Network Services

Dr. Manfred Hauswirth
Laboratoire de Systèmes d'Information Répartis
École Polytechnique Fédérale de Lausanne (EPFL)
http://lsirpeople.epfl.ch/hauswirth/

Oct 24, 2002 – Lecture 4

Overview

- What is P2P?
  - P2P properties
  - Types of P2P Systems
  - A historical view
  - Related approaches
- State-of the-art systems
  - Napster, FastTrack, Gnutella, Freenet
- P2P Data Management
  - Gnutella, Freenet, P-Grid
  - Applications of P-Grid

(Non-technical) Motivations for using P2P

- Exploiting free resources
  - Famous example: SETI@Home
- Sharing costs
  - New economic models (also of interest to large companies)
- Autonomy
  - No dependence on central server
- Self-maintenance
- Anonymity
  - No registration at central server
  - Though non-negligible security problems (open environment)
  - Legal protection, but …

P2P is not new !

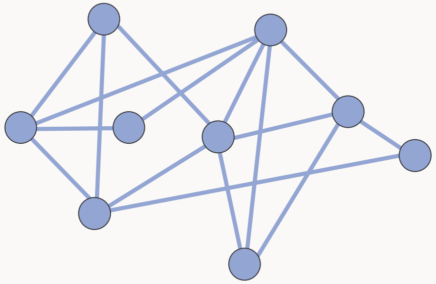
- The original Internet was designed as a P2P system
  - any 2 computers could send packets to each other
    - no firewalls / no network address translation
    - no asymmetric connections (V.90, ADSL, cable, etc.)
  - the back-then “killer apps” FTP and telnet are C/S but anyone could telnet/FTP anyone 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 new ?

- 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

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



Types of P2P Systems

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

P2P Cooperation Models

- Centralized model (Napster)
  - global index held by a central authority (single point of failure)
  - direct contact between requestors and providers
- Decentralized model (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 (FastTrack, DNS)
  - introduction of "super-peers"
  - mix of centralized and decentralized model

So what's new?

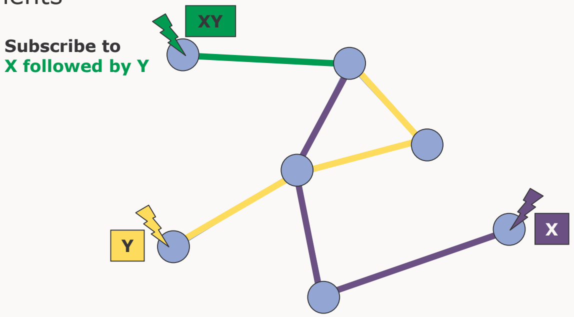
- The original Internet was designed as a P2P system
  - any 2 computers could send packets to each other
  - back-then "killer apps" FTP and telnet are C/S but anyone could telnet/FTP anyone else
  - servers acted as clients and vice versa
  - cooperation was a central goal and "value": no spam or exhaustive bandwidth consumption
- "Old-fashioned P2P": News, DNS
- P2P: a renaissance of the original Internet model

Related Approaches

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

Event-based 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

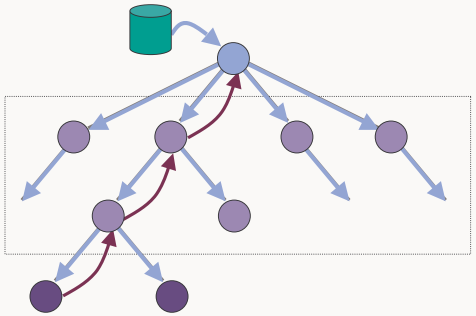


Event-based vs. P2P

- 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

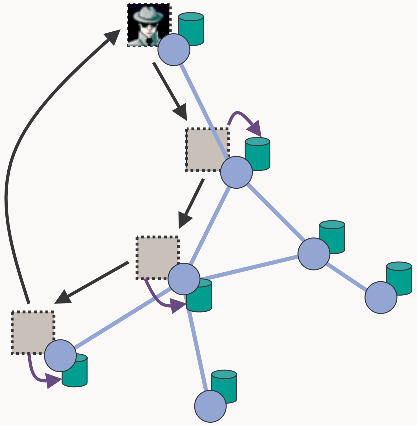


Push Systems vs. P2P

- 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

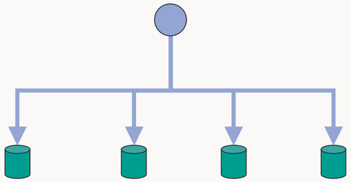


Mobile Agents vs. P2P

- 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)
  - protect node from malicious mobile agents and vice versa
- In many domains P2P systems can take over
  - more apt for distributed data management
  - less requirements (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 DBs vs. P2P

- 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 99]
- Approaches in distributed DB that address autonomy (and scalability)
  - Mariposa: distributed relational DBMS based on an underlying economic model [Stonebraker96]

Usage patterns to position P2P

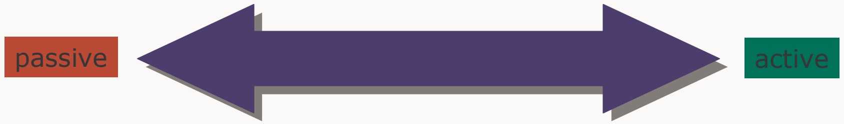
Discovering information is the predominant problem

- Occasional discovery: search engines P2P, MA
  - ad hoc requests, irregular
  - E.g., new town — where is the next car rental?
- Notification: event-based systems push
  - notification for (correlated) events (event patterns)
  - E.g., notify me when my stocks drop below a threshold
- Regular discovery: P2P systems search engines, MA
  - find certain type of information on a regular basis
  - E.g., search for MP3 files of Jethro Tull regularly
- Continuous information feed: push systems event-based
  - subscription to a certain information type
  - E.g., sports channel, updates are sent as soon as available

The interaction spectrum

Event-based systems
Push systems

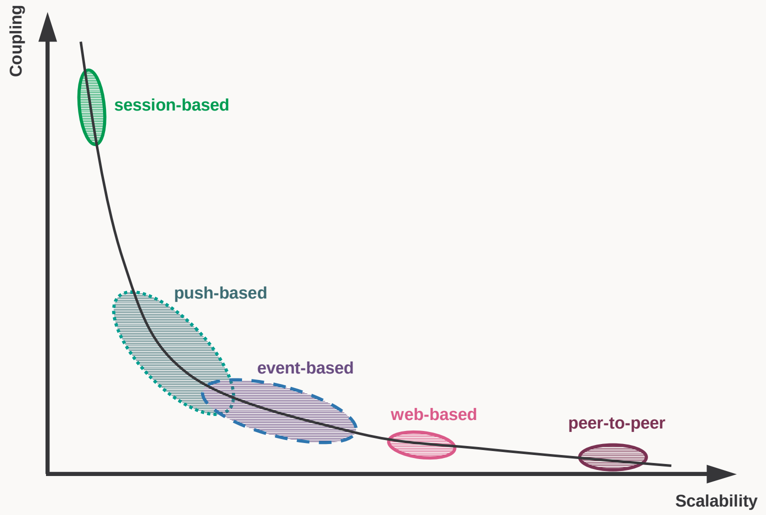
Mobile agents
Peer-to-peer systems



P2P vs. C/S and web-based

	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)

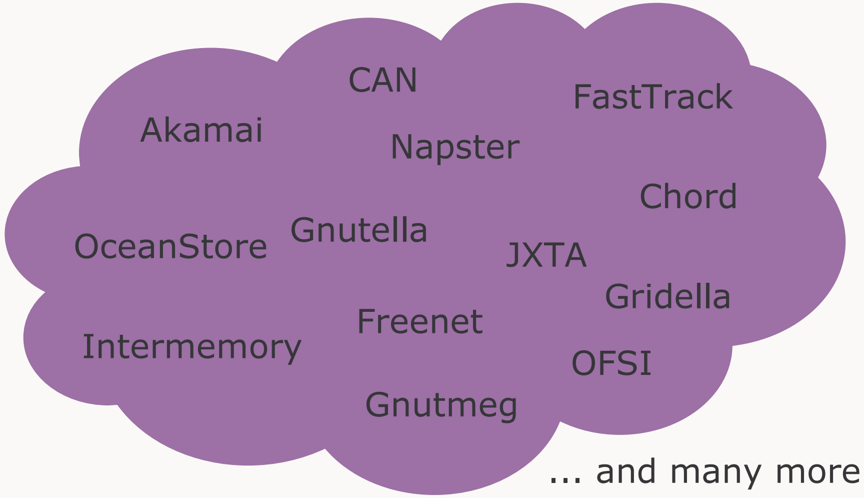
Coupling vs. scalability



P2P design requirements

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

The P2P cloud



... and many more ...

State-of-the-Art Systems

Napster

FastTrack

Gnutella

Freenet

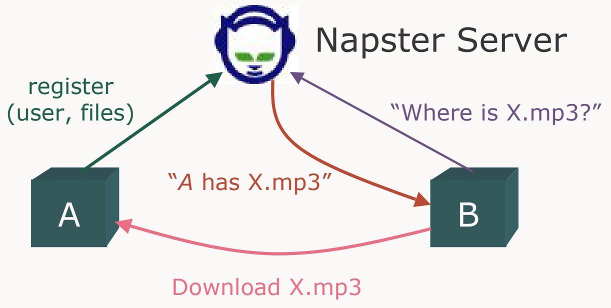
Napster: A brief History

- **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
- **Now:** Basically out of business

Napster's 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's protocol [Drscholl01]

- The protocol was never published openly and is rather complex and inconsistent
- OpenNap: reverse engineered the protocol and published it
- 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"

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)

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>

C/C 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

C/C 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

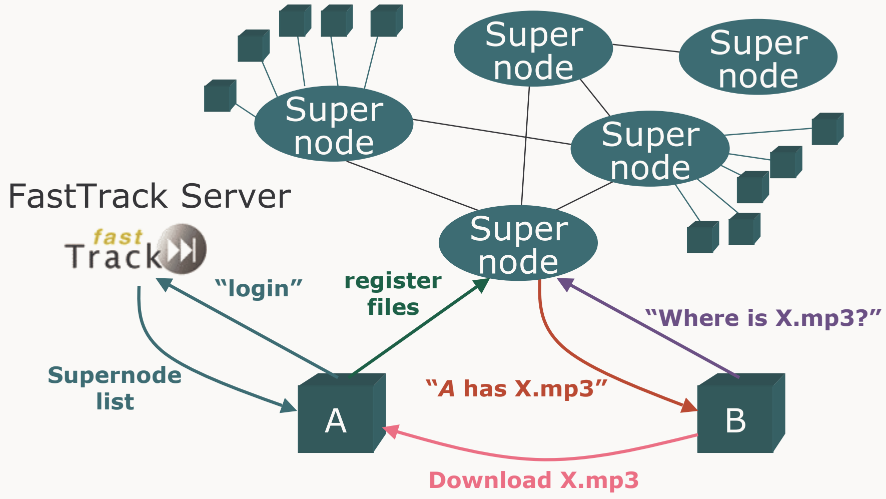
Napster: "Further" Services

- 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

FastTrack: Communication model



FastTrack

- “Hierarchical Napster”
- Not truly decentralized
- Protocol is not open (not published)
- Used by several P2P applications: KaZaA, Grokster, etc.
- Peers can be elected to become Supernodes (possibility to opt out)
- SmartStream: fail-over system for downloads
- FastStream: parallel download from multiple peers

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

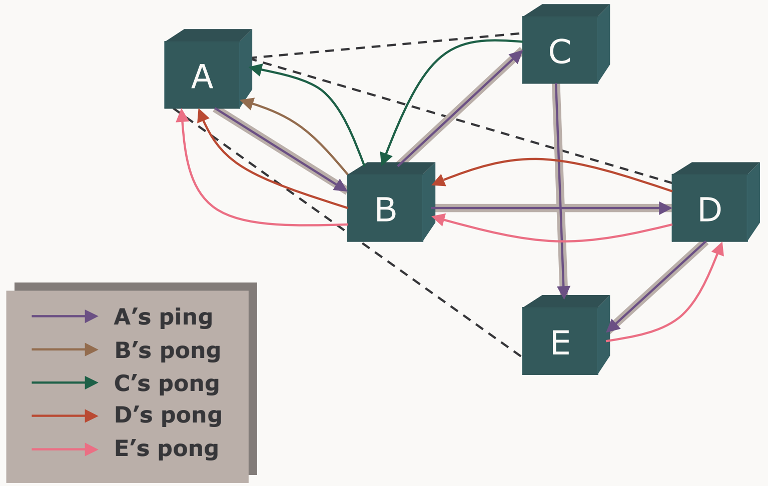
Gnutella’s architecture

- 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)



Protocol: 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

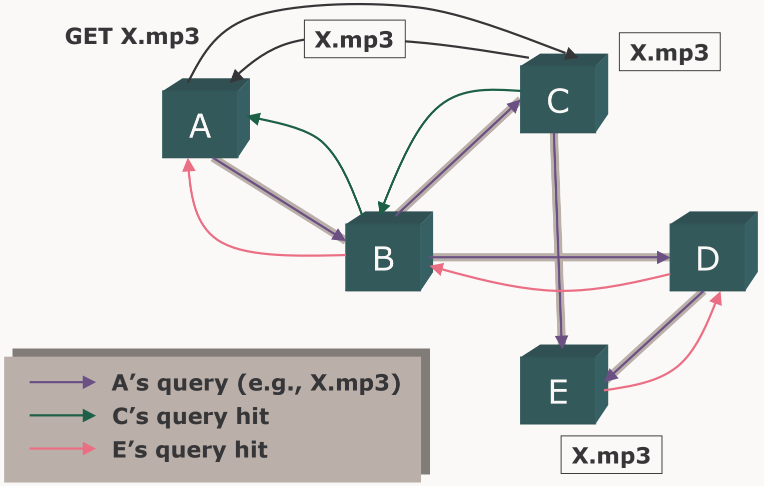
Ping/Pong descriptors

- 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: Searching (Query/QueryHit/GET)



Query descriptor

- Query (0x80):

Byte offset	Minimum speed	Search criteria
	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.

QueryHit Descriptor (0x81)

Byte offset	Number of hits	Port	IP address	Speed	Result set	Servent identifier
	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)

Byte offset	File index	File size	File name
	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

File downloads

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

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

```
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> ...
```

Handling firewalls: the Push descriptor

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

Byte offset	Servent identifier	File index	IP address	Port
	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
```

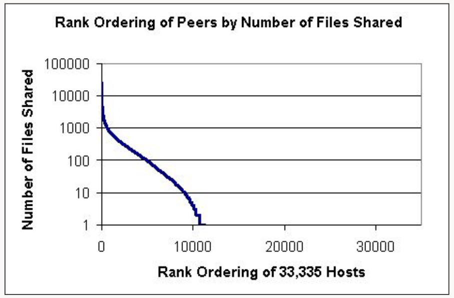
- Upon receiving the GIV request A initiates a normal download via this connection

```
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
```

Free-riding on Gnutella [Adar00]

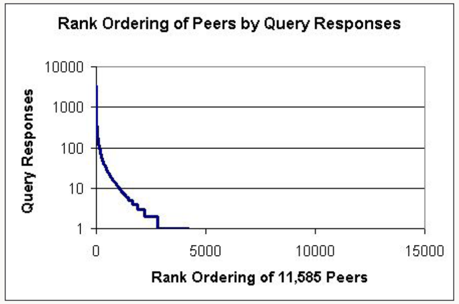
- 24 hour sampling period:
  - 70% of Gnutella users share no files
  - 50% of all responses are returned by top 1% of sharing hosts
- A social problem not a technical one
- Problems:
  - Degradation of system performance: collapse?
  - Increase of system vulnerability
  - "Centralized" ("backbone") 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 shared
  - 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

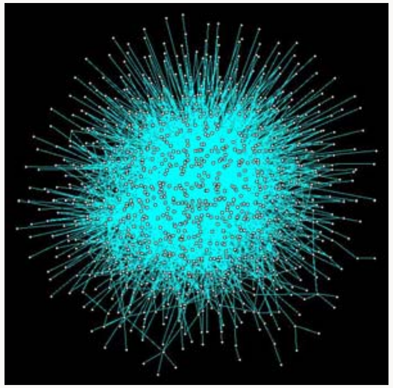
Free-riding statistics - 2 [Adar00]



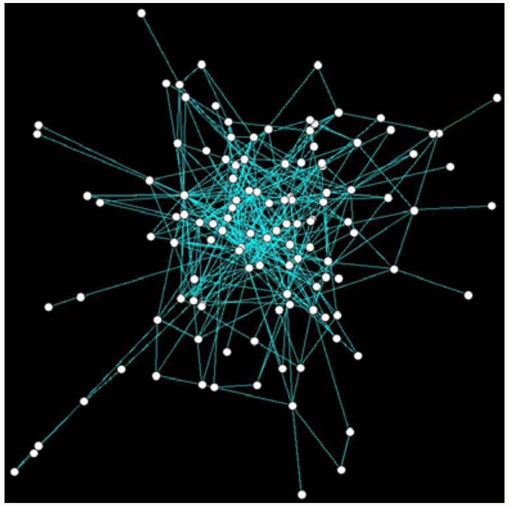
- H3: Many servents share files nobody downloads
- Of 11,585 sharing hosts:
  - Top 1% of sites provide nearly 47% of all answers
  - Top 25% of sites provide 98% of all answers
  - 7,349 (63%) never provide a query response

Topology of Gnutella [Jovanovic01]

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

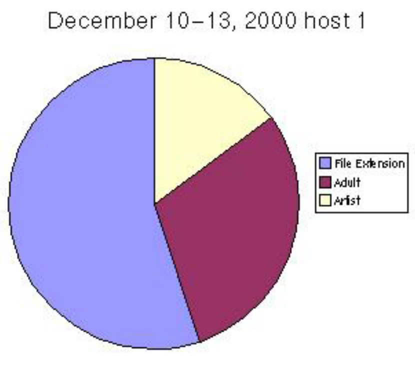


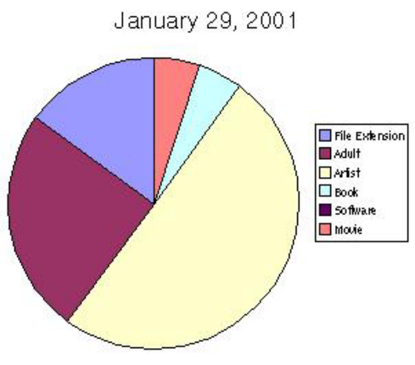
Gnutella backbone [Jovanovic01]



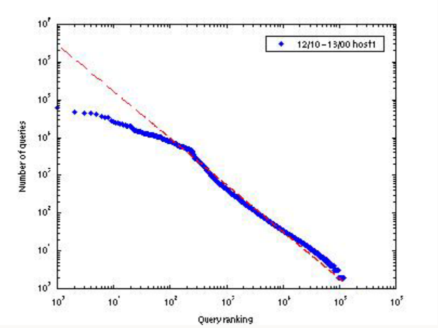
Categories of queries [Sripanidkulchai01]

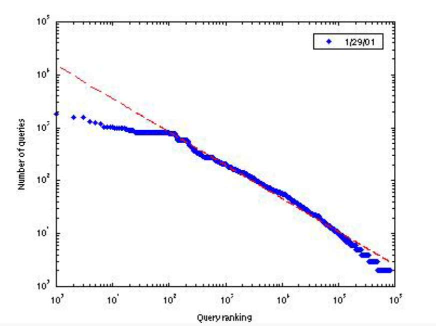
- Categorized top 20 queries





Popularity of queries [Sripanidkulchai01]

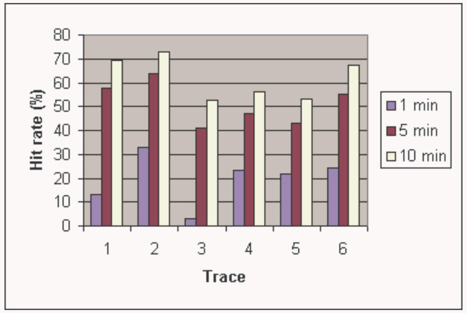


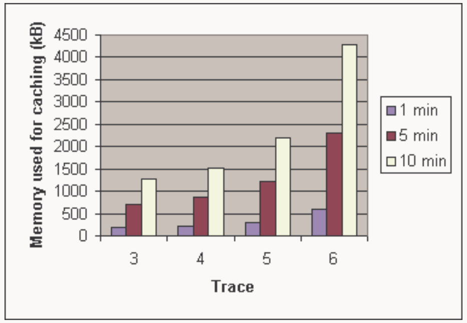


- Very popular documents are approximately equally popular
- Less popular documents follow a Zipf-like distribution (i.e., the probability of seeing a query for the **i$^{\text{th}}$** most popular query is proportional to **1/(i$^{\text{alpha}}$)**
- Access frequency of web documents also follows Zipf-like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

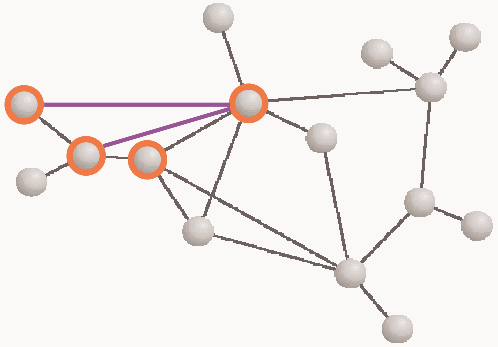
Gnutella: Summary

- 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: 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]

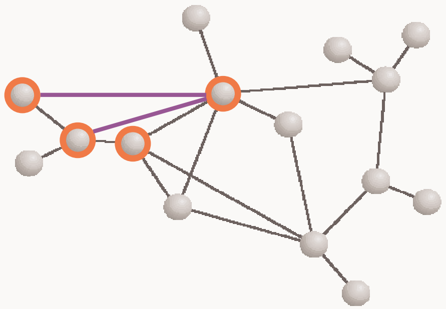


- Graph structure actively evolves over time
  - new links form between nodes
  - files migrate through the network

  ⇒ adaptive routing

Freenet: aspects of searching

- Steepest-ascent hill-climbing search
- Lazy replication (along the path)
- Hops-to-live limit of requests (can be decremented by every node)
- Pseudo-unique request identifiers to prevent loops
- Joining: out-of-band means
- Routing via key similarity measure based on lexicographic distance ("closeness")
- Quality of routing should improve over time:
  - node is listed under certain key in routing tables ⇒ gets more requests for similar keys ⇒ gets "experienced"
  - forwarding requests results in replicating results ⇒ node gets cluster of similar files (keys)



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
d0ui43203803ujoejqhh		tcp/128.121.89.12:9991

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: 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

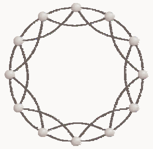
Freenet Client Protocol (FCP)

- 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

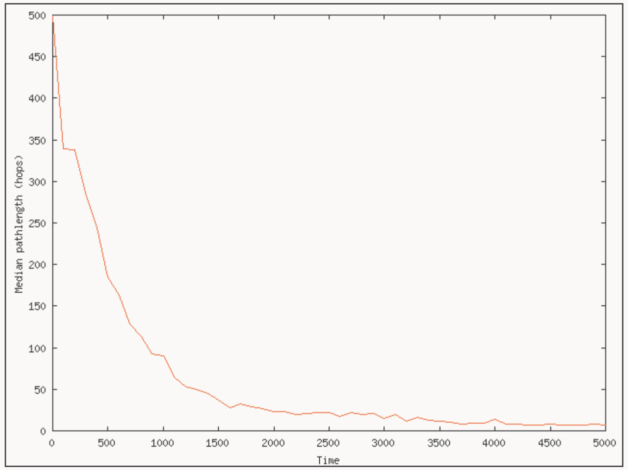
```
ClientPut
HopsToLive=22
URI=freenet:KSK@text/books/1984.html
DataLength=4711
Data
The minstrel in the gallery looked down upon the smiling faces.
He met the gazes -- observed the spaces between the old men's cackle.
He brewed a song of love and hatred -- oblique suggestions -- and he waited.
He polarized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

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)**



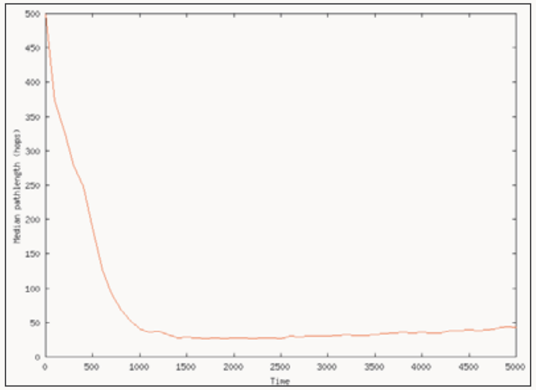
- **each time-step:**
  - **randomly query/insert**
  - **HopsToLive=20**
- **every 100 time-steps: 300 requests (HTL=500) from random nodes and measure actual path length (failure=500).**



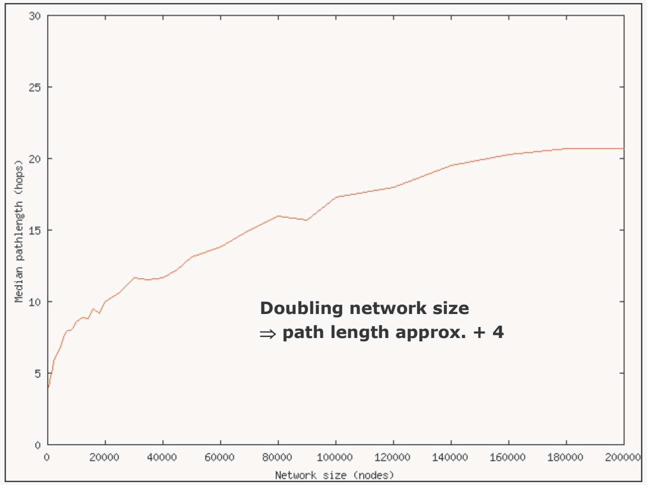
median path length 500 → 6

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

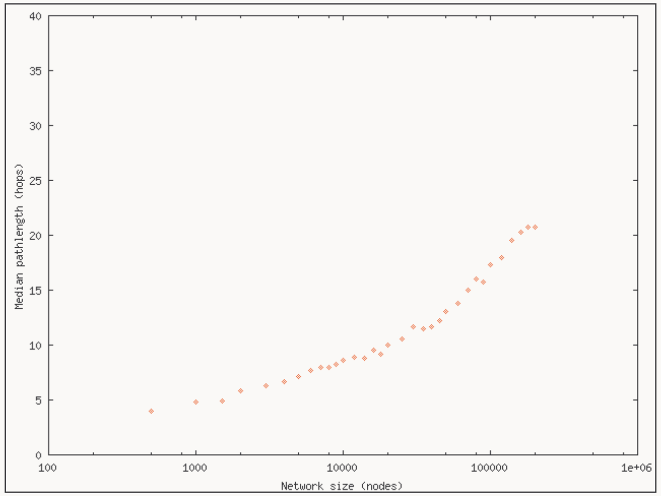


Freenet: scalability [Hong01]



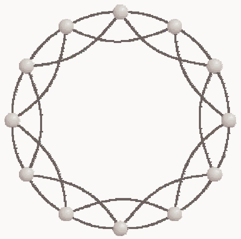
Doubling network size ⇒ path length approx. + 4

Freenet: scalability (log Scale) [Hong01]

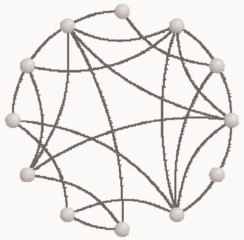


Why does it work? It's a small World! [Hong01]

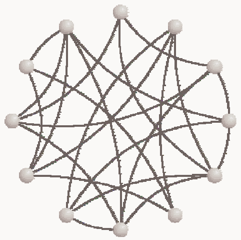
- Milgram: 42 out of 160 letters from Oregon to Boston (~ 6 hops)
- Watts: between order and randomness
  - short-distance clustering + long-distance shortcuts



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



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



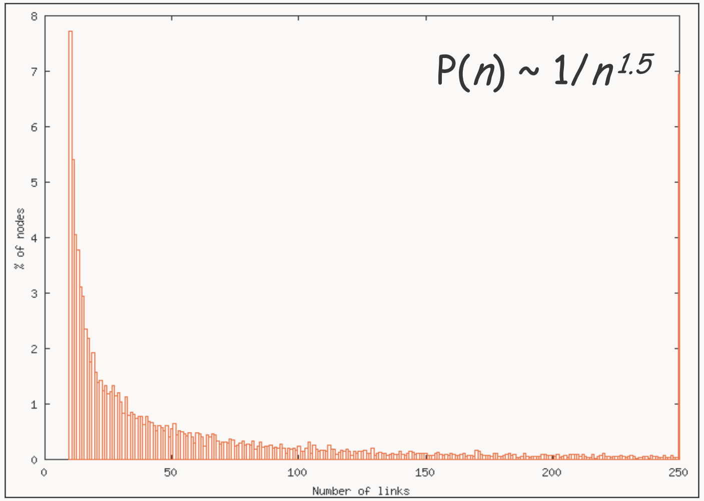
Random graph:
path length ~ log (n)/log(k)
~ 4

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:

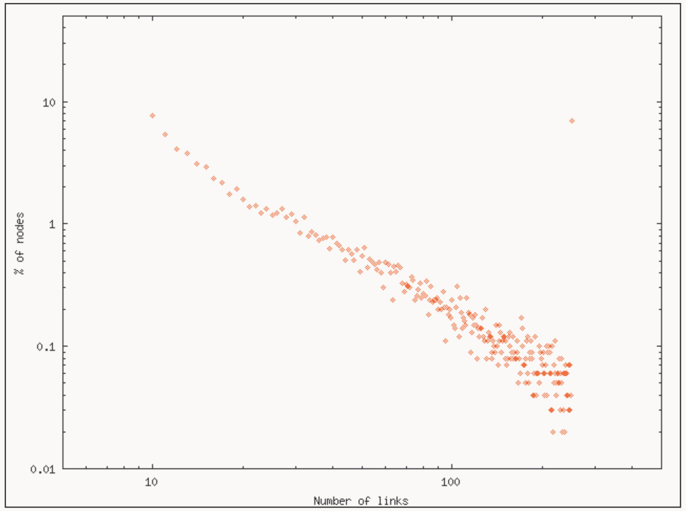
    $P(n) = 1/n^k$
  - Most nodes have only a few connections
  - Some have a lot of links: important for binding disparate regions together

Freenet: Links in the small world [Hong01]



$P(n) \sim 1/n^{1.5}$

Freenet: “Scale-free” link distribution [Hong01]

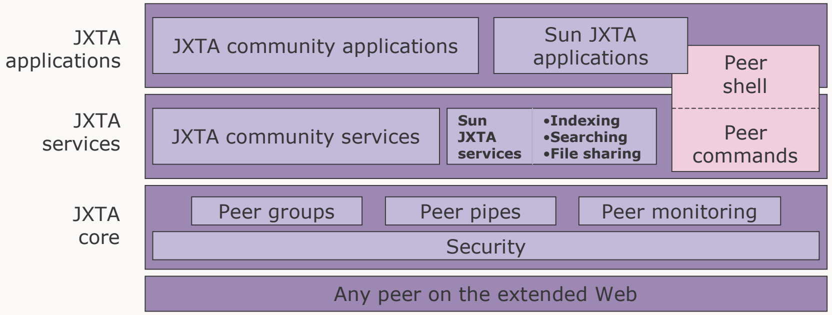


Freenet: summary

- 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

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



JXTA applications

JXTA community applications

Sun JXTA applications

Peer shell

JXTA services

JXTA community services

Sun JXTA services

•Indexing
•Searching
•File sharing

Peer commands

JXTA core

Peer groups

Peer pipes

Peer monitoring

Security

Any peer on the extended Web

Ahead

- More Peer-to-Peer systems: P-Grid
- *Push and event-based systems (depeding on the available time)*
  - *Minstrel*
- Security
  - Attacks, PKI, firewalls, SSL, etc.
- Electronic payment