Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems

Karl Aberer, Manfred Hauswirth

EPFL - IIF
Distributed Information Systems Laboratory (LSIR)
http://lsirwww.epfl.ch/

Goals of the Tutorial

1. **Position the P2P paradigm in the design space of distributed systems**
2. **Get an overview of state-of-the-art P2P systems**
3. **Understand the problem of decentralized data management in P2P systems**
4. **Understand research issues for future systems**

Outline of the Tutorial

- What is P2P?
  - P2P properties
  - Types of P2P Systems
  - A historical view
  - Related approaches
- State-of-the-art systems
  - Napster, Gnutella, Freenet, OceanStore, Farsite, FastTrack
- P2P Data Management
  - Gnutella, Freenet, Chord, CAN, Tapestry, P-Grid
  - Applications of P-Grid
    - Gridella
    - Managing trust
- Conclusions and References
- Appendix: Protocols (Napster, Gnutella, Freenet)

Listen — P2P is around

- P2P systems receives currently a lot of attention
  - File-sharing systems
    - Napster, Gnutella, Freenet, etc.
  - Conferences
    - O'Reilly P2P conference 2001 (conferences.oreilly.com/p2p/)
    - 2001 International Conference on Peer-to-Peer Computing (P2P2001) (www.ida.liu.se/conferences/p2p/p2p2001/)
    - Peer-to-Peer Computing Workshop at Networking 2002 conference (http://www.cnuce.pi.cnr.it/Networking2002/)
    - etc.
- P2P is nothing new – see Arpanet

1. What is P2P? … and what isn't?

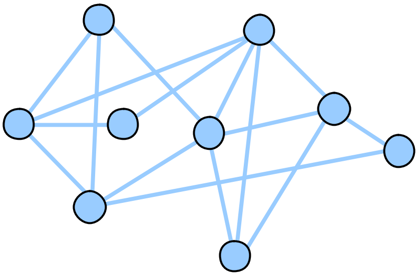
- Clay Shirkey (The Accelerator Group):
  - "Peer-to-peer is a class of applications that take advantage of resources—storage, cycles, content, human presence—available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers."
  - P2P "litmus test:"
    - Does it allow for variable connectivity and temporary network addresses?
    - Does it give the nodes at the edges of the network significant autonomy?
- P2P ~ an application-level Internet on top of the Internet

So what's new? P2P in a historical Context

- The original Internet was designed as a P2P system
  - any 2 computers could send packets to each other
    - no firewalls / no network address translation
    - no asymmetric connections (V.90, ADSL, cable, etc.)
  - the back-then "killer apps" FTP and telnet are C/S but anyone could telnet/FTP anyone else
  - servers acted as clients and vice versa
  - cooperation was a central goal and "value": no spam or exhaustive bandwidth consumption
- Typical examples of "old-fashioned P2P":
  - Usenet News
  - DNS
- The emergence of P2P can be seen as a renaissance of the original Internet model

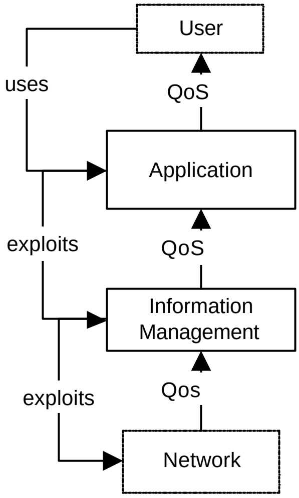
What is P2P?

- Every participating node acts as both a client and a server (“servent”)
- Every node “pays” its participation by providing access to (some of) its resources
- Properties:
  - no central coordination
  - no central database
  - no peer has a global view of the system
  - global behavior emerges from local interactions
  - all existing data and services are accessible from any peer
  - peers are autonomous
  - peers and connections are unreliable



Where is P2P – System layers ?

- Users
  - Commerce and society is P2P
- Application layer
  - E-commerce systems can be P2P or centralized
- Information management
  - Directories and databases can be P2P or centralized
- Networks are P2P
  - Internet



User

uses

QoS

Application

exploits

QoS

Information Management

exploits

Qos

Network

Types of P2P Systems

- E-commerce systems
  - eBay, B2B market places, B2B integration servers, …
- File sharing systems
  - Napster, Gnutella, Freenet, …
- Distributed Databases
  - Mariposa [Stonebraker96], …
- Networks
  - Arpanet
  - Mobile ad-hoc networks

How much P2P is involved?

	P2P user interaction	P2P application	P2P information management
eBay	yes	no	no
Napster	yes	yes	no
Gnutella, Freenet	yes	yes	yes

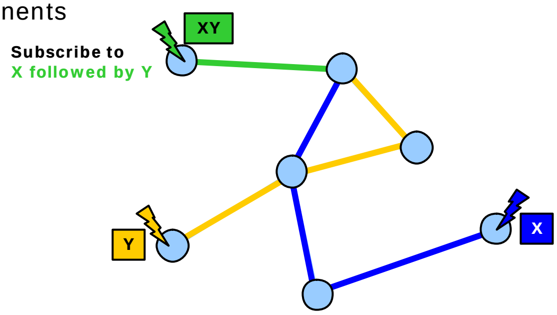
Related Approaches

Related distributed information system approaches:

- Event-based systems
- Push systems
- Mobile agents
- Distributed databases

Event-based (publish/subscribe) Systems

- System model
  - Components (peers) interact by generating and receiving events
  - Components declare interest in receiving specific (patterns of) events and are notified upon their occurrence
  - Supports a highly flexible interaction between loosely-coupled components



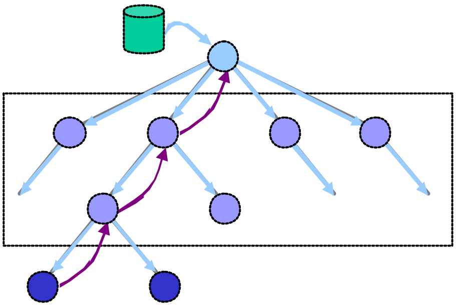
Subscribe to
X followed by Y

Event-based vs. Peer-to-Peer

- Common properties:
  - symmetric communication style
  - dynamic binding between producers and consumers
- Subscription to events ~ “passive” queries
  - EB: notification
  - P2P: active discovery
- Subscription language supports more sophisticated queries and pattern matching (event patterns with time dependencies)
- Event-based systems typically have a specialized event distribution infrastructure
  - EB: 2 node types, P2P: 1 node type
  - EB infrastructure must be deployed

Push Systems

- A set of designated broadcasters offer information that is pre-grouped in channels (weather, news, etc.)
- Receivers subscribe to channels of their interest and receive channel information as it is being “broadcast” (timely distribution)
- Receivers may have to pay prior to receiving the information (pay-per-view, flat fee, etc.)
- Pull → push

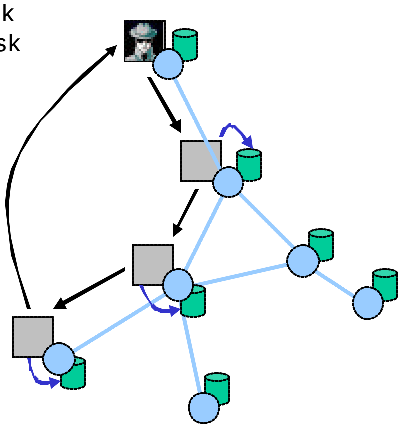


Push Systems vs. Peer-to-Peer

- Asymmetric communication style (P2P: symmetric)
- Focus is on timely data distribution not on discovery
- Filtering may be deployed to reduce data transmission requirements
- Subscription to channels is prerequisite
- Producer/consumer binding is static
- Push systems require a specialized distribution infrastructure
  - Push: 3 node types, P2P: 1 node type
  - Push infrastructure must be deployed

Mobile Agents

- A mobile agent is a computational entity that moves around in a network at its own volition to accomplish a task on behalf of its owner
  - can cooperate with other agents
  - “learns” (“Whom to visit next?”)
- Mobility (heterogeneous network!)
  - Weak: code, data
  - Strong: code, data, execution Stack

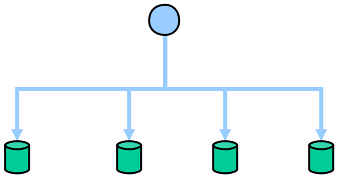


Mobile Agents vs. Peer-to-Peer

- Very similar in terms of search and navigation
  - P2P: the peers propagate requests (search, update)
  - MA: the nodes propagate the agents
  - Mobile agent ~ "active" query
- Mobile agent systems require a considerably more sophisticated environment
  - mobile code support (heavy)
  - security (protect the receiving node from malicious mobile agents and vice versa)
- In many domains P2P systems can take over
  - more apt for distributed data management
  - less requirements (sending code requires much bandwidth, security, etc.)

Distributed Databases

- Fragmenting large databases (e.g., relational) over physically distributed nodes
- Efficient processing of complex queries (e.g., SQL) by decomposing them
- Efficient update strategies (e.g., lazy vs. eager)
- Consistent transactions (e.g., 2 phase commit)
- Normally approaches rely on central coordination



Distributed Databases vs. Peer-to-Peer

- Data distribution is a key issue for P2P systems
- Approaches in distributed DB that address scalability
  - LH* family of scalable hash index structures [Litwin97]
  - Snowball: scalable storage system for workstation clusters [Vingralek98]
  - Fat-Btree: a scalable B-Tree for parallel DB [Yokota 9]
- Approaches in distributed DB that address autonomy (and scalability)
  - Mariposa: distributed relational DBMS based on an underlying economic model [Stonebraker96]
- P2P Data Management has to address both scalability and autonomy

Usage Patterns to Position P2P

Discovering information is the predominant problem

- Occasional discovery: search engines
  - ad hoc requests, irregular
  - E.g., new town — where is the next car rental?

P2P, MA

- Notification: event-based systems
  - notification for (correlated) events (event patterns)
  - E.g., notify me when my stocks drop below a threshold

push

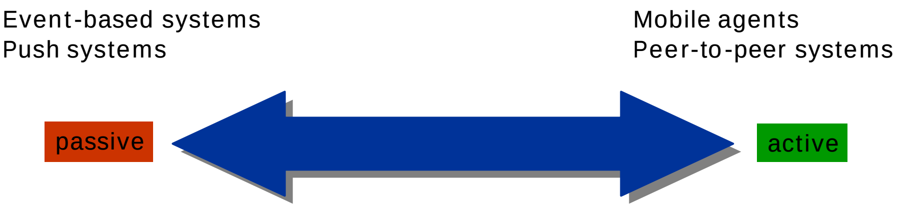
- Systematic discovery: P2P systems
  - find certain type of information on a regular basis
  - E.g., search for MP3 files of Jethro Tull regularly

search engines, MA

- Continuous information feed: push systems
  - subscription to a certain information type
  - E.g., sports channel, updates are sent as soon as available

event-based

The Interaction Spectrum



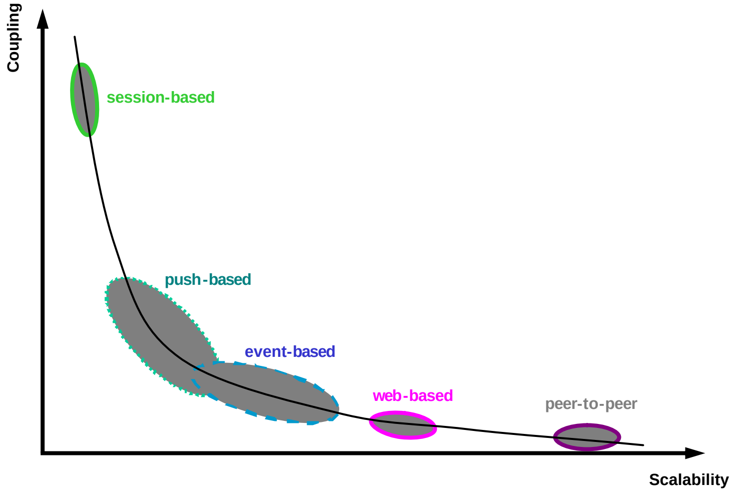
Event-based systems
Push systems

Mobile agents
Peer-to-peer systems

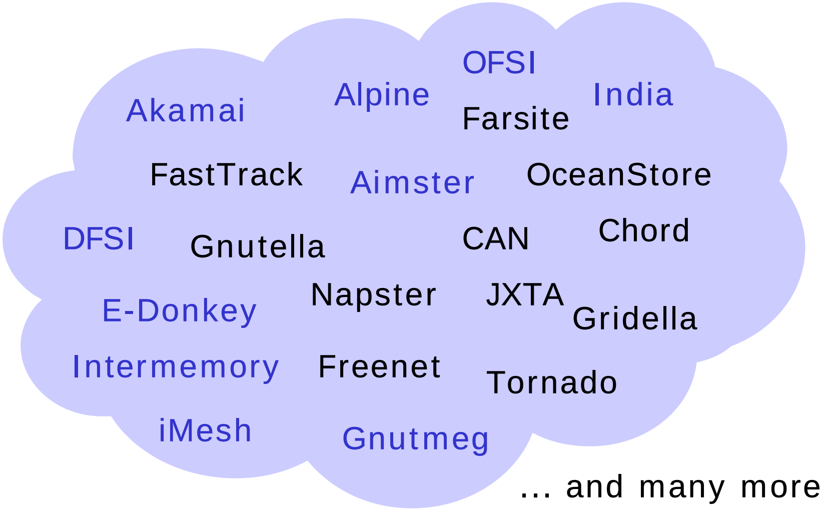
Peer-to-Peer vs. C/S and web-based Systems

	Client-Server		Peer-to-Peer
	Session-based	Web-based	
Coupling	tight	loose	very loose
Comm. Style	asymmetric	asymmetric	symmetric
Number of Clients	moderate (1000)	high (1,000,000)	high (1,000,000)
Number of Servers	few (10)	many (100,000)	none (0)

Coupling vs. Scalability



2. The P2P Cloud



Akamai Alpine OFSI Farsite India

FastTrack Aimster OceanStore

DFSI Gnutella CAN Chord

E-Donkey Napster JXTA Gridella

Intermemory Freenet Tornado

iMesh Gnutmeg

… and many more …

State-of-the-Art Systems

- Napster
- Gnutella
- Freenet
- OceanStore
- Farsite
- FastTrack
- Tornado
- Chord
- CAN
- Gridella

P2P System Models

- Centralized model
  - global index held by a central authority (single point of failure)
  - direct contact between requestors and providers
  - Example: Napster
- Decentralized model
  - Examples: Freenet, Gnutella
  - no global index, no central coordination, global behavior emerges from local interactions, etc.
  - direct contact between requestors and providers (Gnutella) or mediated by a chain of intermediaries (Freenet)
- Hierarchical model
  - introduction of “super-peers”
  - mix of centralized and decentralized model
  - Example: FastTrack (?)

Main P2P Design Requirements

- Resource discovery
- Managing updates
- Scalability
- Robustness and fault tolerance
- Trust assessment and management

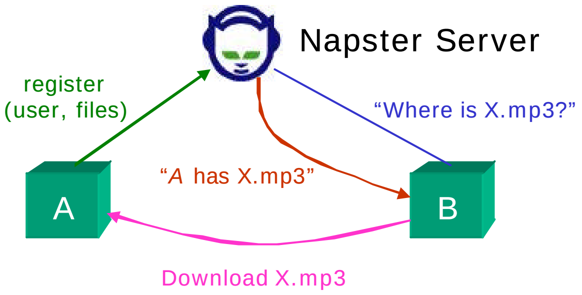
Napster: A brief History

- **May 1999:** Napster Inc. file share service founded by Shawn Fanning and Sean Parker
- **Dec 7 1999:** Recording Industry Association of America (RIAA) sues Napster for copyright infringement
- **April 13, 2000:** Heavy metal rock group Metallica sues Napster for copyright infringement
- **April 27, 2000:** Rapper Dr. Dre sues Napster
- **May 3, 2000:** Metallica's attorney claims 335,000 Internet users illegally share Metallica's songs via Napster
- **July 26, 2000:** Court orders Napster to shut down
- **Oct 31, 2000:** Bertelsmann becomes a partner and drops lawsuit
- **Feb 12, 2001:** Court orders Napster to cease trading copyrighted songs and to prevent subscribers to gain access to content on its search index that could potentially infringe copyrights
- **Feb 20, 2001:** Napster offers $1 billion to record companies (rejected)
- **March 2, 2001:** Napster installs software to satisfy the order

Napster: System Architecture

- Central (virtual) database which holds an index of offered MP3/WMA files
- Clients(!) connect to this server, identify themselves (account) and send a list of MP3/WMA files they are sharing (C/S)
- Other clients can search the index and learn from which clients they can retrieve the file (P2P)
- Combination of client/server and P2P approaches
- First time users must register an account

Napster: Communication Model



Napster: Further Services

- Additionally to its search/transfer features the Napster client offers:
  - A chat program that allows users to chat with each others in forums based on music genre, etc.
  - A audio player to play MP3 files from inside Napster
  - A tracking program to support users in keeping track of their favorite MP3s for later browsing
  - Instant messaging service
- Most of the message types in the protocol deal with hotlist, chat room, and instant messages

Napster: Summary

- (Virtually) centralized system
  - single point of failure ⇒ limited fault tolerance
  - limited scalability (server farms with load balancing)
- Protocol is complicated and inconsistent
- Querying is fast and upper bound for the duration can be given
- “Topology is known”
- Reputation of peers is not addressed
- Many add-on services users like

Gnutella: A brief History

- Developed in a 14 days “quick hack” by Nullsoft (winamp)
- Originally intended for exchange of recipes
- Timeline:
  - Published under GNU General Public License on the Nullsoft web server
  - Taken off after a couple of hours by AOL (owner of Nullsoft)
  - This was enough to “infect” the Internet
  - Gnutella protocol was reverse engineered from downloaded versions of the original Gnutella software
  - Third-party clients were published and Gnutella started to spread

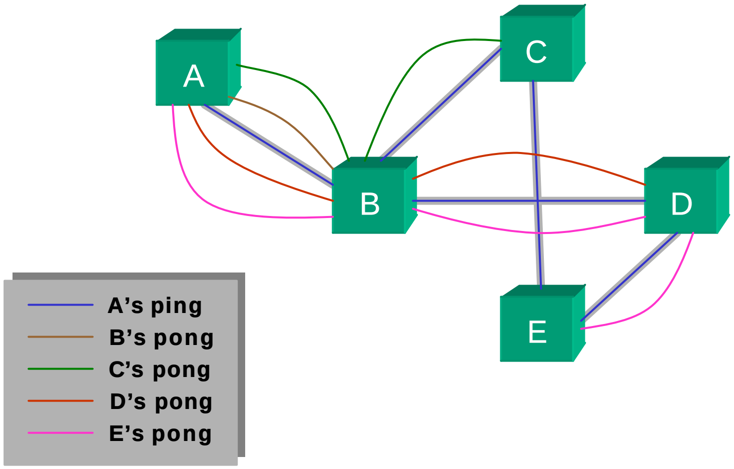
Gnutella: System Architecture

- No central server
  - cannot be sued (Napster)
- Constrained broadcast
  - Every peer sends packets it receives to all of its peers (typically 4)
  - Life-time of packets limited by time-to-live (typically set to 7)
  - Packets have unique ids to detect loops
- Hooking up to the Gnutella systems requires that a new peer knows at least one Gnutella host
  - gnutellahosts.com:6346
  - Outside the Gnutella protocol specification

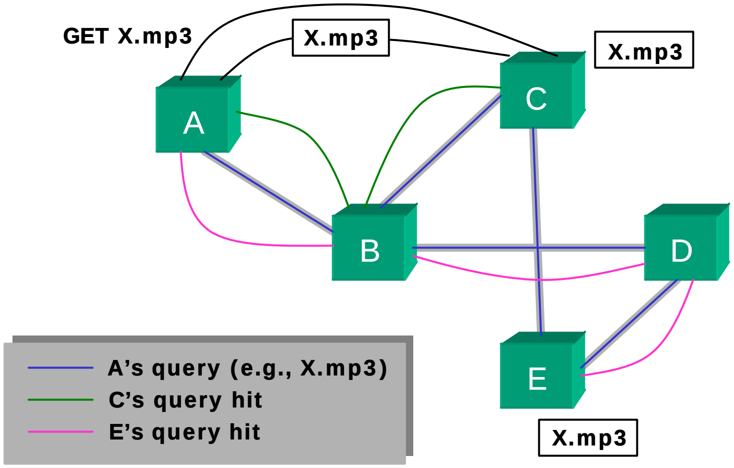
Gnutella: Protocol Message Types

Type	Description	Contained Information
Ping	Announce availability and probe for other servents	None
Pong	Response to a ping	IP address and port# of responding servent; number and total kb of files shared
Query	Search request	Minimum network bandwidth of responding servent; search criteria
QueryHit	Returned by servents that have the requested file	IP address, port# and network bandwidth of responding servent; number of results and result set
Push	File download requests for servents behind a firewall	Servent identifier; index of requested file; IP address and port to send file to

Gnutella: Meeting Peers (Ping/Pong)



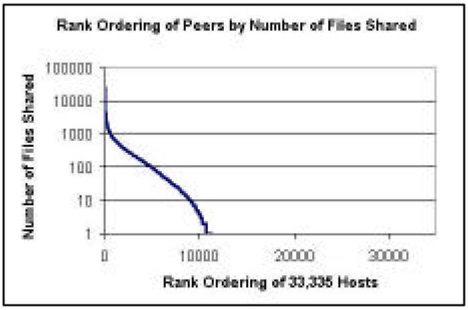
Gnutella: Searching (Query/QueryHit/GET)



Free-riding on Gnutella [Adar00]

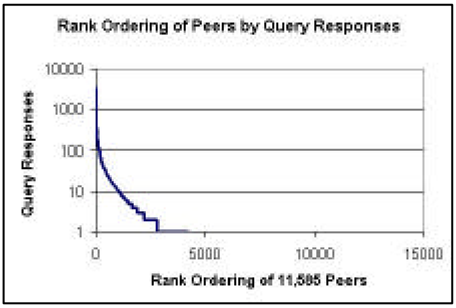
- 24 hour sampling period:
  - 70% of Gnutella users share no files
  - 50% of all responses are returned by top 1% of sharing hosts
- A social problem not a technical one
- Problems:
  - Degradation of system performance: collapse?
  - Increase of system vulnerability
  - "Centralized" ("backbone") Gnutella ⇔ copyright issues?
- Verified hypotheses:
  - H1: A significant portion of Gnutella peers are free riders.
  - H2: Free riders are distributed evenly across domains
  - H3: Often hosts share files nobody is interested in (are not downloaded)

Free-riding Statistics - 1 [Adar00]



- H1: Most Gnutella users are free riders
- Of 33,335 hosts:
  - 22,084 (66%) of the peers share no files
  - 24,347 (73%) share ten or less files
  - Top 1 percent (333) hosts share 37% (1,142,645) of total files s hared
  - Top 5 percent (1,667) hosts share 70% (1,142,645) of total files shared
  - Top 10 percent (3,334) hosts share 87% (2,692,082) of total files shared

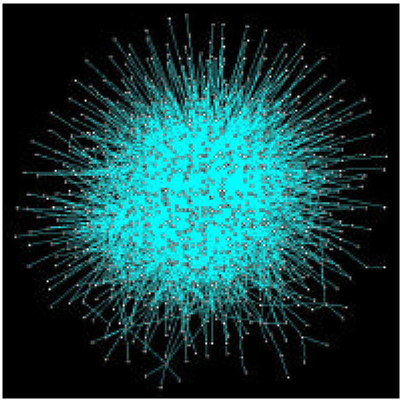
Free-riding Statistics - 2 [Adar00]



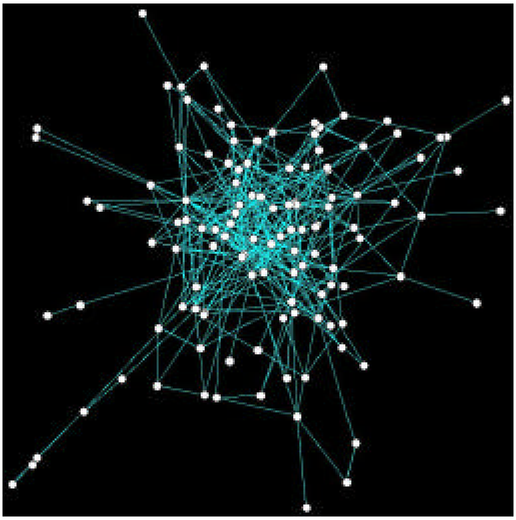
- H3: Many servents share files nobody downloads
- Of 11,585 sharing hosts:
  - Top 1% of sites provide nearly 47% of all answers
  - Top 25% of sites provide 98% of all answers
  - 7,349 (63%) never provide a query response

Topology of Gnutella [Jovanovic01]

- Small-world properties verified ("find everything close by")
- Backbone + outskirts

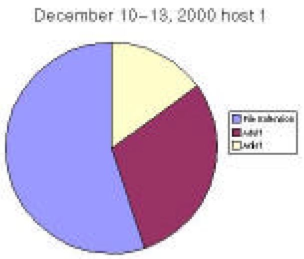


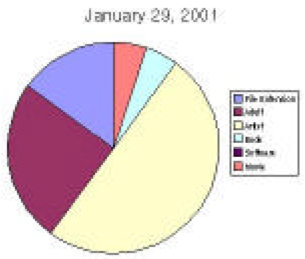
Gnutella Backbone [Jovanovic01]



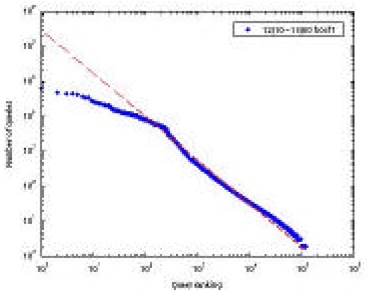
Categories of Queries [Sripanidkulchai01]

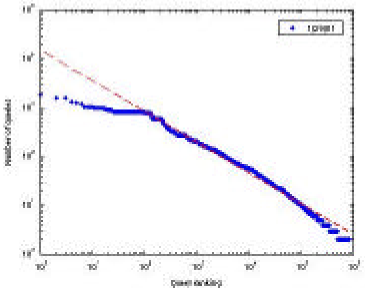
- Categorized top 20 queries





Popularity of Queries [Sripanidkulchai01]

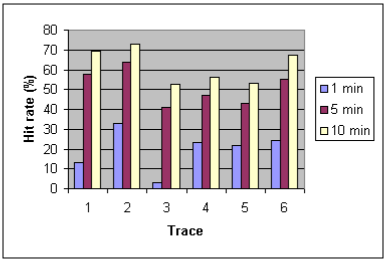


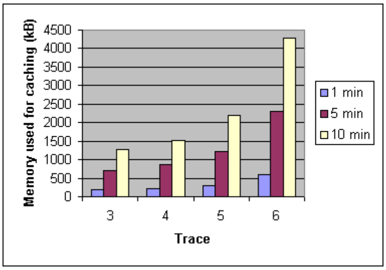


- Very popular documents are approximately equally popular
- Less popular documents follow a Zipf-like distribution (i.e., the probability of seeing a query for the **i**$^{\text{th}}$ most popular query is proportional to **1 / (i$^{\text{alpha}}$)**
- Access frequency of web documents also follows Zipf-like distributions ⇒ caching might also work for Gnutella

Caching in Gnutella [Sripanidkulchai01]

- Average bandwidth consumption in tests: 3.5Mbps





- Best case: trace 2 (73% hit rate = 3.7 times traffic reduction)

Gnutella: Bandwidth Barriers

- Clip2 measured Gnutella over 1 month:
  - typical query is 560 bits long (including TCP/IP headers)
  - 25% of the traffic are queries, 50% pings, 25% other
  - on average each peer seems to have 3 other peers actively connected
- Clip2 found a scalability barrier with substantial performance degradation if queries/sec > 10:

10 queries/sec
* 560 bits/query
* 4 (to account for the other 3 quarters of message traffic)
* 3 simultaneous connections

67,200 bps
⇒ 10 queries/sec maximum in the presence of many dialup users
⇒ won’t improve (more bandwidth - larger files)

Gnutella: Summary

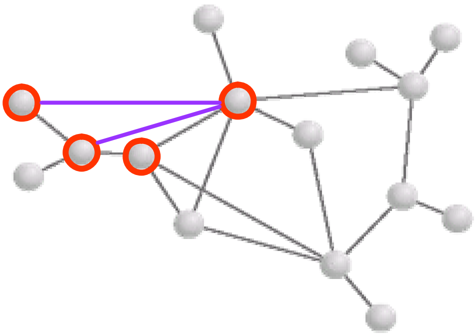
- Completely decentralized
- Hit rates are high
- High fault tolerance
- Adopts well and dynamically to changing peer populations
- Protocol causes high network traffic (e.g., 3.5Mbps). For example:
  - 4 connections C / peer, TTL = 7
  - 1 ping packet can cause $2*\sum_{i=0}^{TTL} C*(C-1)^{i} = 26,240$ packets
- No estimates on the duration of queries can be given
- No probability for successful queries can be given
- Topology is unknown ⇒ algorithms cannot exploit it
- Free riding is a problem
- Reputation of peers is not addressed
- Simple, robust, and scalable (at the moment)

Freenet: System Architecture

- Adaptive P2P system which supports publication, replication, and retrieval of data
- Protects anonymity of authors and readers
  - infeasible to determine the origin or destination of data
  - difficult for a node to determine what it stores (files are sent and stored encrypted)

  ⇒ nobody can be sued
- Requests are routed to the most likely physical location
  - no central server as in Napster
  - no constrained broadcast as in Gnutella
- Files are referred to in a location independent way
- Dynamic replication of data

Freenet: Searching [Hong01]



- Graph structure actively evolves over time
  - new links form between nodes
  - files migrate through the network

  ⇒ adaptive routing

Freenet: Key Types

- Keys are represented as Uniform Resource Identifiers (URIs): freenet:*keytype@data*
- Keyword Signed Keys (KSK)
- Signature Verification Keys (SVK)
- SVK Subspace Keys (SSK)
- Content Hash Keys (CHK)
- Keys can be used for indirections, e.g., KSK → CHK

Freenet: Keyword Signed Keys (KSK)

- User chooses a short descriptive text *sdtext* for a file, e.g., text/computer-science/esec2001/p2p-tutorial
- *sdtext* is used to deterministically generate a public/private key pair
- The public key part is hashed and used as the file key
- The private key part is used to sign the file (not completely save - dictionary attacks)
- The file itself is encrypted using *sdtext* as key
- For finding the file represented by a KSK a user must know *sdtext* which is published by the provider of the file
- Example: freenet:KSK@text/books/1984.html

Freenet: SVKs and SSKs

- Allows people to make a subspace, i.e., controlling a set of keys
- Based on the same public key system as KSKs but purely binary and the key pair is generated randomly
- People who trust the owner of a subspace will also trust documents in the subspace because inserting documents requires knowing the subspace’s private key
- For retrieval: *sdtext* and public key of subspace are published
- SSKs are the client-side representation of SVKs with a document name
- Examples:
  - freenet:SVK@HDOKWIUn10291jqd097euojhd01
  - freenet:SSK@1093808jQWIOEh8923kIah10/text/books/1984.html

Freenet: Content Hash Keys (CHK)

- Derived from hashing the contents of the file ⇒ pseudo-unique file key to verify file integrity
- File is encrypted with a randomly-generated encryption key
- For retrieval: CHK and decryption key are published (decryption key is never stored with the file)
- Useful to implement updating and splitting, e.g., in conjunction with SVK/SSK:
  - to store an updateable file, it is first inserted under its CHK
  - then an indirect file that holds the CHK is inserted under a SSK

  ⇒ others can retrieve the file in two steps given the SSK

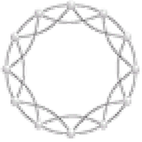
  ⇒ only the owner of the subspace can update the file
- Example: freenet:CHK@UHE92hd92hseh912hJHEUh1928he902

Freenet: Inserting Files

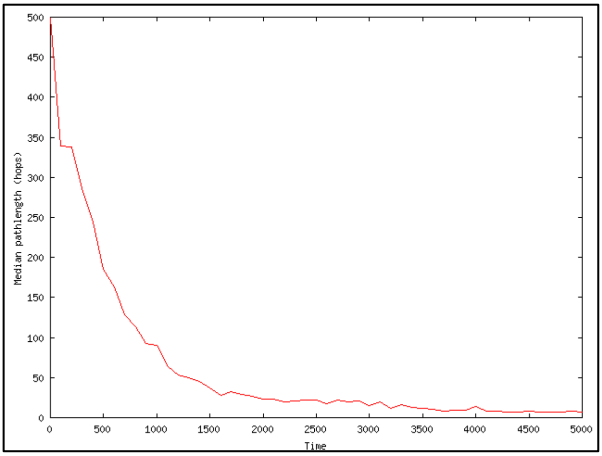
- First a key (KSK, CHK, etc.) is calculated
- An insert message with this proposed key and a hops-to-live value is sent to the local peer
- Then every peer checks whether the proposed key is already present in its local store
  - yes ⇒ return stored file (original requester must propose new key)
  - no ⇒ route to next peer for further checking (routing uses the same key similarity measure as searching)
  - continue until hops-to-live are 0 or failure
- Hops-to-live is 0 and no collision was detected ⇒ insert file along the path established by initial query

Freenet: Evolution of Path Length [Hong01]

- 1000 identical nodes
- max 50 data items/node
- max 200 references/node
- Initial references: (i-1, i-2, i+1, i+2) mod n
- node key: hash(i)



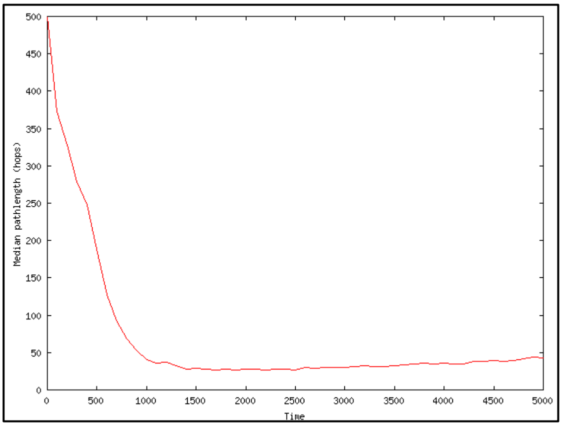
- each time-step:
  - randomly query/insert
  - HopsToLive=20
- every 100 time-steps: 300 requests (HTL=500) from random nodes and measure actual path length (failure=500).



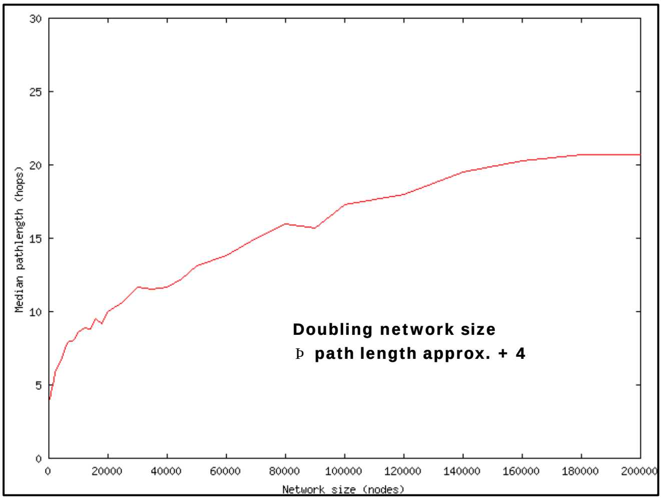
median path length 500 ® 6

Freenet: The Importance of Routing [Hong01]

- Existence of short paths is not enough – they must be found
- Adaptivity helps Freenet to find good paths
- A random-routing network: median path length ~ 50

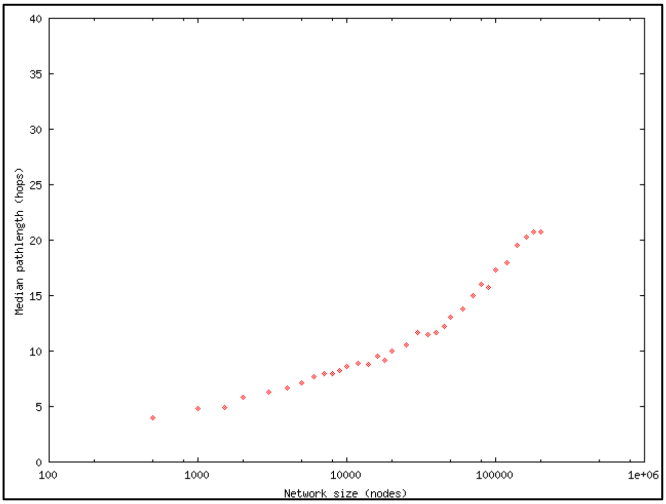


Freenet: Scalability [Hong01]



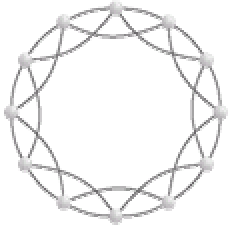
Doubling network size
Þ path length approx. + 4

Freenet: Scalability (log Scale) [Hong01]

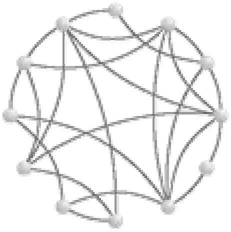


Why does it work? It’s a small World! [Hong01]

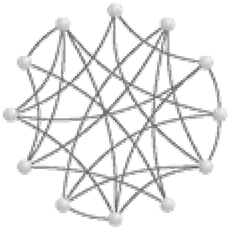
- Milgram: 42 out of 160 letters from Oregon to Boston (~ 6 hops)
- Watts: between order and randomness
  - short-distance clustering + long-distance shortcuts



Regular graph:
n nodes, k nearest neighbors
Þ path length ~ n/2k
4096/16 = 256



Rewired graph (1% of nodes):
path length ~ random graph
clustering ~ regular graph



Random graph:
path length ~ log (n)/log(k)
~ 4

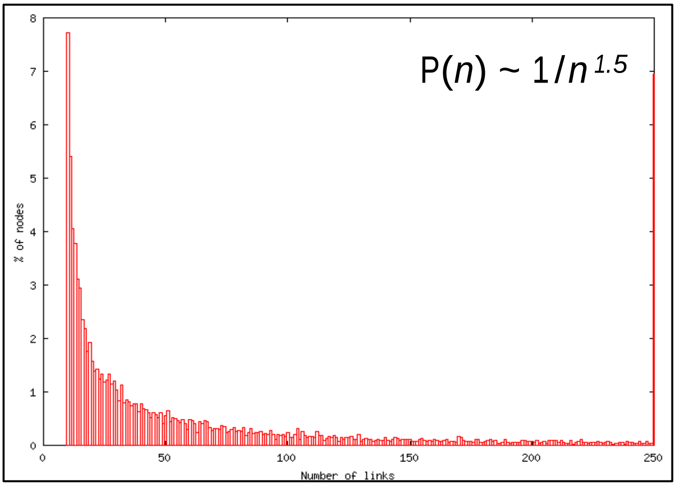
Links in the small World [Hong01]

- “Scale-free” link distribution
  - Scale-free: independent of the total number of nodes
  - Characteristic for small-world networks
  - The proportion of nodes having a given number of links *n* is:

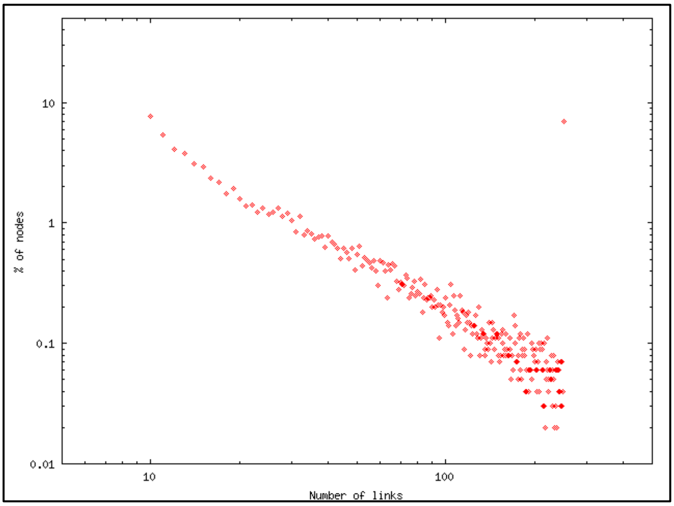
    $$P(n) = 1/n^{k}$$

  - Most nodes have only a few connections
  - Some have a lot of links: important for binding disparate regions together

Freenet: Links in the small World [Hong01]



Freenet: “Scale-free” Link Distribution [Hong01]



Freenet: Summary

- Completely decentralized
- High fault tolerance
- Robust and scalable
- Automatic replication of content
- Adopts well and dynamically to changing peer populations
- Spam content less of a problem (subspaces)
- Adaptive routing preserves network bandwidth
- No estimates on the duration of queries can be given
- No probability for successful queries can be given
- Topology is unknown ⇒ algorithms cannot exploit it
- Routing “circumvents” free-riders
- Reputation of peers is not addressed
- Supports anonymity of publishers and readers

OceanStore: System Overview

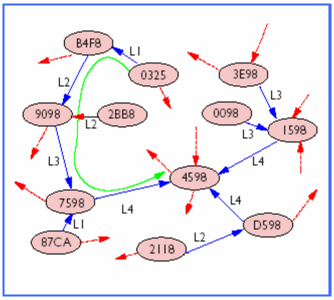
- Internet-based, distributed, global storage infrastructure
- Consists of possibly millions of cooperating servers
- Data is split up in fragments which are stored redundantly on servers
- Sophisticated replication to ensure data availability and performance (introspection-based self-tuning)
- High fault tolerance: monitoring, erasure (m-of-n) coding, self-verifying fragments, automatic repair, byzantine update commitment, etc.
- Automatically recovers from server and network failures, incorporates new resources, and adjusts to usage patterns
- Each participating server is a client and a server => P2P system

OceanStore: Routing Infrastructure

- Data identified by a key must be located => all fragments must be located and verified
- Tapestry subsystem
  - Self-organizing routing and object location system
  - Routing: hashed suffix routing

Tapestry routing example. A potential path for a message originating at node 0325 destined for node 4598.Tapestry routes the message through nodes **** ? ***8 ? **98 ? *598 ? 4598, where asterisks represent wildcards. The role each hop plays in the path is marked with a level number.

[Rhea01]



Farsite: System Overview

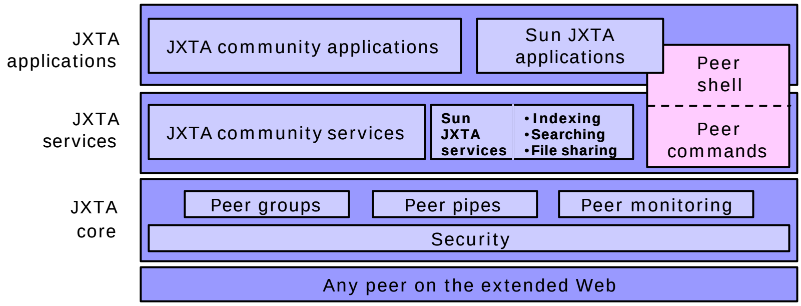
- Serverless, distributed file system that does not assume mutual trust among the client computers on which it runs
  - Logically: the system functions as a central file server
  - Physically: there is no central server machine; instead, a group of desktop client computers collaboratively establish a virtual file server that can be accessed by any of the clients.
- System properties:
  - Global name space for files
  - Location-transparent access to files
  - Reliability: Multiple encrypted replicas on clients
  - Hierarchical directory structure via a distributed directory service
- Only simulations are published so far
  - no algorithms publicly available
  - no system architecture and design publicly available

FastTrack: System Overview

- A P2P library that powers some of the currently most successful P2P search engines (KaZaA, Morpheus, Grokster)
- Proprietary, encrypted protocol
- No system/protocol information available
- giFT: reverse engineering effort (http://gift.sourceforge.net)
- Presumed system architecture
  - Nodes holding data
  - Hierarchical supernodes providing search capabilities and forwarding search requests to other superpeers
  - A central super-superpeer

Project JXTA (SUN)

- A network programming platform for P2P systems
  - 3-layer architecture
  - 6 XML-based protocols: discovery, membership, routing, …
  - abstractions: peer groups, pipes, advertisements, …
- Goal: a uniform platform for applications using P2P technology and for various P2P systems to interact



3. Data Storage and Search in P2P Systems

- Problem
  - Peers in a P2P system need to share information
  - Central database would contradict the P2P paradigm
  - Can a distributed database be supported by peers without central control
- Example
  - Directory of all files in a file-sharing system
- Basic operations in a database
  - Searching data (efficiently)
  - Updating data (consistently)
- Basic operations in a peer network
  - Joining the network (efficiently)
  - Leaving the network (consistently)

Approaches

- B2B servers, Napster, eBay etc.
  - Central database !
- Gnutella
  - Search requests are broadcast
  - Anecdote: the founder of Napster computed that a single search request (18 Bytes) on a Napster community would generate 90 Mbytes of data transfers. [http://www.darkridge.com/~jpr5/doc/gnutella.html]
- Decentralization of data management and efficiency seem to contradict each other !

Question

- Can a set of peers without central coordination provide
  - **efficient** search on a distributed database
  - while the storage space at each peer is compared to the whole database **small**
- Efficient search
  - searchtime(query) ≈ Log(size(database))
  - #searchmessages(query) ≈ Log(size(database))
- Small storage space
  - storagespace(agent) ≈ Log(size(database))
- Answer
  - In principle, yes !
  - Requires **scalable data access structures**
  - Autonomy needs to be preserved !

Note on Search Time

- Search time is not a problem when the peer network is a small world graph
- Always exists a path of length Log(size(database))
- Explore all possible paths (like e.g. Gnutella)

- Problem: number of messages required !

Assumptions

- There exists a physical networks where peers can contact each other
- Peers have a physical address (called **reference** in the following)
- Properties of the network connections can be taken into account (latency, cost etc.)
- There exists a mechanism that allows peers to make initial contact with some other peers in the network (e.g. bootstrap list)
- Data objects are identified by keys k

Problem Definition

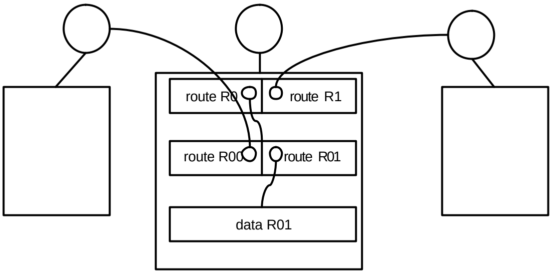
- Peers with address $\mathbf{p_d}$ store data items **d** that are identified by a key **k**
- In order to locate a peer that stores **d** we have to search for key **k** in the lookup table consisting of tuples of form $\mathbf{(k, p_d)}$
- Thus, the database we have to manage consists of the key-value pairs $\mathbf{(k, p_d)}$
- We do not further consider the storage of data items **d**
- Further, one can distinguish search types
  - equality, prefix, range, containment, similarity search

Common Characteristics

- The information **(k, p_d)** is distributed over the peers
  - Each peer stores some of this information locally
- Search request for **k** can be addressed to every peer with address **p**
  - we write **p->search(k)**
- If a peer has the information not locally available it routes the request to another peer **p'**
  - i.e. it sends a request **p'->search(k)**
  - for selecting **p'** it maintains **routing information**

P2P Data Access Structures

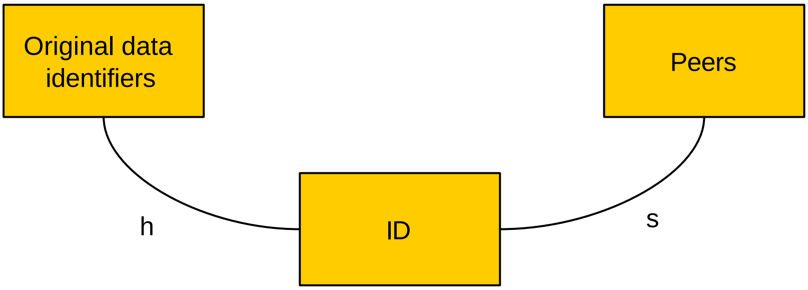
- Every peer maintains a small fragment of the database and a routing table
- The peers implement a routing strategy
- Replication can be used to increase robustness



Directed Routing

- In order to cut down the message number requests can be routed only to peers that may store the data
- Therefore we must associate with peers the set of data identifiers (keys) they store
- We assume a set **ID** of data keys is given
  - Either the original data keys or there exists some (hash) function for mapping the data keys into **ID**
- Each peer is associated with an element or a subset of **ID**
- Different methods to establish this association

Data Identifiers



Differences

- Structure and content of routing information
  - Search request propagation strategy
  - Processing of updates
  - Strategy to construct routing information
  - Joining and leaving the network
- Robustness
  - use of replication
- Scalability, Complexity
  - Efficiency of search, messages and updates
  - Maintenance of routing tables
- Search types supported
- Autonomy
  - Association of specific role with peer (address)
- Global knowledge
  - nature of keys, number of addresses

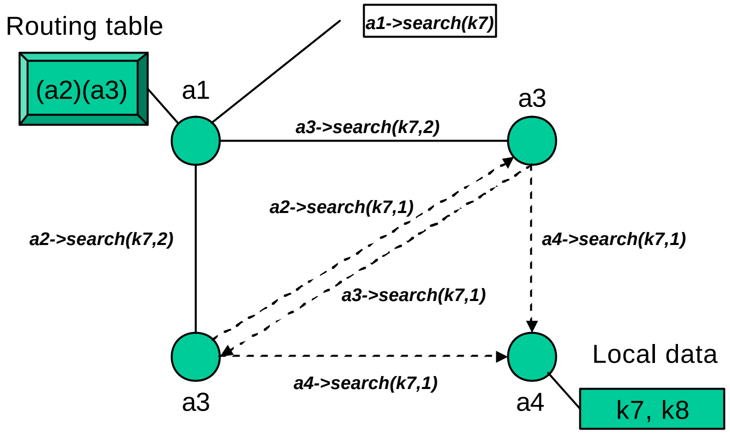
Approaches

- Existing P2P Systems
  - Gnutella
  - Freenet
- Research
  - CHORD
  - Content-Addressable Networks
  - Tapestry
  - P-Grid

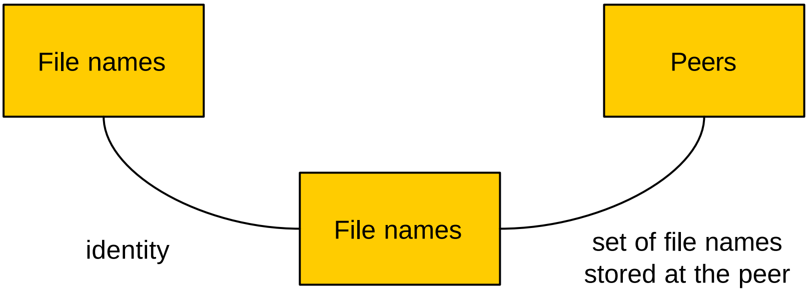
Gnutella

- Each peer knows a fixed number of **other peers**, e.g. 4
- Other peers are found randomly, e.g. through ping messages
- Search requests are forwarded to those peers, with a limited **time-to-live**, e.g. 7
- Peers can answer the request if they store the corresponding file

Gnutella Access Structure



Gnutella Data Identifiers



File names

Peers

File names

identity

set of file names
stored at the peer

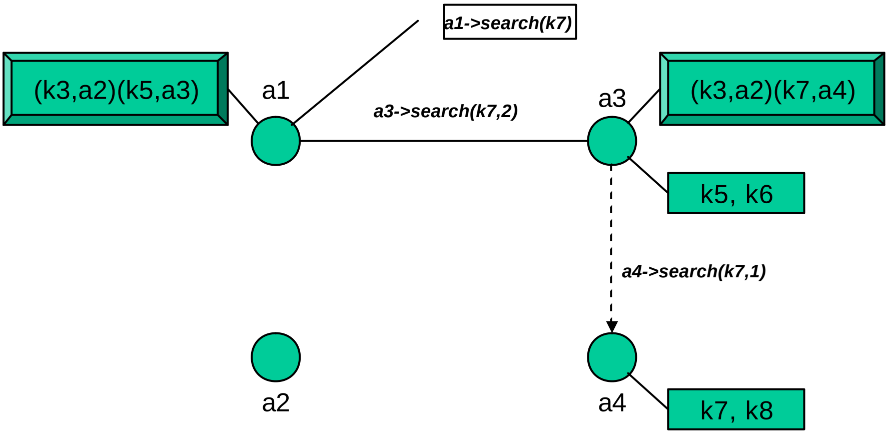
Gnutella Discussion

- Search types
  - Any possible string comparison
- Scalability
  - Search very poor with respect to number of messages
  - Probably search time O(Log n) due to small world property
  - Updates excellent: nothing to do
  - Routing information: low cost
- Robustness
  - High, since many paths are explored
- Autonomy
  - Storage: no restriction, peers store the keys of their files
  - Routing: peers are target of all kinds of requests
- Global Knowledge
  - None required

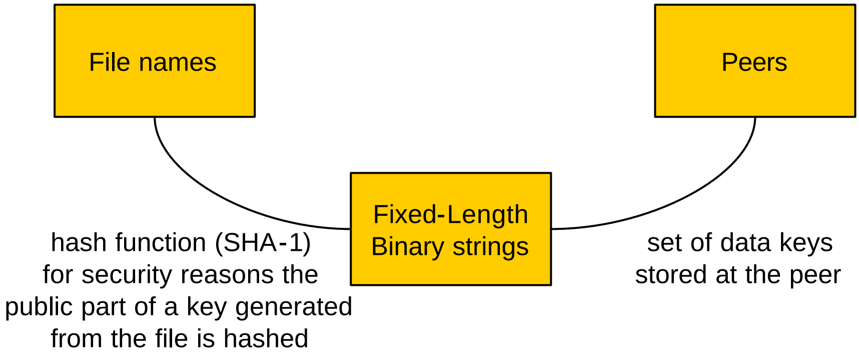
Freenet

- Each peer knows a fixed number of other peers **and a key,** that the peers store
- Search requests are routed to the peer with the **most similar key**
  - If not successful the next similar key is used etc.
  - Similarity based on lexicographic distance (any other measure would be possible as well)
- Search requests have limited life time, e.g. 500
- Peers can answer requests if they store the requested items
- When the answer is passed back, the intermediate peers can use it to update their routing information

Freenet Access Structure



Freenet Data Identifiers



File names

Peers

Fixed-Length Binary strings

hash function (SHA-1)
for security reasons the
public part of a key generated
from the file is hashed

set of data keys
stored at the peer

Freenet Searching

- Peers store keys, data and addresses

Key	Data	Address
8e47683isdd0932uje89	ZT38hwe01h02hdhgdzu	tcp/125.45.12.56:6474
456r5wero04d903iksd0	Rhweui12340jhd091230	tcp/67.12.4.65:4711
f3682jkjdn9ndaqmmxia	eqwe1089341ih0zuhge3	tcp/127.156.78.20:8811
wen09hjfdh03uhn4218	erwq038382hjh3728ee7	tcp/78.6.6.7:2544
712345jb89b8nbopledh		tcp/40.56.123.234:1111
d0ui43203803ujoejghh		tcp/128.121.89.12:9991

- As with Gnutella search requests have
  - limited life time, but typical higher, e.g., 500
  - message identifiers to avoid cycles

Freenet Searching

- If a search request arrives
  - Either the data is in the table
  - Or the request is forwarded to the addresses with the most similar keys (lexicographic similarity, edit distance) till a answer is found
- If an answer arrives
  - The key and address of the answer are inserted into the table
  - The least recently used key is evicted

Joining Freenet

- As with Gnutella joining requires knowledge of initial node
- Quality of routing should improve over time when searches and updates arrive
  - Node is listed under certain key in routing tables
  - Therefore gets more requests for similar keys
  - Therefore tends to store more entries with similar keys (clustering) when receiving results and caching them

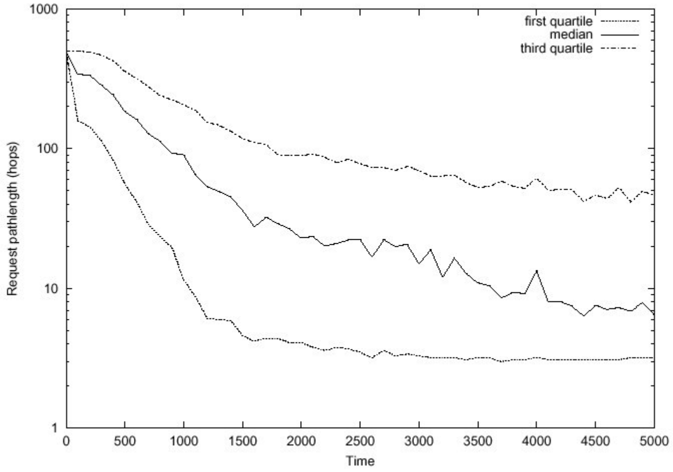
Freenet: Update (Insertion)

- First a key of the file is calculated
- An insert message with this proposed key and a hops-to-live value is sent to the neighbors
- Then every peer checks whether the proposed key is already present in its local store
  - yes ⇒ return stored file (original requester must propose new key)
  - no ⇒ route to next peer for further checking (routing uses the same key similarity measure as searching)
  - continue until hops-to-live are 0 or failure
- Hops-to-live is 0 and no collision was detected ⇒ insert file along the path established by initial query

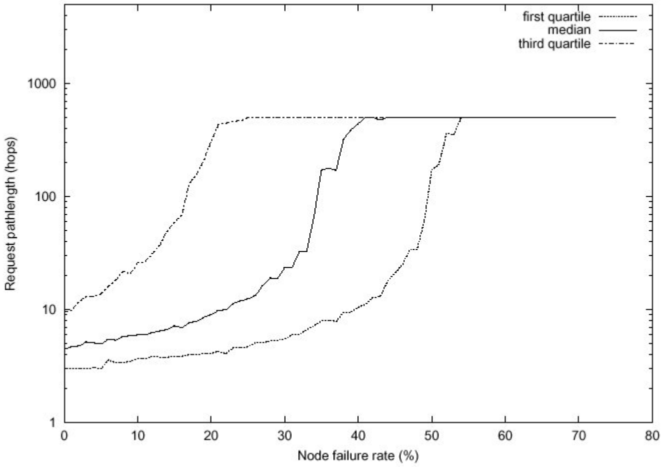
Freenet Simulation

- Test network with 1000 nodes
- Each node stores 50 data items
- Routing table size of 250
- Initial topology: ring
- Time-to-live for inserts: 20
- Time-to-live for searches: 500

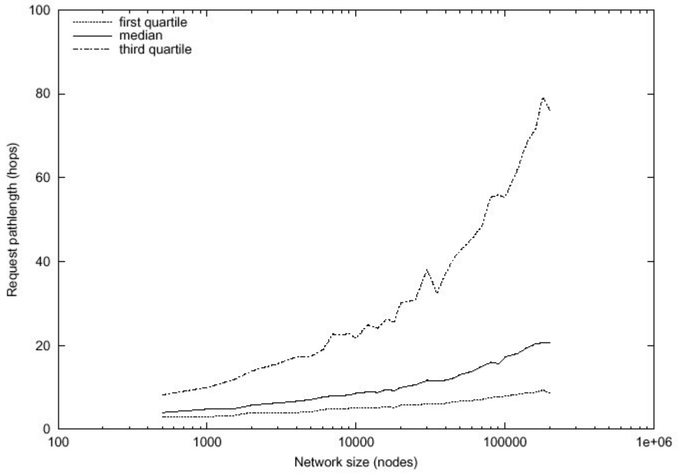
Evolution of Path Length



Path Length under Network Failure



Request Path Length vs. Network Size



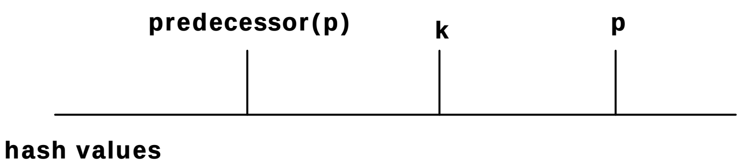
Freenet Discussion

- Search types
  - Only equality, exact keys need to be known, e.g., published in a directory
  - However, if keys were not hashed, semantic similarity might be used for routing
- Scalability
  - Search good, seems to be O(Log n) in number of nodes n
  - Update good, like search
  - Routing information: a bootstrapping phase is required
- Robustness
  - Good, since alternative paths are explored
- Autonomy
  - Storage no restriction
  - Routing: dependency between stored keys and received requests
- Global Knowledge
  - Key hashing

CHORD Consistent Hashing

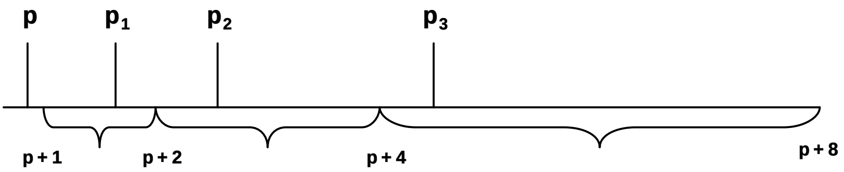
- Based on a hashing of search keys and peer addresses on binary keys of length **m**
- Each peer with hashed identifier p is responsible (=stores values associated with the key) for all hashed keys k such that

$$k \in \,]\, \text{predecessor}(p), p \,]$$

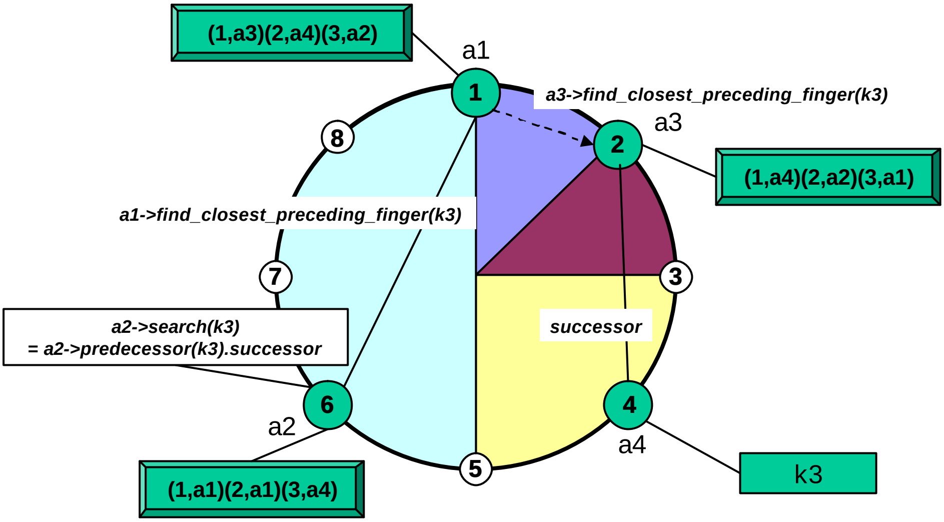


CHORD Routing Table

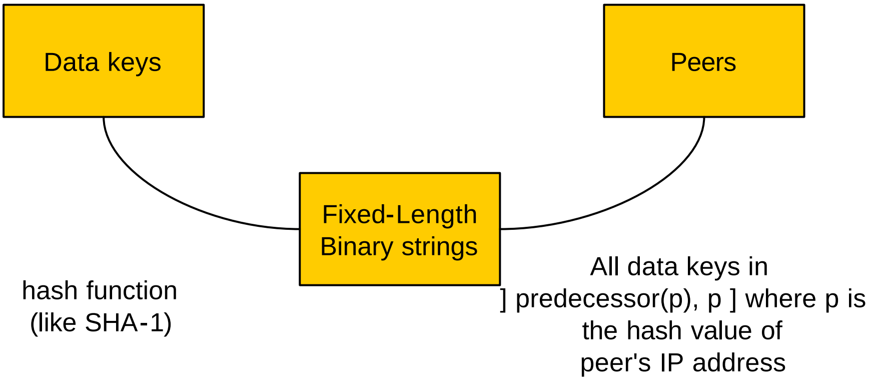
- Each peer p stores a “finger table” consisting of the first peer with hashed identifier p_i such that
  p_i =successor($p+2^{i-1}$) for i=1,..,m
- We write also p_i = finger(i, p)
- A search algorithm ensures the reliable location of the data
  - Complexity O(log n), n nodes in the network



CHORD Access Structure (m=3)


(1,a3)(2,a4)(3,a2)
a1
1
a3->find_closest_preceding_finger(k3)
a3
2
(1,a4)(2,a2)(3,a1)
8
a1->find_closest_preceding_finger(k3)
7
3
a2->search(k3)
= a2->predecessor(k3).successor
successor
6
4
a2
a4
5
k3
(1,a1)(2,a1)(3,a4)

CHORD Data Identifiers

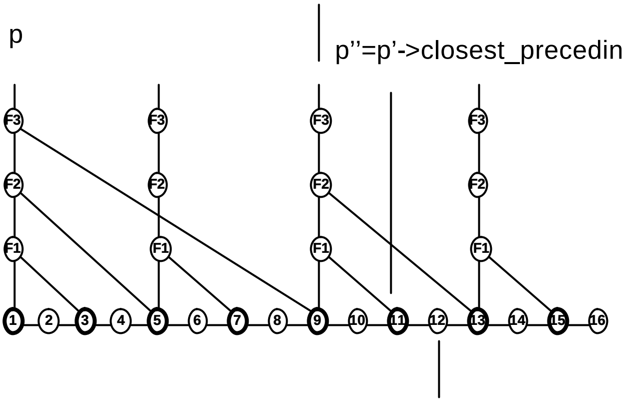


CHORD Search Example (m=4)

p'=p->closest_preceding_finger(k)

p

p''=p'->closest_preceding_finger(k)



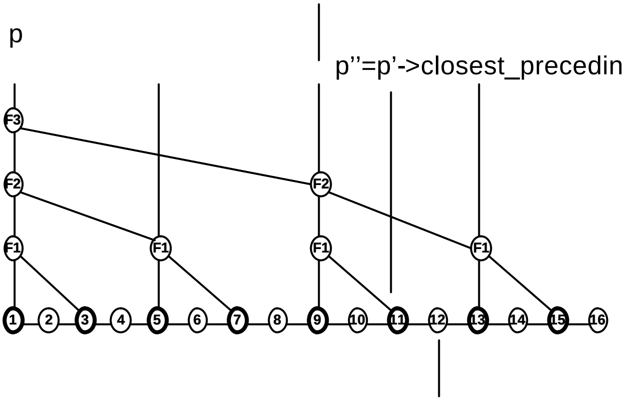
k ∈ (p'',successor(p'')]

Another Perspective on CHORD (as a Tree)

p'= p->closest_preceding_finger(k)

p

p''=p'->closest_preceding_finger(k)



k ∈ (p'',successor(p'')]

CHORD Search Algorithm in Detail

- Searching for k: find predecessor of k, then go to successor

- p->find_predecessor(k)

```
{p' := p;
while ($k \notin (p', p'.successor]$ )
  p':=p'->closest_preceding_finger(k);
return p'}
```

- p-> closest_preceding_finger(k);

```
{for i=m down to 1
   if(finger(i, p) $\in$ (n, k) )
      return finger(i, p);
return p}
```

Joining the CHORD Network

- Initialize predecessors and fingers of the node
  - Uses an existing node to identify them by search
  - Naive approach requires O(m Log n) searches
  - Optimization: if i-th finger interval empty then finger(i)=finger(i+1)
  - Reduces runtime to O(Log^2 n)
- Update predecessors and fingers of existing nodes
  - Search through the predecessors
  - Runtime O(log^2 n)
- Notify higher level software that the new node is now responsible for its keys

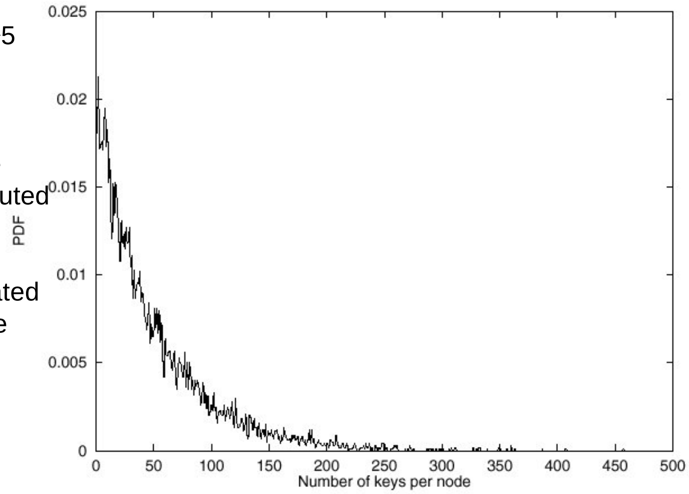
Distribution of Keys per Node

Network size n=10^5

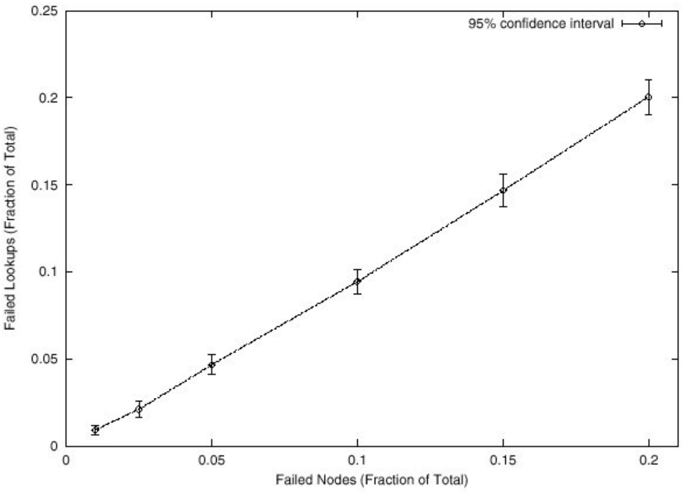
5 10^5 keys

Node identifiers are not uniformly distributed

Proposed solution: map multiple unrelated logical nodes to one physical node



Failure Resilience

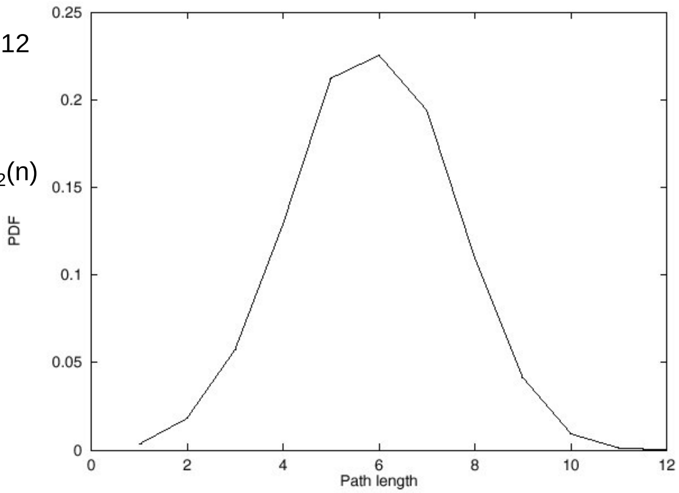


Length of Search Paths

Network size n=2^12

100 2^12 keys

Path length ½ $\mathrm{Log}_2(n)$



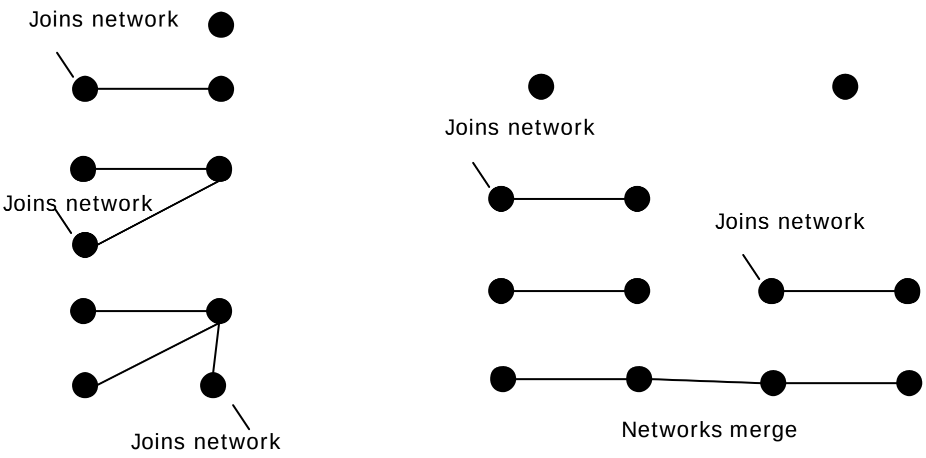
CHORD Discussion

- Search types
  - Only equality
- Scalability
  - Search O(Log n) w.h.p.
  - Update requires search, thus O(Log n) w.h.p.
  - Construction: O(Log^2 n) if a new node joins
- Robustness
  - Replication might be used by storing replicas at successor nodes
- Autonomy
  - Storage and routing: none
  - Nodes have by virtue of their IP address a specific role
- Global knowledge
  - Mapping of IP addresses and data keys to key common key space
  - Single Origin

Evolution of Network

Single Initial Node (e.g. CHORD)

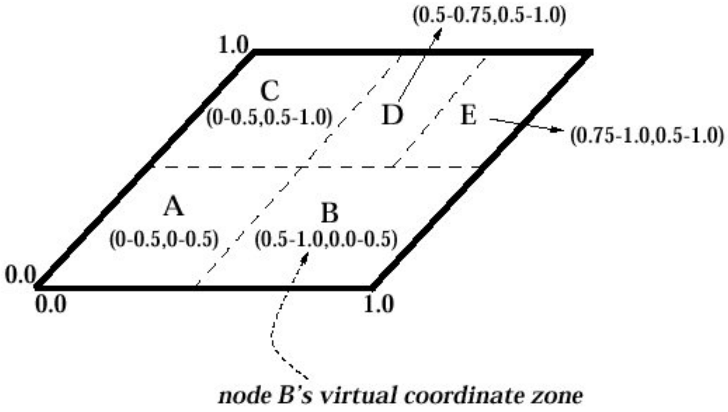
Multiple Initial Nodes (e.g. Gnutella, Freenet)



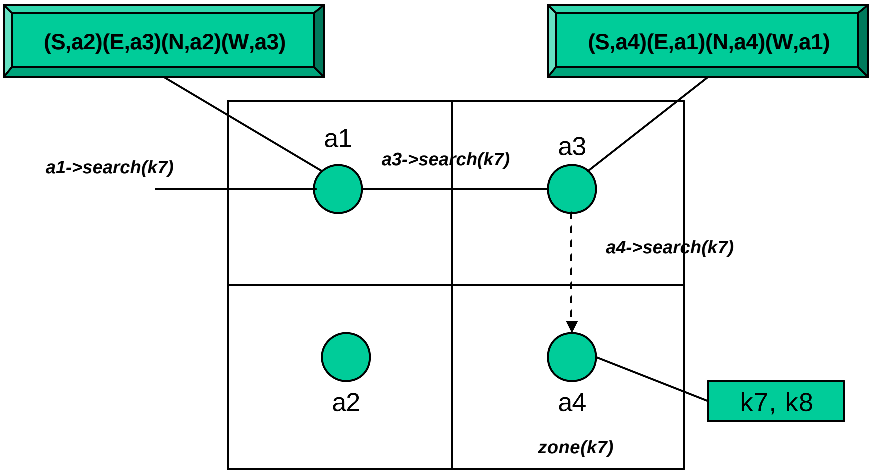
Content-Addressable Networks (CAN)

- Based on hashing of keys into a **d-dimensional space (a torus)**
- Each peer is responsible for keys of a subvolume of the space (a zone)
- Each peer stores the peers responsible for the neighboring zones for routing
- Search requests are greedily forwarded to the peers in the closest zones
- Assignment of peers to zones depends on a random selection made by the peer

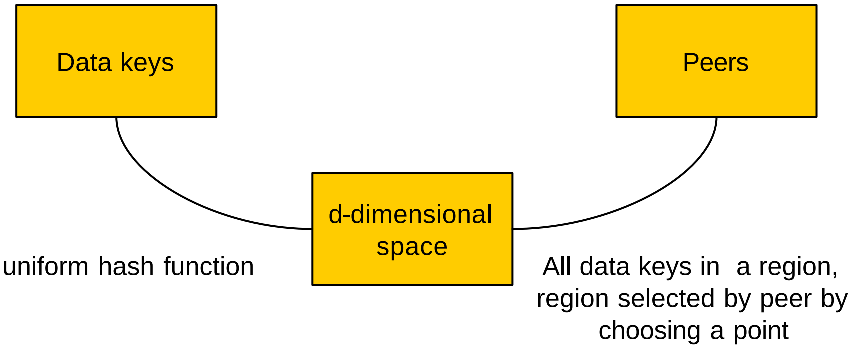
CAN Zones (2D Space)



CAN Access Structure



CAN Data Identifiers



Data keys

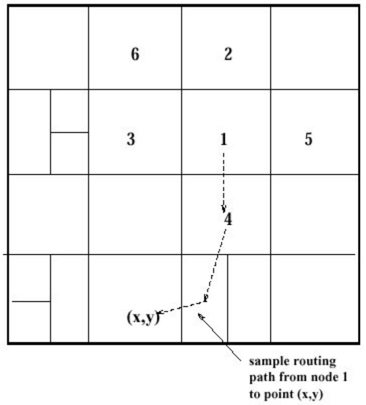
Peers

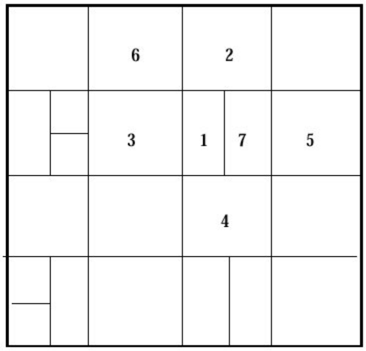
d-dimensional space

uniform hash function

All data keys in a region, region selected by peer by choosing a point

Joining the CAN Network





1's coordinate neighbor set = {2,3,4,5}
7's coordinate neighbor set = { }

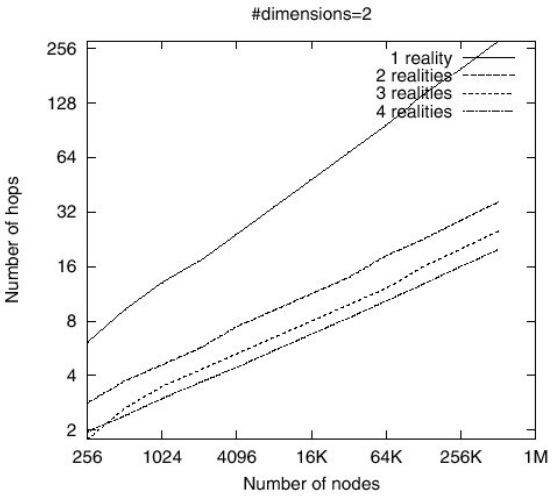
1's coordinate neighbor set = {2,3,4,7}
7's coordinate neighbor set = {1,2,4,5}

Neighboring nodes inform each other about new neighbors

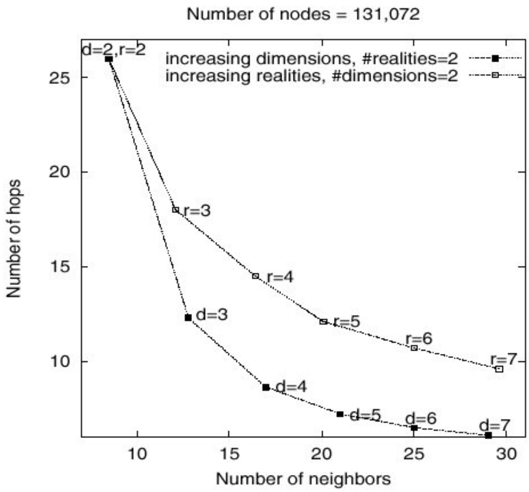
CAN Refinements

- Multiple Realities
  - We can have r different coordinate spaces
  - Nodes hold a zone in each of them
  - Creates r replicas of the (key, value) pairs
  - Increases robustness
  - Reduces path length as search can be continued in hte reality where the target is closest
- Overloading zones
  - Different peers are responsible for the same zone
  - Splits are only performed if a maximum occupancy (e.g. 4) is reached
  - Nodes know all other nodes in the same zone
  - But only one of the neighbors

CAN Path Length



Increasing Dimensions and Realities



CAN Discussion

- Search types
  - equality only
  - however, could be extended using spatial proximity
- Scalability
  - Search and update: good O(d n^(1/d)), depends on configuration of d
  - Construction: good
- Robustness
  - Good with replication
- Autonomy
  - Free choice of coordinate zone
- Global Knowledge
  - Hashing of keys to coordinates, realities, overloading
  - Single origin

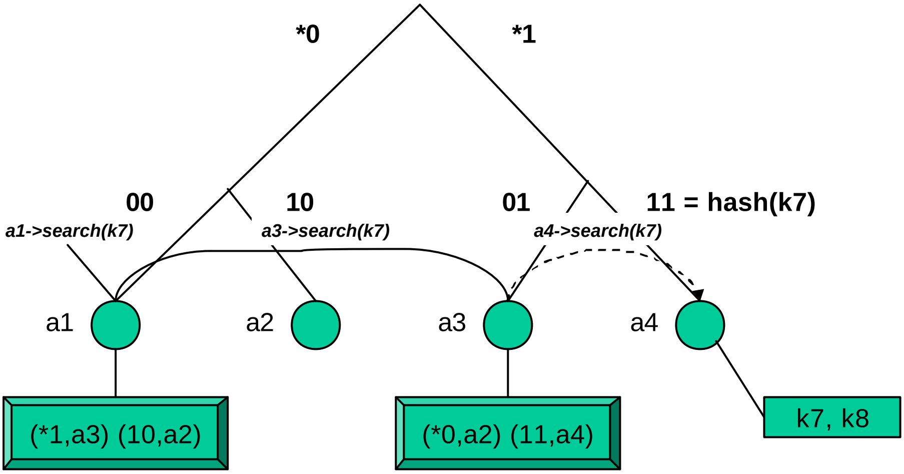
Tapestry

- Based on building distributed, **n-ary search trees**
- Each peer is assigned to a leaf of the search tree
- Each peer stores references for the other branches in the tree for routing
- Search requests are either processed locally or forwarded to the peers on the alternative branches
- Each peer obtains an ID in the node ID space
- Each data object obtains a home peer based on a distributed algorithm applied to its ID

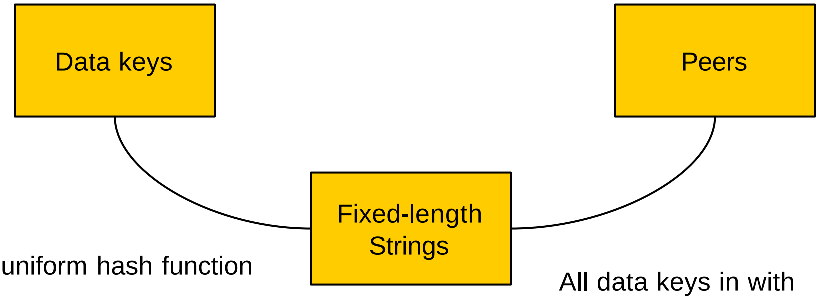
Tapestry

- Peers cache <OID, NID> between data object storage and home node to speed up access (compare to FreeNet)
- Replication is supported, each replica obtains an own ID extending the OID
- Access structure based on [Plaxton]

Tapestry Routing



Tapestry Data Identifiers

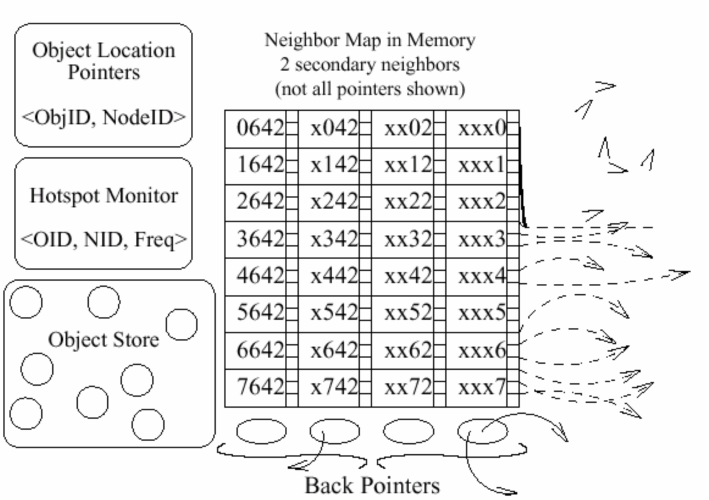


uniform hash function

All data keys in with
longest common suffix with peer ID,
deterministic distributed
algorithm to select unique

Remark: original Plaxton proposal
used global ordering

Data Structure at One Peer



Object Location Pointers

<ObjID, NodeID>

Hotspot Monitor

<OID, NID, Freq>

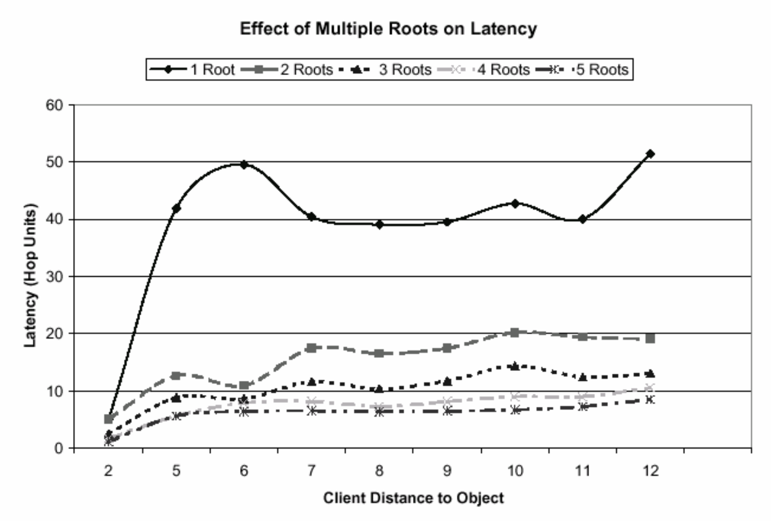
Object Store

Neighbor Map in Memory
2 secondary neighbors
(not all pointers shown)

0642	x042	xx02	xxx0
1642	x142	xx12	xxx1
2642	x242	xx22	xxx2
3642	x342	xx32	xxx3
4642	x442	xx42	xxx4
5642	x542	xx52	xxx5
6642	x642	xx62	xxx6
7642	x742	xx72	xxx7

Back Pointers

Effect of Multiple Home Nodes



Tapestry Discussion

- Search types
  - Equality searches
- Scalability
  - Search and update O(Log n)
  - Node join operation is scalable
- Robustness
  - High when using replication
- Autonomy
  - Assignment of node IDs not clear
- Global Knowledge
  - Hashing of object Ids, replication scheme
  - Single origin

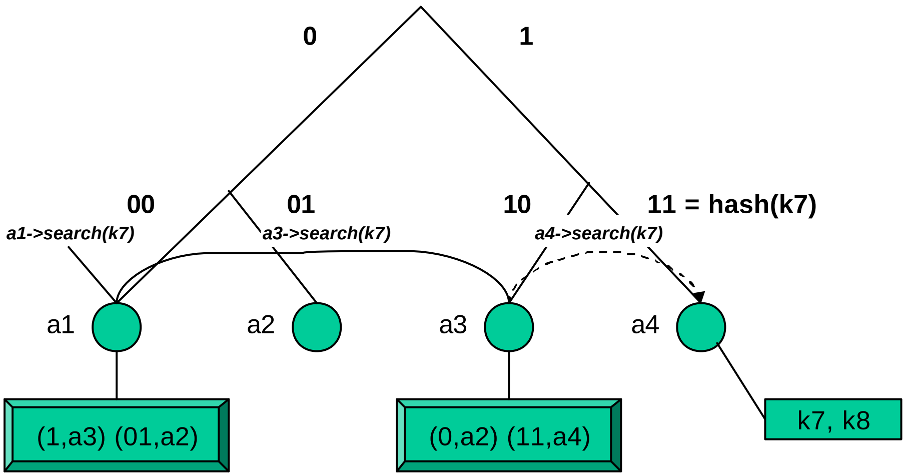
P-Grid

- Similar data organization as Tapestry, however node IDs of variable length
- Data objects stored at peer if node ID is prefix of data key
- Assignment of peers is performed by repeated mutual splitting of the search space among the peers
  - Tapestry-like data organization combined with CAN-like construction
- Splitting stops when abortion criteria is fulfilled
  - Maximal key length
  - Minimal number of known data items

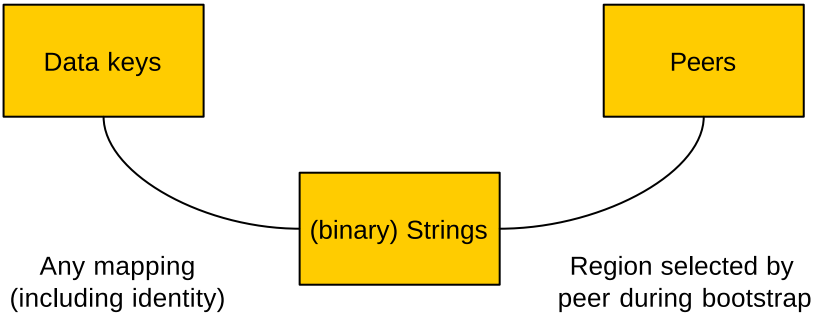
P-Grid

- Different P-Grids can merge during splitting (multiple origin possible, unlike CAN)
- Replication is obtained when multiple peers reside in same fragment of ID space

P-Grid Access Structure



P-Grid Data Identifiers



Data keys

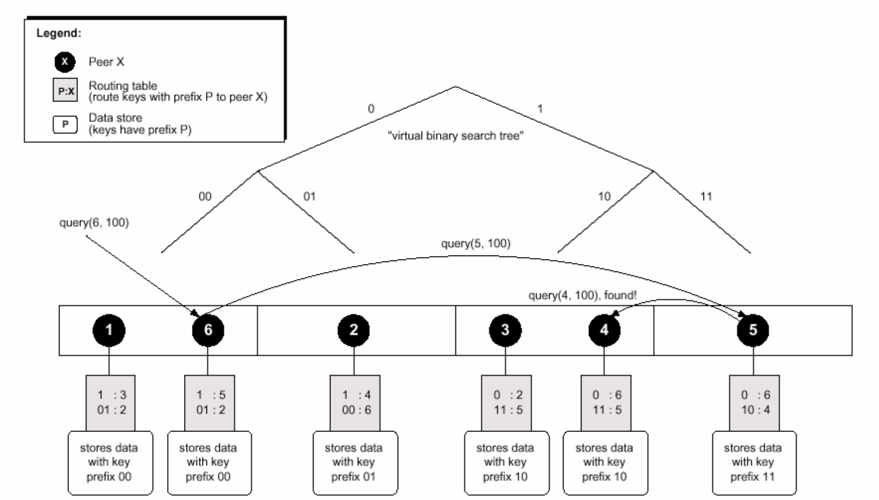
Peers

(binary) Strings

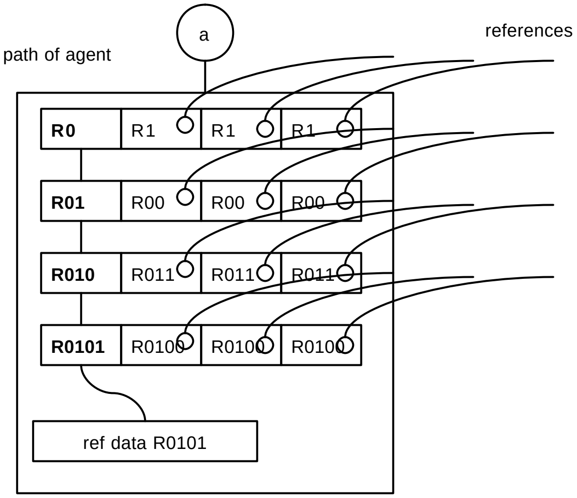
Any mapping
(including identity)

Region selected by
peer during bootstrap

P-Grid Search Structure Refinement

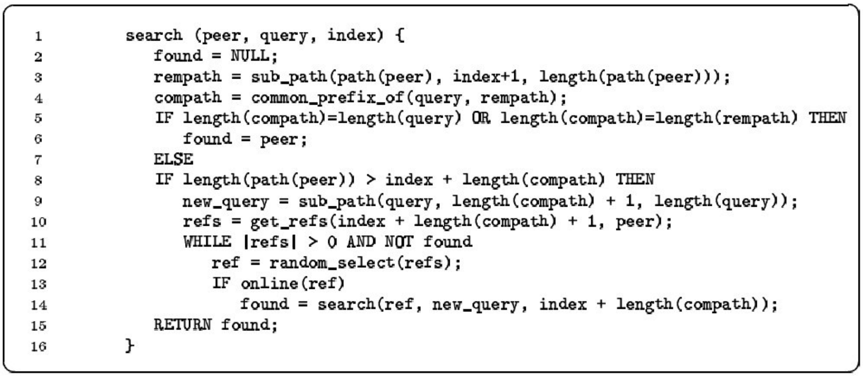

Legend:
X Peer X
P:X Routing table
(route keys with prefix P to peer X)
P Data store
(keys have prefix P)
"virtual binary search tree"
0
1
00
01
10
11
query(6, 100)
query(5, 100)
query(4, 100), found!
1
6
2
3
4
5
1 :3
01 : 2
1 :5
01 : 2
1 :4
00 : 6
0 :2
11 : 5
0 :6
11 : 5
0 :6
10 : 4
stores data with key prefix 00
stores data with key prefix 00
stores data with key prefix 01
stores data with key prefix 10
stores data with key prefix 10
stores data with key prefix 11

Data Structure of a Peer



P-Grid Search Algorithm

- get_refs(i, p) returns references peer p stores at path level i
- online(p) is true if the peer can be reached



```
search (peer, query, index) {
   found = NULL;
   rempath = sub_path(path(peer), index+1, length(path(peer)));
   compath = common_prefix_of(query, rempath);
   IF length(compath)=length(query) OR length(compath)=length(rempath) THEN
      found = peer;
   ELSE
   IF length(path(peer)) > index + length(compath) THEN
      new_query = sub_path(query, length(compath) + 1, length(query));
      refs = get_refs(index + length(compath) + 1, peer);
      WHILE |refs| > 0 AND NOT found
         ref = random_select(refs);
         IF online(ref)
            found = search(ref, new_query, index + length(compath));
   RETURN found;
}
```

P-Grid Construction Algorithm (Bootstrap)

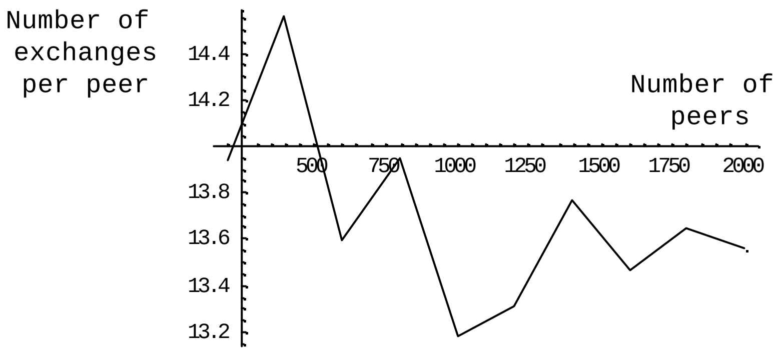
- When agents meet (randomly)
  - Compare the current search paths p and q
- Case 1: p and q are the same
  - If the abort criteria is not satisfied extend the paths and split search space, i.e. to p0 and q1
- Case 2: p is a subpath of q, i.e. q = p0…
  - If the abort criteria is not satisfied extend p by the inverse, i.e. p1
- Case 3: only a common prefix exists
  - Forward to the referenced peer with longer path
  - Limit forwarding by *recmax*
- The agents remember each other and exchange in addition references at all levels
- No further extensions of the path are performed when a maximal keylength *k* is reached

Simulations

- Implementation in Mathematica
- Simulation parameters *(n, k, recmax, refmax)*
  - Agent population size *n*
  - Key length *k*
  - Recursion depth *recmax*
  - Multiple references *refmax*
- Determine number of meetings required to reach on average 99% of maximal path length
- 3 experiments

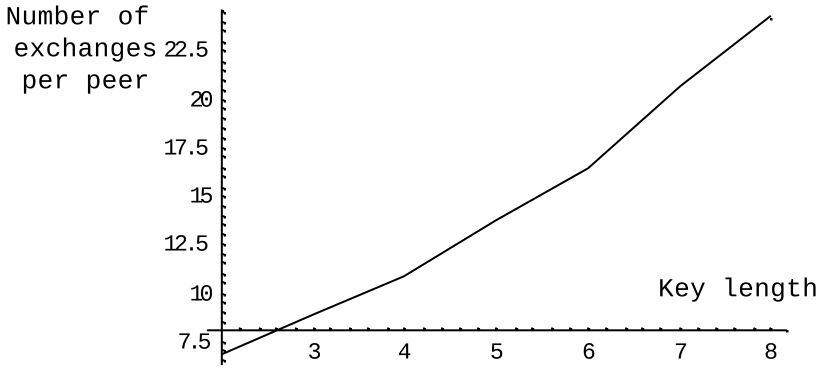
Dependency on Peer Population Size

- *(n = 200..2000, k = 6, recmax =2, refmax =1)*



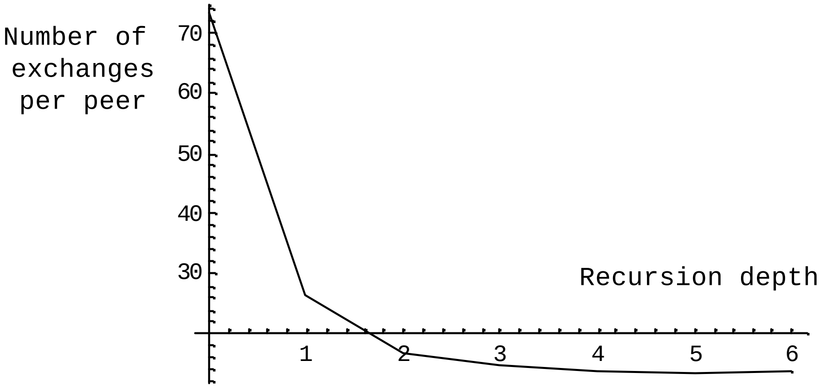
Dependency on Key Length

- *(n = 1000, k = 2..8, recmax =k/2, refmax =1)*
- linear



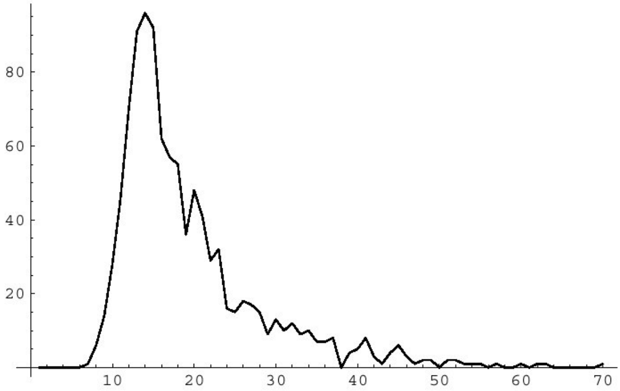
Dependency on Recursion Depth

- *(n = 500, k = 6, recmax =0..6, refmax =1)*
- k/2 sufficient



Replica Distribution

- *(n = 20000, k = 10, recmax =*, refmax =20)*



Properties of P-Grids

- Convergence ?
  - Does not depend on population size
  - Depends on key length linearly
  - Depends on recursion depth
- Distribution of replicas ?
  - Simulations indicate a "normal distribution"
  - Access paths to replicas are non-uniformly distributed
- Balanced trees ?
  - Simple argument (and simulations) show that this is very likely

P-Grid Variations

- No maximal key length
  - Tree depth follows data density: unbalanced trees
  - Search cost still O(Log(n)) in number of messages !
- Updates
  - Peers maintain list of replicas
- Growing and shrinking of keys
  - problem: integrity of referenced peers
- Joining and leaving P-Grids
  - Joining: apply bootstrap algorithm
  - Leaving: invalidation of references
- Overlaying multiple P-Grids

P-Grid Flexibility

- The P-Grid approach represents rather a framework than a single solution
  - Many parameters are left open
  - leave room for optimization
  - e.g., taking into account
    - access probability
    - existing data distribution
    - reachability and access cost

P-Grid Discussion

- Search types
  - Prefix/Range searches
- Scalability
  - Search and update O(Log n)
  - Construction: bootstrap is efficient
- Robustness
  - High due to replication
- Autonomy
  - Storage and routing are connected
  - Free choice whether a specific path is supported
- Global Knowledge
  - Mapping of data keys on binary keys

Summary and Comparison of Approaches

	Paradigm	Search Type	Search Cost (messages)	Fixed Data Assignment	Common Network Origin
Gnutella	Breadth-first search on graph	String comparison	$2*\sum_{i=0}^{TTL} C*(C-1)^i$	no	no
Freenet	Depth-first search on graph	Equality	O(Log n) ?	no	no
Chord	Implicit binary search trees	Equality	O(Log n)	yes	yes
CAN	d-dimensional space	Equality	O(d n^(1/d))	no	yes
Tapestry	Prefix trees	Equality	O(Log n)	yes	no ?
P-Grid	Binary prefix trees	Prefix	O(Log n)	no	no

Related Approaches from Distributed DB

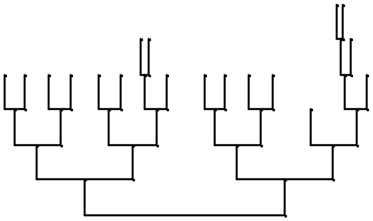
- Litwin's LH*
  - distribution of database fragments over networks of workstations
- Widmayer
  - distribution of one binary tree (can not be balanced)
- Mariposa
  - Distribution of RDB without central coordination
  - market mechanism for resource allocation
  - No indexing
- Db-Tree, Yokota
  - B-Trees with scalable replication to speed up parallel data access

P-Grid Applications

- Gridella
- Trust management

Gridella: An Enhanced Gnutella System

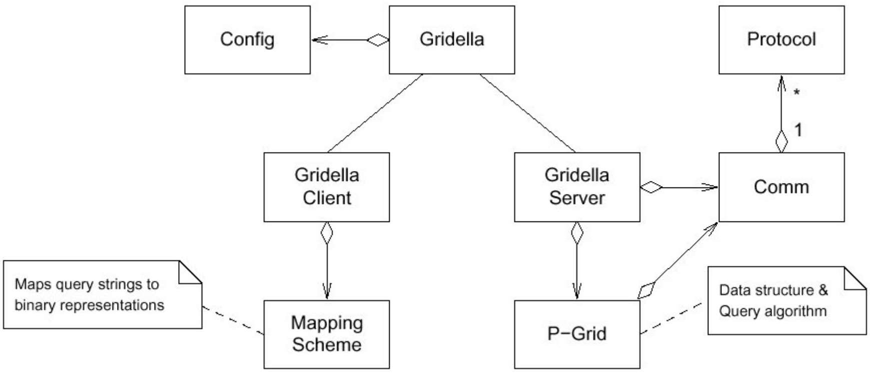
- Currently under implementation
- Uses Gnutella protocol for compatibility (other protocols can be plugged in)
- Controls routing of search requests using P-Grid
- Non-uniform distribution of search keys
  - Build statistics
  - Compute a global, prefix-preserving hash function



Computing the Required Resources

- Assume
  - 10^7 searchable keys (substrings of filenames)
  - 10 Bytes for storing a peer address
  - 10^5 Bytes per peer provided for indexing
  - 30 % online probability
  - 99 % desired answer reliability
- Then
  - Approx. 20,000 peers can be supported
  - refmax = 20 is sufficient

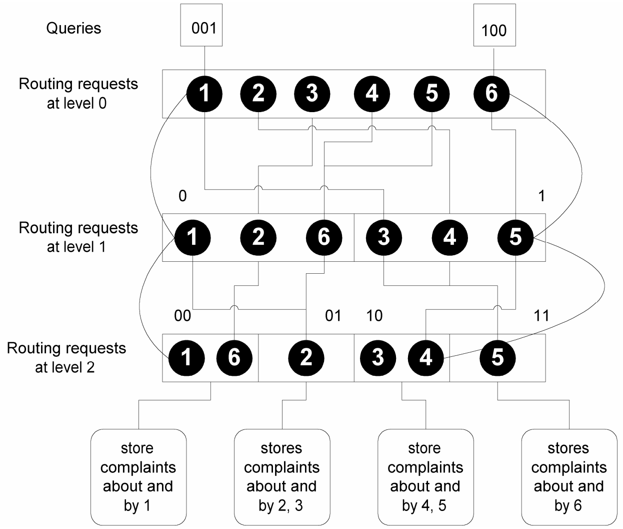
Gridella Architecture



Trust Management based on Reputation

- Approach
  - Record complaints by peers
  - Build a decentralized data warehouse based on P-Grids
  - Each peer computes average number of complaints
  - It retrieves from the data warehouse all complaints on (and by) a peer
  - It assesses also the trustworthiness of the peers reporting theses numbers
  - It decides upon the formula
- Result
  - Even with a large fraction of cheaters (25% are cheating 25% of the time) they can be reliably recognized

Using P-Grid to store Trust Data



Queries

Routing requests at level 0

Routing requests at level 1

Routing requests at level 2

store complaints about and by 1

stores complaints about and by 2, 3

store complaints about and by 4, 5

stores complaints about and by 6

Research Issues

- P2P for reliable E-Commerce
  - dynamic business models
  - trust establishment
  - peer-to-peer transactions
  - Decision making
- Quality of service
  - improved fault tolerance
  - quality guarantees
- Richer data model
  - relational XML
  - meta-data model
  - improved search
- Multimedia
- Message-based applications
  - scalability
  - improved search capabilities
- Mobility

References - 1

[Aberer01a] Karl Aberer. *P-Grid:A self-organizing access structure for P2P information systems*. Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS 2001), 2001. http://lsirwww.epfl.ch/publications/tr/TR2001-016.pdf

[Aberer01b] Karl Aberer, Zoran Despotovic. *Managing Trust in a Peer-2-Peer Information System*. To appear in the Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2001) 2001. http://lsirwww.epfl.ch/publications/tr/TR2001-029.pdf

[Aberer01c] Karl Aberer, Manfred Hauswirth, Magdalena Punceva, Roman Schmidt. *Improving Data Access in P2P Systems*. IEEE Internet Computing, vol. 6 nr. 1, 2002.

[Adar00] Eytan Adar and Bernardo A. Huberman. *Free Riding on Gnutella*. Technical report. Xerox PARC. September 9, 2000. http://www.parc.xerox.com/istl/groups/iea/papers/gnutella/Gnutella.pdf

[Clarke01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. LLNCS 2009. Springer Verlag 2001. http://www.freenetproject.org/index.php?page=icsi-revised

References - 2

[Clip01] Clip2. *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*. June 15, 2001. http://www.clip2.com/GnutellaProtocol04.pdf

[Dabek01] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, Hari Balakrishnan. *Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service*. Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), 2001. http://www.pdos.lcs.mit.edu/papers/chord:hotos01/hotos8.pdf

[Drscholl01] Dr. Scholl. *Napster protocol specification*. April 7, 2001. http://opennap.sourceforge.net/napster.txt

[Eugster00] Patrick Th. Eugster, Rachid Guerraoui, and Joe Sventek. *Type-Based Publish/Subscribe*. Technical report TR-DSC-2000-29. Swiss Federal Institute of Technology, Lausanne (EPFL), June 2000. http://dscwww.epfl.ch/EN/publications/documents/tr00_029.ps.

[Freenet01] Freenet Project. *Protocol Specs*. 2001. http://www.freenetproject.org/doc/book.html

[Gnutella01] *Gnutella homepage*, 2001. http://gnutella.wego.com/

[Gong01] Li Gong. *JXTA: A Network Programming Environment*. IEEE Internet Computing, 5(3):88-95, May/June 2001. http://dlib.computer.org/ic/books/ic2001/pdf/w3088.pdf

References - 3

[Hubaux01] J.P. Hubaux, Th. Gross, J.-Y- Le Boudec M. Vetterli. *Towards self-organized mobile ad-hoc networks: the Terminodes project.* IEEE Communications Magazine, January 2001. http://www.terminodes.org/publications/commag01a.pdf

[Hong01] Theodore Hong. *Performance in Decentralized Filesharing Networks*. Presentation given by at the O'Reilly Peer-to-Peer Conference, San Francisco. February 14-16, 2001. http://www.freenetproject.org/p2p-theo.ppt

[Jovanovic01] M.A. Jovanovic, F.S. Annexstein, and K.A.Berman. *Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella*. University of Cincinnati, Laboratory for Networks and Applied Graph Theory, 2001. http://www.ececs.uc.edu/~mjovanov/Research/paper.ps

[Jxta01] *Project JXTA homepage*, 2001. http://www.jxta.org/

[Kleinberg99] Jon Kleinberg. *The Small-World Phenomenon: An Algorithmic Perspective*. Technical report 99-1776. Cornell Computer Science, October 1999. http://www.cs.cornell.edu/home/kleinber/swn.pdf

[Langley01] Adam Langley. *The Freenet Protocol*. 2001. http://www.freenetproject.org/index.php?page=protocol

References - 4

[Litwin97] Witold Litwin, Marie-Anne Neimat. *LH*s: A High-Availability and High-Security Scalable Distributed Data Structure*. Proceedings of Research Issues in Data Engineering (RIDE), 1997.

[Napster01] *Napster homepage*, 2001. http://www.napster.com/

[Oram01] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, March 2001. http://www.oreilly.com/catalog/peertopeer/.

[Plaxton97] C. Greg Plaxton, Rajmohan Rajaraman, Andréa W. Richa: *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*. Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures, pages 311-20, 1997. http://www.cs.utexas.edu/users/plaxton/ps/1997/spaa.ps

[Ratnasamy01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. *A Scalable Content-Addressable Network*. Proceedings of the ACM SIGCOMM, 2001.

[Rosenblum97] D.S. Rosenblum and A.L. Wolf. *A design framework for Internet-scale event observation and notification*. Proceedings of the ESEC/FSE '97. LLNCS 1301, pages 344-60. Springer Verlag, Berlin, September 1997. ftp://ftp.cs.colorado.edu/users/alw/doc/papers/esec97b.pdf

References - 5

[Sripanidkulchai01] Kunwadee Sripanidkulchai. *The popularity of Gnutella queries and its implications on scalability*. February 2001. http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

[Stoica 00] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*. Proceedings of the ACM SIGCOMM, 2001.

[Stonebraker96] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, Andrew Yu. *Mariposa: A Wide-Area Distributed Database System*. VLDB Journal 5(1): 48-63, 1996. http://epoch.CS.Berkeley.EDU:8000/personal/aoki/papers/vldbj96.pdf

[Vingralek98] Radek Vingralek, Yuri Breitbart, Gerhard Weikum. *Snowball: Scalable Storage on Networks of Workstations with Balanced Load.* Distributed and Parallel Databases 6(2): 117-156, 1998.

[Yokota99] Haruo Yokota, Yasuhiko Kanemasa, Jun Miyazaki. *Fat-Btree: An Update-Conscious Parallel Directory Structure*. Proceedings of the 16th International Conference on Data Engineering (ICDE), pages 448-457, 1999.

Appendix - Protocols

- Napster
- Gnutella
- Freenet

Napster Protocol

Napster: The Protocol [Drscholl01]

- The protocol was never published openly and is rather complex and inconsistent
- OpenNap have reverse engineered the protocol and published their findings
- TCP is used for C/S communication
- Messages to/from the server have the following format:

	length	type	data
Byte offset	**0 1**	**2 3**	**4 ..... n**

  - length specifies the length of the data portion
  - type defines the message type
  - data: the transferred data
    - plain ASCII, in many cases enclosed in double quotes (e.g., filenames such as “song.mp3” or client ids such as “nap v0.8”

Napster: Sample Messages - 1

Type	C/S	Description	Format
0	S	Error message	<message>
2	C	Login	<nick><pwd><port><client info><link type>
3	S	Login ack	<user’s email>
5	S	Auto-upgrade	<new version><http-hostname:filename>
6	C	New user login	<nick><pwd><port><client info><speed> <email address>
100	C	Client notification of shared file	“<filename>”<md5><size><bitrate> <frequency><time>
200	C	Search request	[FILENAME CONTAINS “artist name”] MAX_RESULTS <max> [FILENAME CONTAINS <song] [LINESPEED <comp> <link type>] [BITRATE <comp> “bit rate”] [FREQ <comp> “freq”] [WMA-FILE] [LOCAL_ONLY]
201	S	Search response	“<filename>”<md5><size><bit rate> <frequency><length><nick><ip address>
202	S	End of search response	(empty)

Napster: Sample Messages - 2

Type	C/S	Description	Format
203	C	Download request	<nick> "<filename>"
204	S	Download ack	<nick><ip><port> "<filename>" <md5> <linespeed>
206	S	Peer to download not available	<nick> "<filename>"
209	S	Hotlist user signed on	<user><speed>
211	C	Browse a user's files	<nick>
212	S	Browse response	<nick> "<filename>"<md5><size> <bit rate><frequency><time>
213	S	End of browse list	<nick>[<ip address>]
500	C	Push file to me (firewall problem)	<nick> "<filename>"
501	S	Push ack (to other client)	<nick><ip address><port> "<filename>" <md5><speed>

Napster: Client-Client Communication - 1

- Normal download (*A* downloads from *B*):
  - *A* connects to *B*'s IP address/port as specified in the 204 message returned by the server (response to 203)
  - *B* sends the ASCII character "1"
  - *A* sends the string "GET"
  - *A* sends <mynick> "<filename>" <offset>
  - *B* returns the file size (not terminated by any special character!) or an error message such as "FILE NOT SHARED"
  - *A* notifies the server that the download is ongoing via a 218 message; likewise *B* informs the server with a 220 message
  - Upon successful completion *A* notifies the server with a 219 message; likewise *B* informs the server with a 221 message

Napster: Client-Client Communication - 2

- Firewalled download (*A* wants to download from *B* who is behind a firewall):
  - *A* sends a 500 message to the server which in turn sends a 501 message (holding *A*'s IP address and data port) to *B*
  - *B* connects *A* according to the 501 message
  - *A* sends the ASCII character "1"
  - *B* sends the string "SEND"
  - *B* sends <mynick> "<filename>" <size>
  - *A* returns the byte offset at which the transfer should start (plain ASCII characters) or an error message such as "INVALID REQUEST"
  - *A* notifies the server that the download is ongoing via a 218 message; likewise *B* informs the server with a 220 message
  - Upon successful completion *A* notifies the server with a 219 message; likewise *B* informs the server with a 221 message

Gnutella Protocol

Gnutella: Descriptors

- Meeting
  - GNUTELLA CONNECT/0.4\n\n
  - GNUTELLA OK\n\n
- “Descriptor header” (general packet header)

	Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length
Byte offset	0 15	16	17	18	19 22

  - Descriptor ID: 16 byte unique id
  - Payload descriptor: packet type (e.g., 0x00 = Ping)
  - TTL: the number of times the descriptor will be forwarded
  - Hops: TTL(0) = TTL(i) + Hops(i)
  - Payload length: the length of the descriptor immediately following this header

Gnutella: Ping/Pong Descriptors

- Ping (0x00): Descriptor header with payload 0x00
- Pong (0x01):

	Port	IP address	Number of files shared	Number of kilobytes shared
Byte offset	0 1	2 5	6 9	10 13

  - Port: on which the responding host can accept connections
  - IP address: of the responding host
  - Number of files shared
  - Number of kilobytes shared

Gnutella: Query Descriptor

- Query (0x80):

	Minimum speed	Search criteria
Byte offset	0 1	2 ....

  - Minimum speed: the minimum network bandwidth of the servent (in kb/s) that should respond to this query
  - Search criteria: a null (i.e., 0x00) terminated string; the maximum length of this string is bounded by the “Payload length” field of the descriptor header.

Gnutella: QueryHit Descriptor (0x81)

	Number of hits	Port	IP address	Speed	Result set	Servent identifier
Byte offset	0	1 2	3 6	7 10	11	n n+16

- Number of hits: in the result set
- Port: on which the responding host can accept connections
- IP address: of the responding host
- Speed: of the responding host (in kb/s)
- Servent identifier: 16-byte string uniquely identifying the servent
- Result set (*number of hits* records)

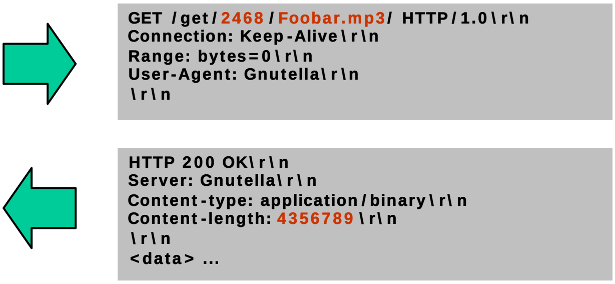
	File index	File size	File name
Byte offset	0 3	4 7	8 ......

  - File index: a number assigned by the responding host to uniquely identify the file matching the corresponding query
  - File size: size of the file (in bytes)
  - File name: double null (0x0000) terminated name of the file

Gnutella: File Downloads

- Out of band via simplified HTTP
- Connect to IP/address given in QueryHit
- Example:

2468	4356789	Foobar.mp3\0x00\0x00
File index	File size	File name



```
GET /get/2468/Foobar.mp3/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0\r\n
User-Agent: Gnutella\r\n
\r\n
```

```
HTTP 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 4356789\r\n
\r\n
<data> ...
```

Gnutella: Handling Firewalls - Push Descriptor

- If a host cannot be contacted directly (firewall)
- The servent receiving a Push descriptor (0x40) initiates the file transfer (outgoing connection)

	Servent identifier	File index	IP address	Port
Byte offset	0 15	16 19	20 23	24 25

  - Servent identifier: 16-byte string uniquely identifying the servent who is requested to push the file
  - File index: uniquely identifying the file to be pushed
  - IP address: of the host to which the file should be pushed
  - Port: to which the file should be pushed
- Does not work if both servents are behind firewalls

Gnutella: Push

- Servent *A* receives a QueryHit from servent *B* who is behind a firewall and cannot accept incoming connections other than on its Gnutella port
- *A* sends a Push descriptor to *B*

Servent identifier	File index	IP address	Port

- *B* opens a connection to the IP address/port given in the Push descriptor and sends:

```
GIV <File index>:<Servent identifier>/<File name>\n\n
```

- Upon receiving the GIV request *A* initiates a normal download via this connection

```
GET /get/<File index>/<File name>/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0\r\n
User-Agent: Gnutella\r\n
\r\n
```

Freenet Protocol

Freenet: Inter-node Protocol

```
DataReply
UniqueID=C24354BF458EBE1448CFDA
Depth=9
HopsToLive=22
Source=tcp/123.156.205.23:2386
DataLength=4711
KeepAlive=true
Data
The minstrel in the gallery looked down upon the smiling faces.
He met the gazes -- observed the spaces between the old men's cackle.
He brewed a song of love and hatred -- oblique suggestions -- and he waited.
He polarized the pumpkin-eaters -- static-humming
   panel-beaters -- freshly day-glow'd factory cheaters
   (salaried and collar-scrubbing).
....
```

Message types:

- HandshakeRequest → HandshakeReply (connection establishment)
- DataRequest (+ SearchKey field) → DataReply (+ Data), TimeOut
- InsertRequest (+ SearchKey field) → InsertReply → InsertData (+ Data)
- QueryRestarted

Freenet Client Protocol (FCP)

- Between client and the local node (to support client developers)
- Message types:
  - ClientHello
    - NodeHello
  - ClientGet
    - URIError, Restarted, DataNotFound, RouteNotFound, DataFound, DataChunk
  - ClientPut
    - URIError, Restarted, RouteNotFound, KeyCollision
  - GenerateCHK , GenerateSVKPair
    - Success

```
ClientPut
HopsToLive=22
URI=freenet:KSK@text/books/1984.html
DataLength=4711
Data
The minstrel in the gallery looked down upon the smiling faces.
He met the gazes -- observed the spaces between the old men's cackle.
He brewed a song of love and hatred -- oblique suggestions -- and he waited.
He polarized the pumpkin-eaters -- static-humming
   panel-beaters -- freshly day-glow'd factory cheaters
   (salaried and collar-scrubbing).
....
```