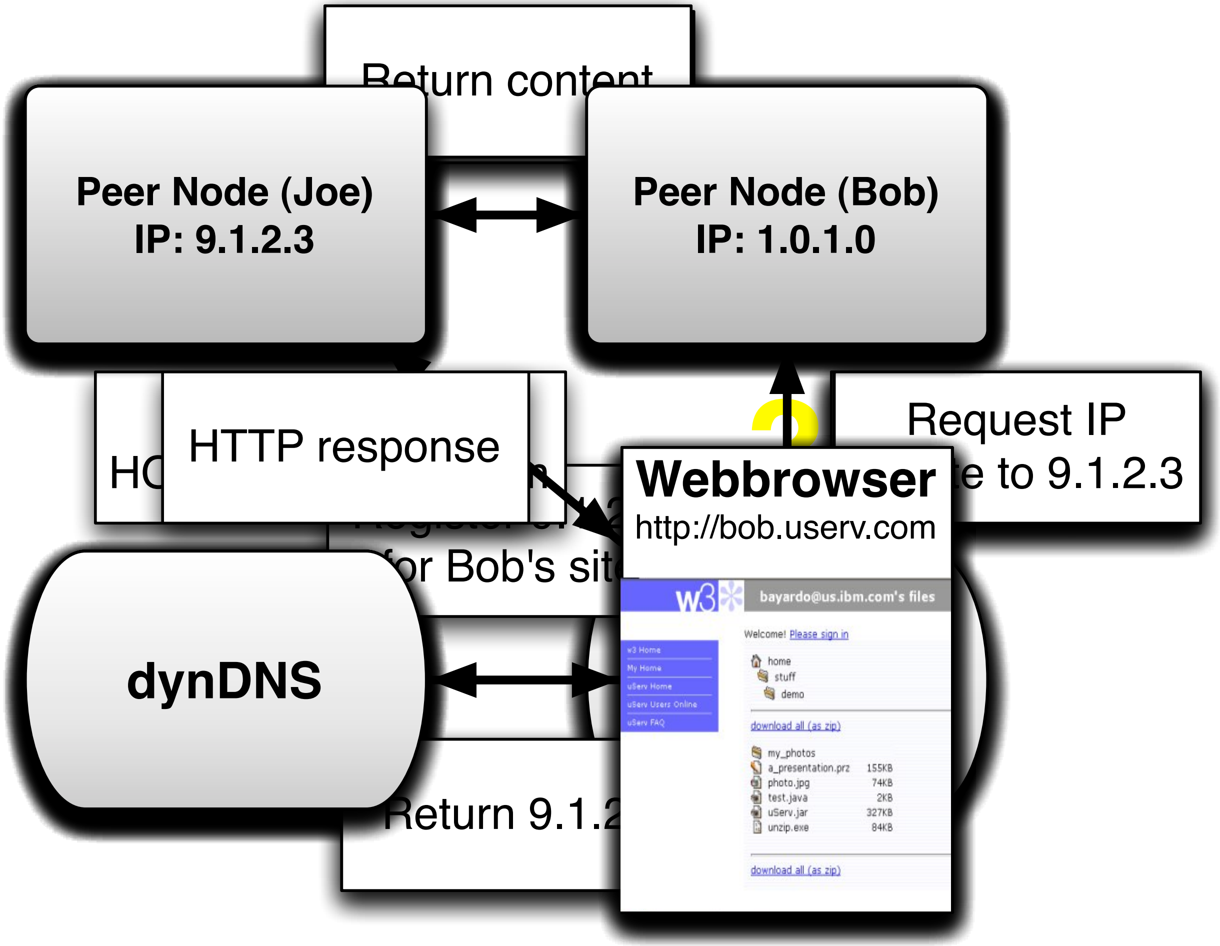
Replication



Proxying – traversing firewalls

- Problem: User machines may have port 80 (in-going) blocked
- Solution: Maintain socket connection from firewalled peer to proxying peer
- All traffic routed through proxy and over permanent socket

Proxying



Return content
Peer Node (Joe)
IP: 9.1.2.3
Peer Node (Bob)
IP: 1.0.1.0
HTTP response
Request IP
dynDNS
Webbrowser
http://bob.userv.com
bayardo@us.ibm.com's files
Welcome! Please sign in
home
stuff
demo
download all (as zip)
my_photos
a_presentation.prz 155KB
photo.jpg 74KB
test.java 2KB
uServ.jar 327KB
unzip.exe 84KB
download all (as zip)

Summary

- Relegates all heavy lifting to the peers
  - content never reaches the server
- Servers handle DNS and light coordination tasks
- Elegant design that seamlessly integrates P2P networking with existing technologies
- Most efficient in organisations such as IBM with centralised authentication

Summary

- **Scalability**
  - Central components are not heavily utilised, but the coordinator will still be a bottleneck if the network becomes sufficiently large
  - All major traffic handled by the peers
  - Replication and proxying ensures high availability
- **Fairness**
  - You share your own stuff
  - And stuff you agree to replicate or proxy

Summary

- **Integrity and security**
  - Safer than email distribution
  - Security tied to authentication scheme used at organisation
- **Anonymity, deniability, censorship resistance**
  - Not even a little – not the purpose of this system

Overview

- YouServ
- **YouSearch**
- **PAST**
- **SCRIBE**

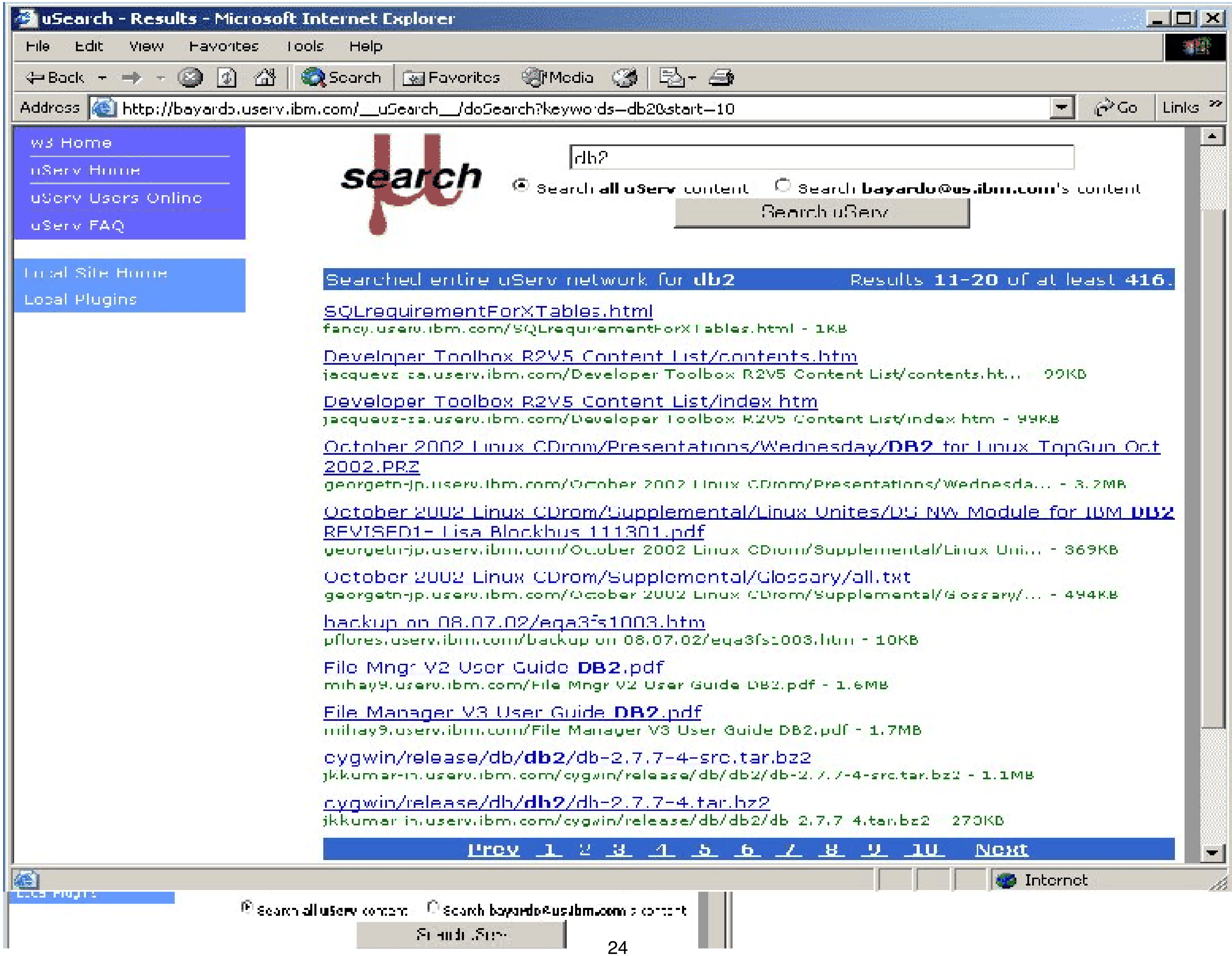
YouSearch

- **Problems with search on personal Web servers**
- **YouSearch**
  - Distributed indexing
  - Scalability
  - Caching
- **Summary**

Searching personal web servers

- **While data is kept on Web server, standard Web techniques are not applicable**
- **Crawling**
  - slow
  - never up to date (and never complete)
  - can return dead links
  - requires some big central server

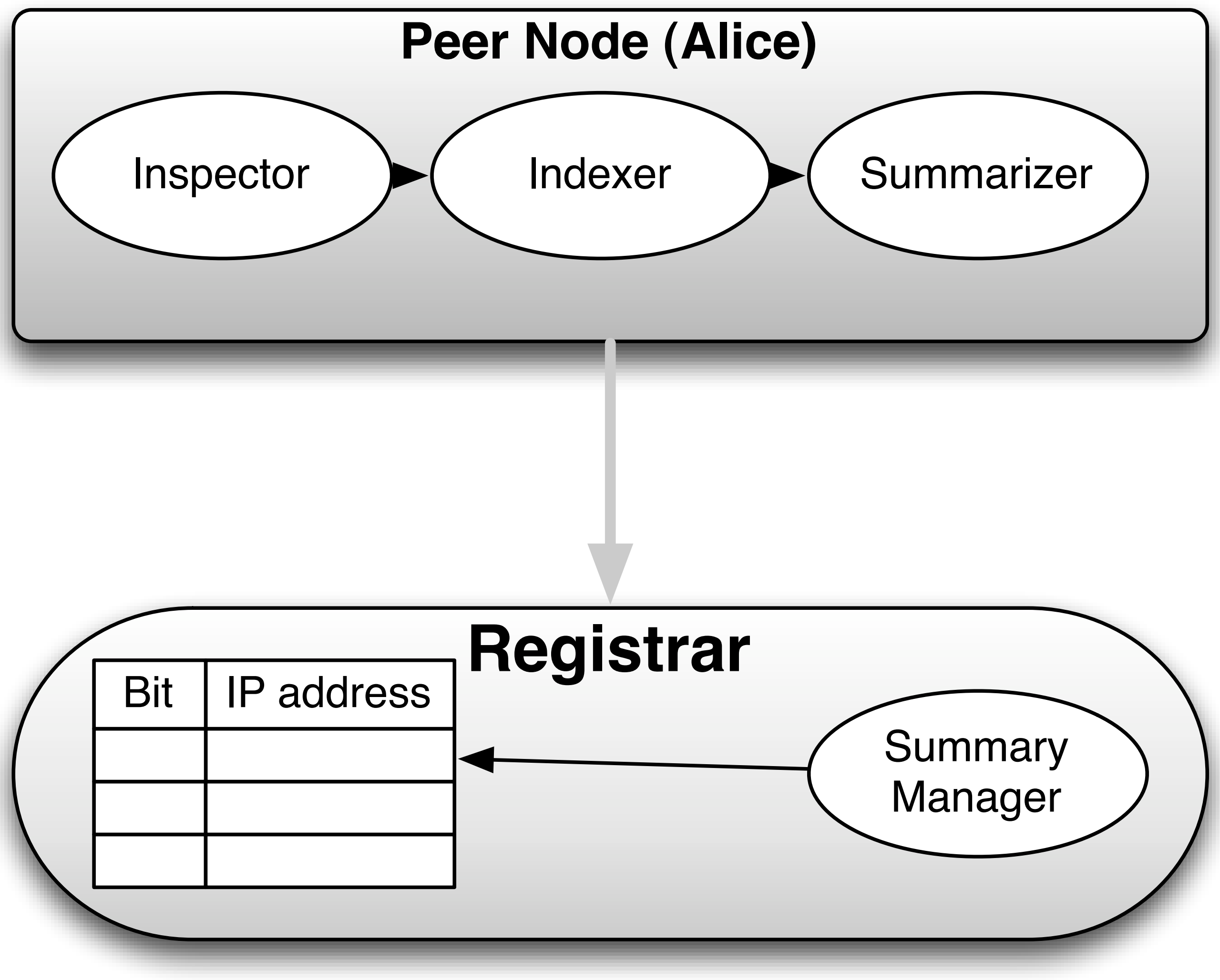
YouSearch interface



Distributed indexing

- **Nodes index their own documents**
- **A "summarizer" computes a bloom filter over the index and sends it to the central registrar**
  - bloom filters: bit vectors created with hashes over terms
  - if H('term') yields *k*, the *k*th bit is set in the bit vector
  - if the *k*th bit is not set, then 'term' is not present
  - (if you are interested, there is an *excellent* description of Bloom filters at Wikipedia http://en.wikipedia.org/wiki/Bloom_filter)
- **Central registrar combines bloom filters from clients**

Architecture



Bloom filters

Insert "foo"

Lookup "baz"

Hash1("baz") = 0

Hash2("baz") = 8

Hash3("baz") = 17

Answer: "baz" doesn't exist

Answer: "baz" <u>may</u> exist

1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0

m = 24

k = 3

Scalability

- *K* hash functions may be used to make a bloom filter more precise; all of the resulting bits (for each hash function) are thus set in the bit vector.
- This technique is used in YouServ but…
  - Instead of setting *k* bits per term in a single bit vector *k* bit vectors are maintained, one for each hash function.
  - This makes for future scalability of the registrar's part of the searching, i.e., *k* registrars may be used in parallel.

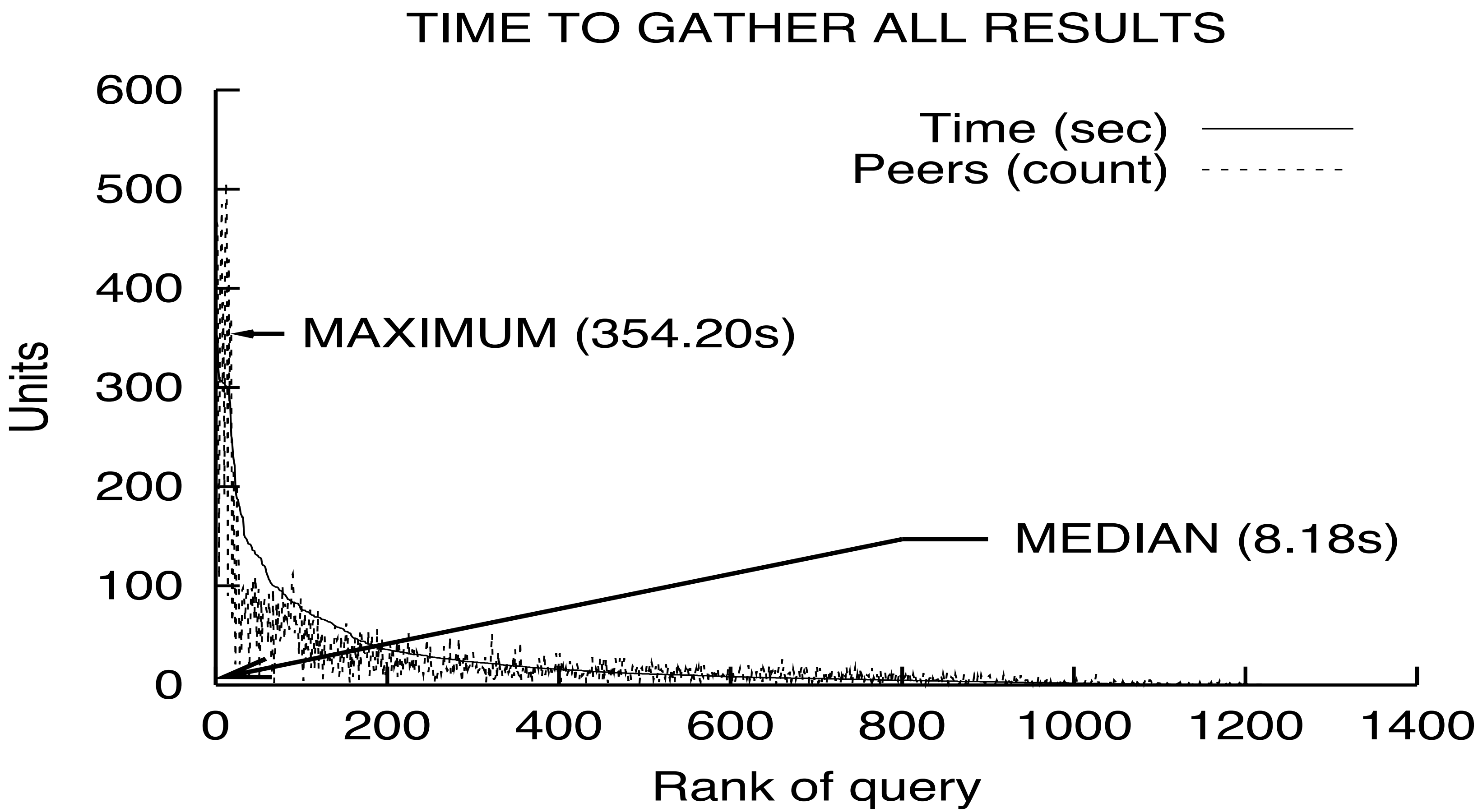
Searching in YouSearch

- Searching can originate at any peer running YouServ
- The registrar can quickly decide which peers might have documents matching the query using the Bloom filters
- The set of matching peers is returned to the originator, who then queries the peers in turn
- “Dead peers tell no tales”
  - but are discovered by the other peers and reported to the central registrar
- Results are presented to the user as they come in

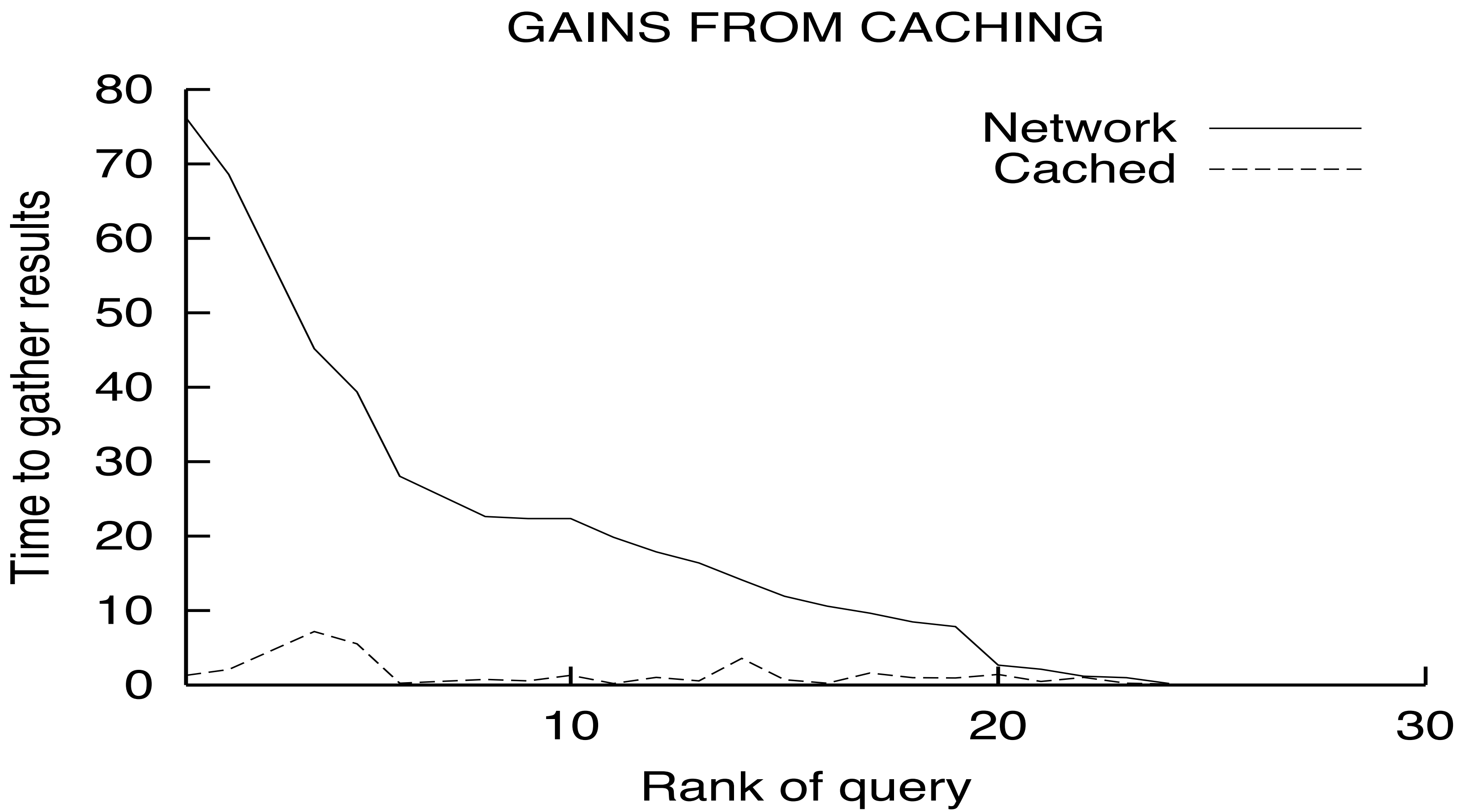
Caching / Context

- **Queries are cached by querying peer who informs the registrar**
  - registrar stores only query and IP address
  - cached queries have a TTL
- **Work groups as context**
  - search context (only stuff within this work group)
  - recommendation of search results to other work group members
  - search results can be made persistent by users

Performance of search



Caching matters



Summary

- **Relegates all heavy lifting to the peers**
  - indexing, summarizing, and performing actual search
- **Servers handle light coordination of search tasks**
  - look up in Bloom filters, coordinate caching

Summary

- **Scalability**
  - Central components are not heavily utilised, can be clustered if need be
  - All major traffic handled by the peers
  - Caching removes a lot of load from both the coordinator and the individual peers
- **Fairness**
  - You host the search engine capable of searching through your own stuff
- **Integrity and security**
  - only public stuff is searchable so no security measures have been taken
- **Anonymity, deniability, censorship resistance**
  - Not even a little – not the purpose of this system

Overview

- YouServ
- YouSearch
- **PAST**
- **SCRIBE**

PAST

- **What is the purpose of PAST?**
- **Key characteristics of PAST**
- **PAST operations**
  - insert
  - lookup
  - reclaim
- **Storage management**
  - replication and diversion
  - caching
- **PAST evaluation**

Purpose of PAST

- Exploit multitude and diversity of Internet nodes to achieve strong persistence and high availability
- Create global storage utility for backup, mirroring, ...
- Share storage and bandwidth of a group of nodes – larger than capacity of any individual node

Key characteristics of PAST

- **Large-scale P2P persistent storage utility**
  - strong persistence (resilient to failure)
  - high availability
  - scalability
  - security
- **Self-organizing, Internet-based structured overlay of nodes cooperate**
  - route file queries
  - store replicas of file
  - cache popular files
- **Based on Pastry**

PAST design

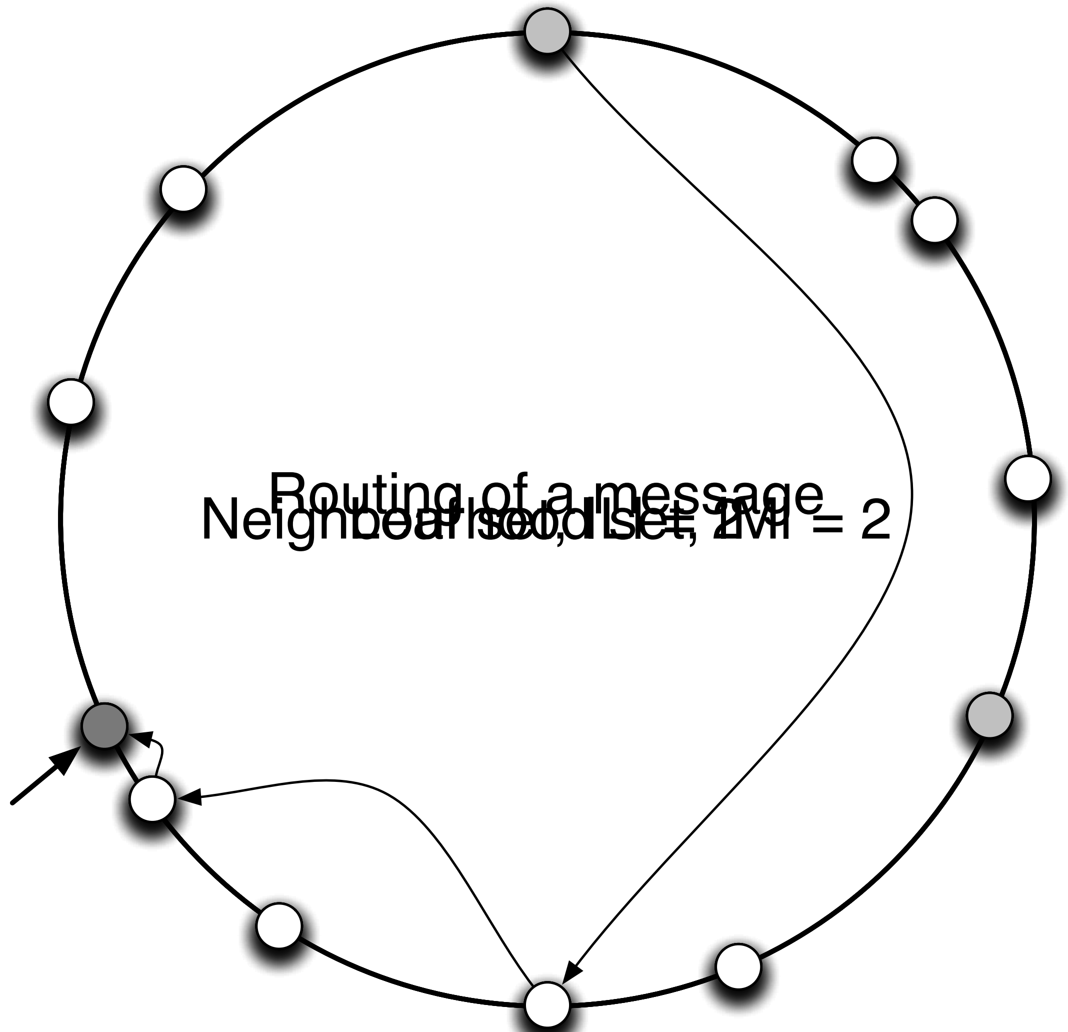
- **Any node running the PAST system may participate in the PAST network**
  - nodes minimally act as access points for users, but may also contribute storage and routing capabilities to the network
  - nodes have 128 bit quasi-random IDs (lower 128 bit of SHA-1 on the node's public key)

    → nodes with adjacent IDs diverse
  - file publishers have public/private cryptographic keys

Quick Pastry recap

- **Pastry is a structured P2P network**
  - supports effective, distributed object location and routing
    - O(log N) routing
    - Routing tables of size O(log N)
- **A “routing ring”**
  - nodes are given unique, quasi-random IDs and are placed on the ring accordingly
  - during placement a routing table is built
- **Locality awareness**
  - Pastry maintains a “neighbourhood set” of the |M| nodes that are closest by some proximity measure (e.g., routing hops)

The Pastry routing ring



PAST operations

- `fileId = Insert(name, owner-credentials, k, file)`
  - inserts replicas of file on the k nodes whose IDs are numerically closest to fileId ($k \leq |L|$)
  - the system must maintain k copies of the file
- `file = Lookup(fileId)`
  - retrieve file designated by fileId if it exists and one of the k replica hosts are reachable
  - the file is usually retrieved from the "closest" (in terms of proximity) of the k nodes
- `Reclaim(fileId, owner-credentials)`
  - weak delete: lookup of fileId is no longer guaranteed to return a result

Insert

- `fileId = Insert(name, owner-credentials, k, file)`
  - fileId is calculated (SHA-1 of file name + public key + random number ("salt"))
  - Storage required deducted against a client quota
  - File certificate created and signed with private key
    - Contains fileId, SHA-1 of file content, replication factor k, the random salt, various meta data
  - File certificate + file is then routed to the fileId destination
  - Destination verifies certificate, forwards to k-1 closest nodes (i.e. to k-1 nodes in its Pastry leaf set)
  - Destination returns store receipt if all accepts

So where does the file go?

NodeId 10233102

Leaf set

	Smaller	Larger	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	

Neighbourhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Lookup

- `file = Lookup(fileId)`
  - Given a requested fileId, a lookup request is routed towards a node with ID closest to fileId
  - Any node storing a replica may respond with file and file certificate (and won't forward the query)
  - Since k numerically adjacent nodes store replicas and Pastry routes towards local nodes, a node close in the proximity metric is likely to reply

Reclaim (weak delete)

- `Reclaim(fileId, owner-credentials)`
  - Analogous to insert, but with a “reclaim certificate” verifying that the original publisher reclaims the file
  - A reclaim receipt is received, used to reclaim storage quota
  - Reclaim is not the same as delete – copies may still be out there, but there are no longer any guarantees

Storage management

- **We want the aggregated size of stored files to be close to the aggregated capacity in a PAST network, before insert requests are rejected**
  - unused disk space is wasted disk space
  - should be done in a decentralized way...
- **Two ways of ensuring this**
  - replica diversion
  - file diversion

Replica diversion

- Balances free storage space among nodes in a leaf set
- If a node cannot store a replica locally, it asks a node in its leaf set (but outside of k) if it can, and stores a pointer to the file
  - protocol must handle failure of leaf nodes then
- In case of failure
  - storage node fails → find a new node
  - original node fails → k+1 leaf becomes member of k
  - k+1 leaf fails → take the next one

When to accept a replica

- **Acceptance of a replica at a node for storage is subject to policies**
  - file size divided by available size should be lower than a certain threshold (leave room for small files)
  - threshold lower for nodes containing diverted replicas (leave most space for primary replicas)

File diversion

- **If there is no room for the file in the given ID space, file diversion can be employed**
  - this is done simply by calculating a new fileId by choosing a new random salt
  - only used as a last resort, when replica diversion can not be used
- **Once a new fileId is obtained the file can try to insert it once more**
  - and if that also fails file diversion can be used once more...
    - but you have to stop at some point and realise that there is no more room in the network

Caching

- **Goals of cache management**
  - minimize access latency (here routing distance)
  - maximize throughput
  - balance query load in system
- **The *k* replicas ensures availability, but also gives some load balancing and latency reduction because of locality properties of Pastry**
- **A file is cached in PAST at a node traversed in lookup or insert operations, if the file size is less than some fraction of the node's remaining cache size**
- **Caching files are evicted as needed**

PAST evaluation

- **Experimental setup**
  - prototype implementation of PAST in Java
  - network emulation environment
  - all nodes run in same Java VM
- **Different normal distributions of storage capacity of nodes used**
- **Workload data from traces of file usage**
  - eight Web proxy logs (1,863,055 entries, 18.7 GB)
  - workstation file system (2,027,908 files, 166.6 GB)
  - "problematic to get data of real P2P usage"
- **2250 PAST nodes, k=5, b=4**

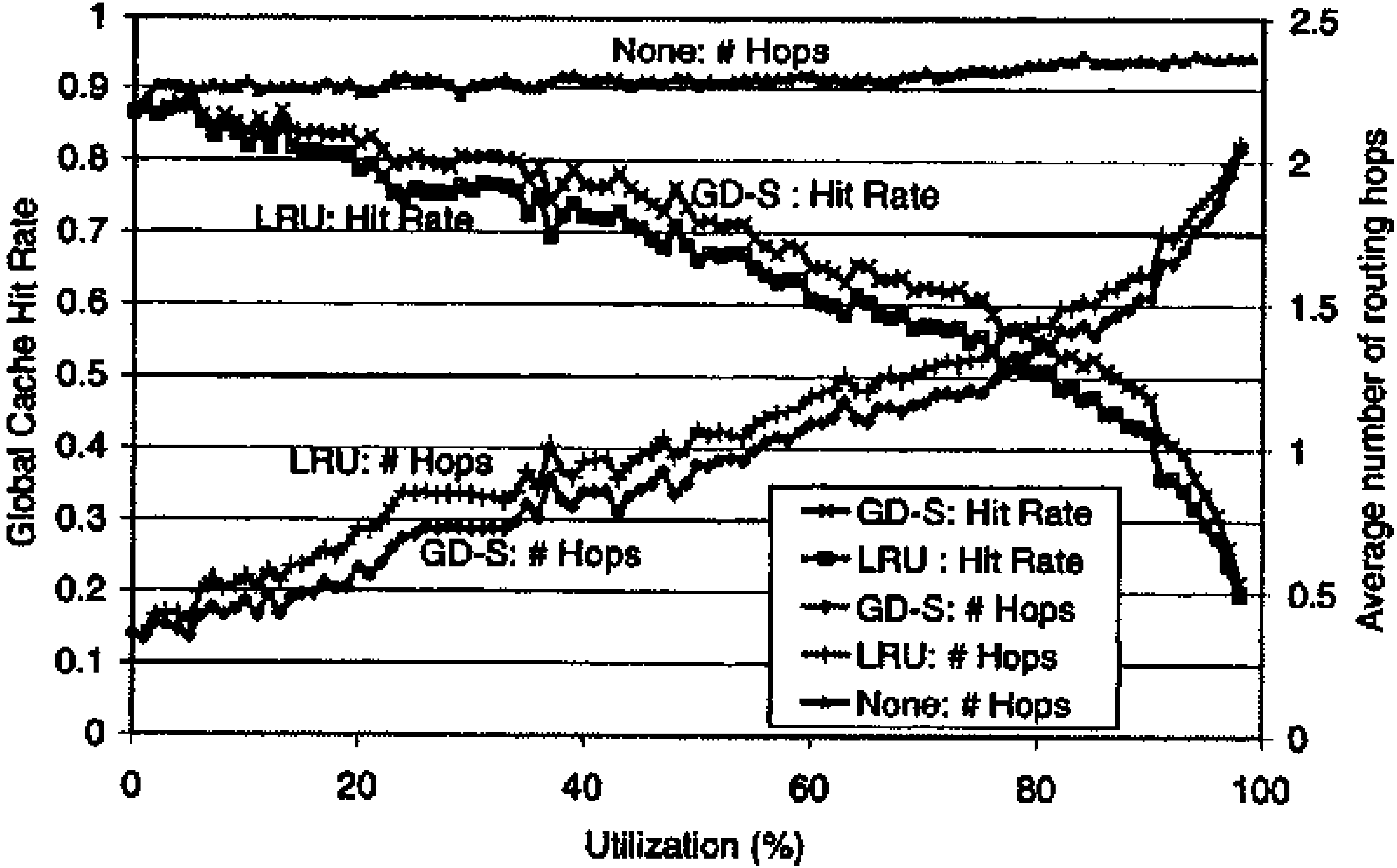
Storage management is needed

- **Experiment without replica diversion and file diversion**
  - primary replica threshold = 1, diversion replica threshold = 0
  - insertion rejection on first file insertion failure
- **51.1% insertion rejection...**
- **60.8% ultimate storage utilization...**

Storage management is effective

- **Adding file and replica diversion changes the picture completely**
  - primary replica threshold = 0.1, diversion replica threshold = 0.05
- **Insertion rejection is now down to 0.6% – 5.5%**
- **Storage utilisation is up to 94.0% – 99.3%**

Caching is *good*



Summary

- **Scalability**
  - tied to the scalability of Pastry
- **Fairness**
  - There is a quota system, so in order to use storage space you must also provide some
- **Integrity and security**
  - file integrity is ensured using hashes
  - public/private key system ensures (if properly used) ownership and privacy
  - k copies makes data loss unlikely
- **Anonymity, deniability, censorship resistance**
  - not really the system for anonymous data storage
  - caching ensures that a file can not be requested out of existence

Overview

- YouServ
- YouSearch
- PAST
- **SCRIBE**

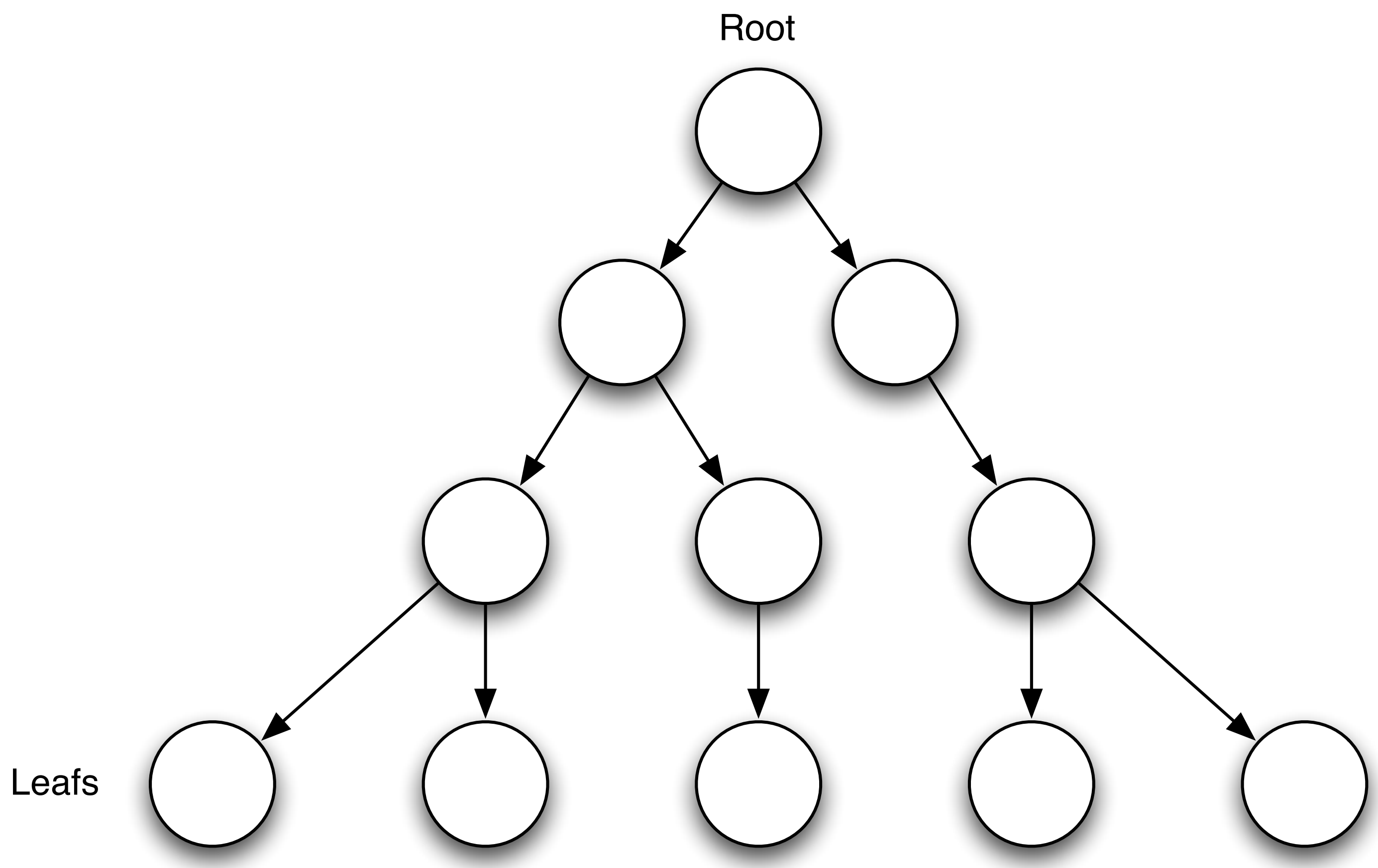
SCRIBE

- **What is the purpose of SCRIBE?**
- **General observations about P2P multicast trees**
- **SCRIBE**
  - Group management
  - Message dissemination
  - Tree maintenance
- **SCRIBE evaluation**

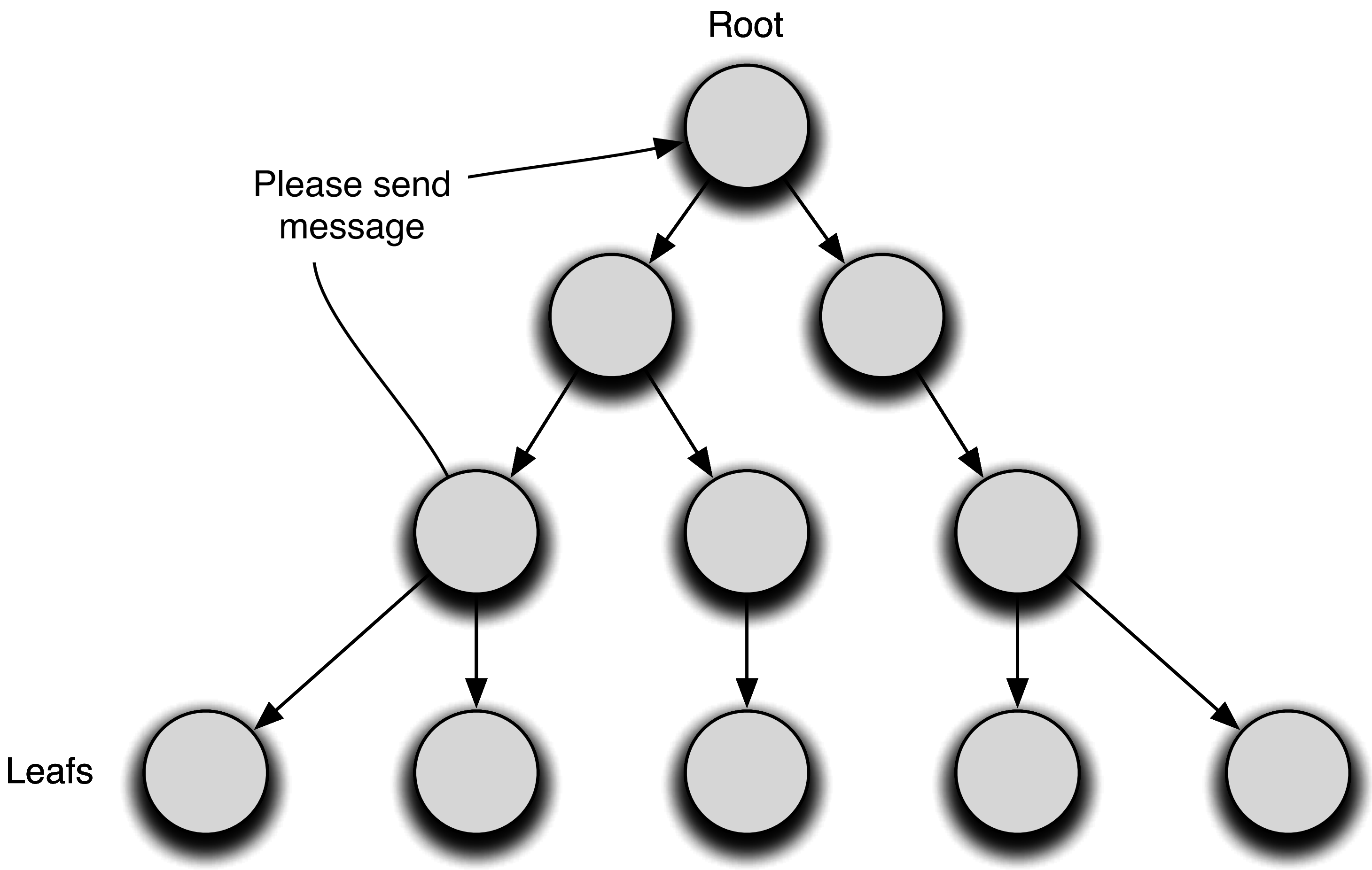
The purpose of SCRIBE

- **SCRIBE is an example of a P2P based multicast system.**
- **Multicast can be used for a number of things**
  - group chat (text, sound, or video)
  - live media streaming (sound or video)
  - multiplayer games
  - . . . and anything else you can think of where a group of peers need to communicate one-to-many or many-to-many in an efficient manner

An example multicast tree

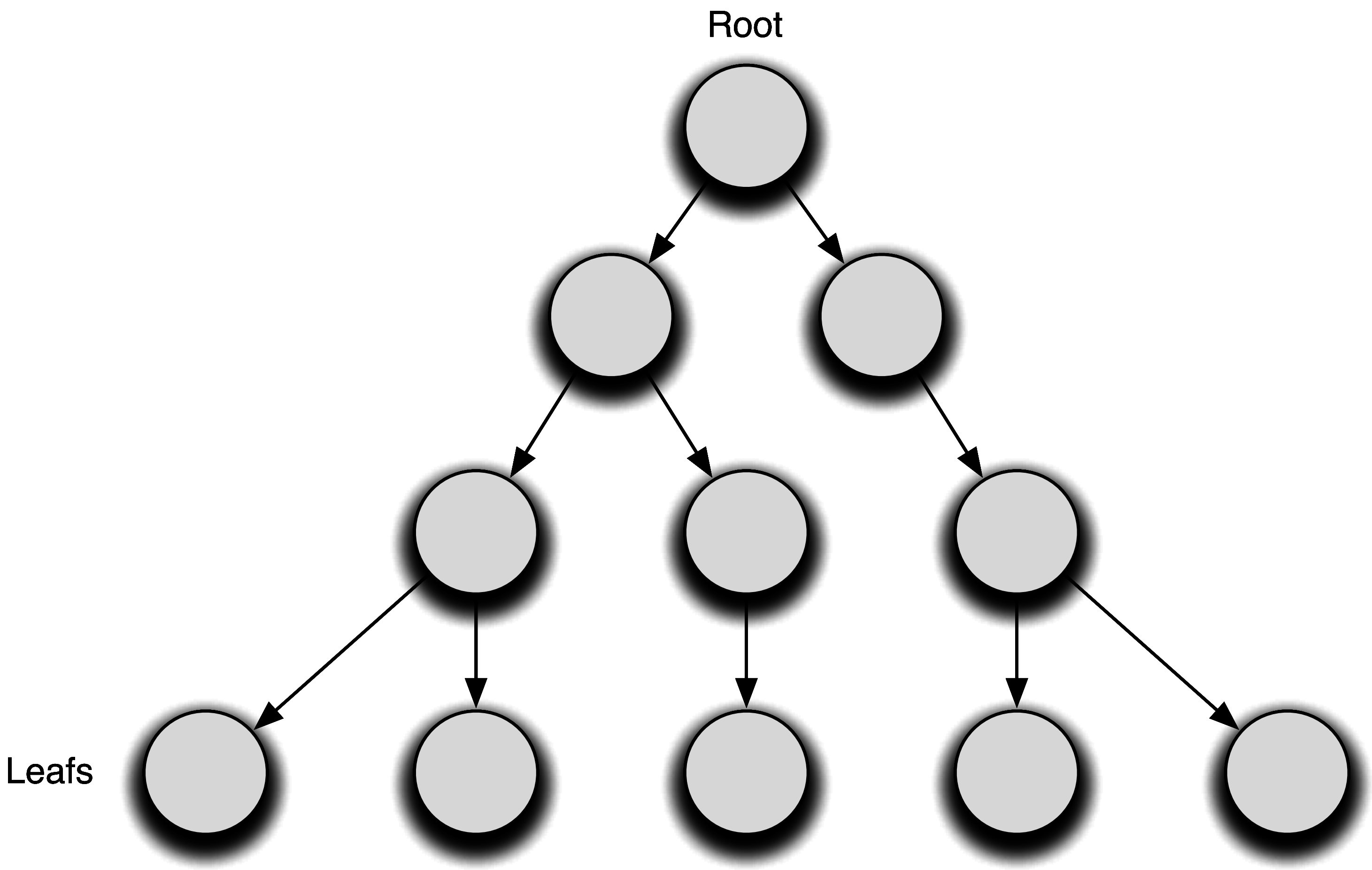


Message passing in multicast trees
— top-down —



Message passing in multicast trees

— crawl —



Message dissemination, pros and cons

- **Letting the root control message flow means that *sequence* becomes easy to handle**
  - … but the “connection” load on the root can be huge
- **In the crawling dissemination all nodes are equal**
  - … but maintaining a sequence is hard
- **Which one to use depends heavily on use-case**
  - e.g., in live media streaming (one-to-many) a top-down approach would be fine
  - in a game, where sequence is of importance, top-down would probably also be preferable
  - in systems that must scale to thousands of users, and where sequence is of less importance, a crawl is best suited

Security

- **There are many ways to be malicious in a multicast system**
  - by flooding the network – messages are replicated throughout the network making it easy to create a tsunami of data
  - by not forwarding messages
  - . . . and much more
- **There are lots of systems that try to circumvent these attacks**
  - e.g., by using multiple trees, cryptographic measures etc.

SCRIBE: Creating a group

- **A groupID is generated**
  - … it's the SHA-1 hash of the textual name + creator name
  - which gives a uniform distribution of rendezvous nodes = balances the load
- **A "create" message is routed towards the node in the Pastry network whose ID is closest to the groupID**
- **The receiving node is now the rendezvous point for the group (the root of the tree).**

SCRIBE: Joining the group

- **The peer sends a "join" message towards the groupID**
- **An intermediate node forwarding this message will:**
  - If the node is currently a forwarder for the group it adds the sending peer as a child and we're done.
  - If the node is not a forwarder, it adds the sender as a child and then *sends its own "join" message* towards the groupID
    - thus becoming a forwarder for the group
- **Every node in a group is a *forwarder* – but this does not mean that it is also a *member***

SCRIBE: Leaving the group

- **The peer locally marks that it is no longer a member but merely a forwarder**
- **It then proceeds to check whether it has any children in the group**
  - and if it hasn't it sends a "leave" message to its parent (which continues recursively up the tree if necessary)
  - otherwise it stays on as a forwarder

SCRIBE: Sending messages

- **Messages are sent directly (outside of Pastry) to the rendezvous (root) node**
  - it is thus a **top-down** approach

SCRIBE: Repairing the tree

- **Periodically, each non-leaf node sends a heartbeat message to its children**
  - multicast messages are used as an implicit heartbeat
- **If children have not received a heartbeat for a set amount of time it simply rejoins the group by sending a “join” message towards the rendezvous node.**

SCRIBE: Recovering from a lost root

- **The state kept in the root node is replicated in the Pastry leaf set of the root**
- **These nodes are numerically close to the root = the new root is therefore amongst these nodes**
- **When the root disappears the children will rejoin the group, and the new root will receive these join requests**
  - upon which it notices that it is now the root and starts acting accordingly

Summary

- **Scalability**
  - tied to the scalability of Pastry
    - which is good by the way
- **Fairness**
  - All nodes are responsible for forwarding messages in the tree (except for leaf nodes)
  - Some non-member nodes must also forward messages
  - The role of rendezvous (root) node is uniformly distributed

Summary

- **Integrity and security**
  - Being a top-down approach some authentication scheme can be employed at the root node
  - . . . but apart from that security is not really discussed in the paper
- **Anonymity, deniability, censorship resistance**
  - is not considered at all