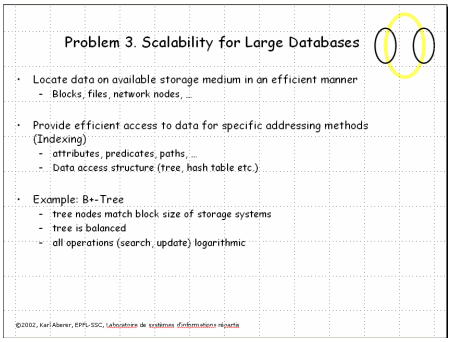
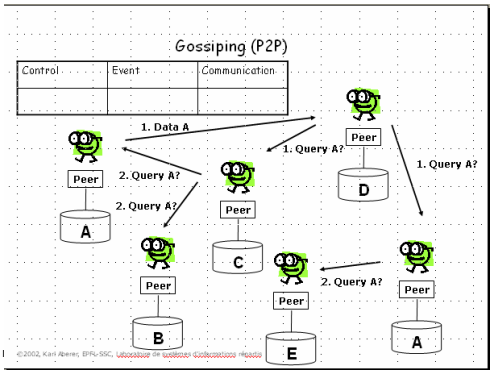
Distributed Data Management
Part 3 - Peer-2-Peer Systems

Overview

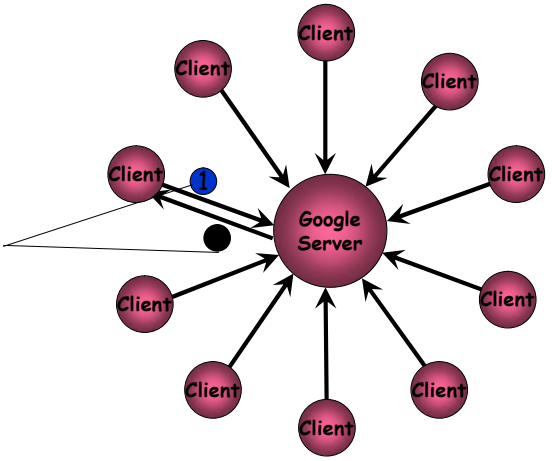
1. P2P Systems and Resource Location
2. Unstructured P2P Networks
3. Hierarchical P2P Networks
4. Structured P2P Networks





Centralized Information Systems

- Web search engine
  - Global scale application
- Example: Google
  - 150 Mio searches/day
  - 1-2 Terabytes of data
    (April 2001)



- Strengths
  - Global ranking
  - Fast response time
- Weaknesses
  - Infrastructure, administration, cost
  - A new company for every global application ?

Google: 15000 servers

Conventional information systems in distributed environments follow a client-server architecture and are therefore centralized. Google is a typical example for this.

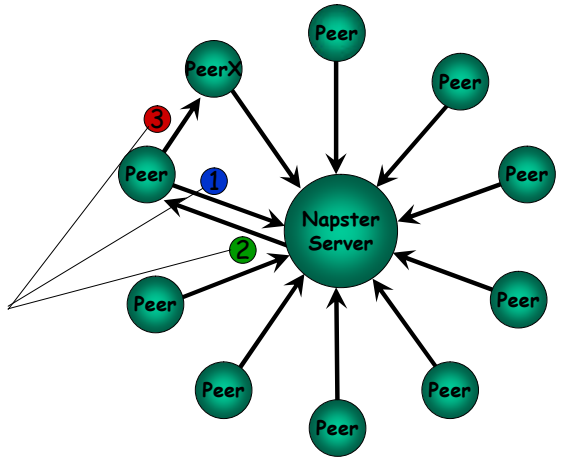
Google as a general-purpose Web search engine is a global scale application. It has to support huge numbers of users, searches and to store large amounts of data, namely (extracts from) all Web pages that are reachable by crawlers and the links among them. This information is centrally stored at the Google server site. The server site consists of a workstation cluster of about 15.000 machines.

When a user wants to search for Web pages it submits the request to the Google server, as shown in the example, and Google returns a list of result pages. Therefore the users are clients that request some service from Google, and Google is the server application that provides the service. We have here a clear distinction of the two roles of providing a service and consuming a service. Google is an example of a centralized information system, because everything that is needed to provide the service is provided from one central node.

Placing such a large investment into a central server pays off: it is possible to search globally, yet very effectively, for information, and the service is fast. The problem is that running the Google server requires a large infrastructure with all the associated cost and administration, which is provided by a company of non-trivial size. The question is, is it necessary to start a major company for every new global scale application ?

(Semi-)Decentralized Information Systems

- P2P Music file sharing
  - Global scale application
- Example: Napster
  - 1.57 Mio. Users
  - 10 TeraByte of data
    (2 Mio songs, 220 songs per user)
    (February 2001)



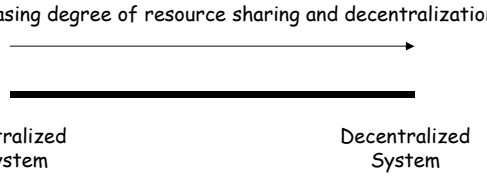
Request and transfer file f.mp3 from peer X directly

Napster: 100 servers

Napster, a music file sharing application, has taken a different approach to providing a global scale application. If we look at the numbers we see that Napster is dealing with a problem of similar scale as Google, for example, when considering the amount of data that needs to be stored. Now, the interesting thing about Napster was how it is implemented: though it deals with about the same amount of data and users as Google, it was able to provide the service which a much smaller number of servers, about 100. How is this possible ? Napster is outsourcing the most resource-intensive part of the work to the users of Napster. This part of the work is the storage and exchange of the music files. The Napster server is just a means to locate the users that can provide the desired service, namely the music file. This is illustrated in the example: first a user asks Napster for some music file. Rather than providing the desired file directly, Napster just tells from which other user the file can be obtained. This operation is inexpensive. In a next step the user directly contacts the other user in order to obtain the desired file. Thus most of the service is provided by the users themselves and the distinction between servers that provide the service, and clients that consume the service is blurred. Therefore the users are no longer called clients, but peers, and the resulting systems are called peer-to-peer systems.

Lessons Learned from Napster

- Strengths: Resource Sharing
  - Every node "pays" its participation by providing access to its resources
    - physical resources (disk, network), knowledge (annotations), ownership (files)
  - Every participating node acts as both a client and a server ("servent"): P2P
  - global information system without huge investment
  - decentralization of cost and administration = avoiding resource bottlenecks
- Weaknesses: Centralization
  - server is single point of failure
  - unique entity required for controlling the system = design bottleneck
  - copying copyrighted material made Napster target of legal attack



If we assess Napster we observe that its main strength derives from the fact that Napster decentralizes the most costly parts of the information service.

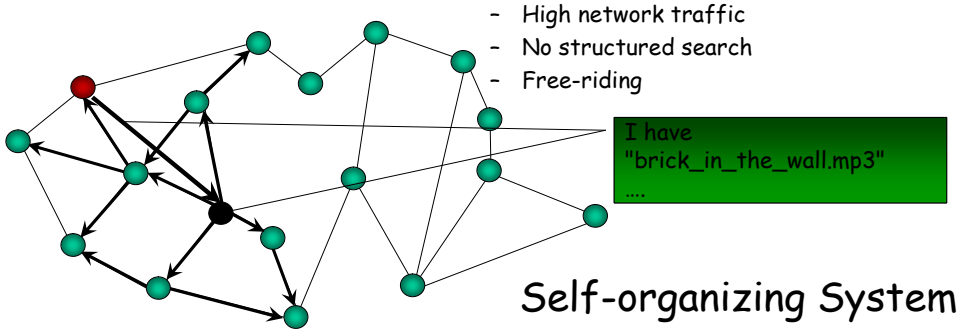
Quite obviously it decentralizes the resource-intensive storage and transmission of files. Less obviously, it also decentralizes the effort of annotating music files, such that they become searchable. This is an important cost, since manual annotation is expensive. And finally, it decentralizes the effort of obtaining ownership of music files (which was of course the main success factor of Napster). Thus we see that decentralization of cost enables the implementation of global scale information systems without huge investment by exploiting already existing and otherwise unused resources provided by the users of the system, both in terms of computational resources as well as knowledge. From a slightly different perspective one can say that Napster avoids different bottlenecks by decentralization of cost and administration.

Still, Napster used for part of its service a central server. Therefore the functioning of the system was dependent on a well-identifiable entity, which eventually led to the closing of the service because of legal steps taken by music industry.

Fully Decentralized Information Systems

- P2P file sharing
  - Global scale application
- Example: Gnutella
  - 40.000 nodes, 3 Mio files (August 2000)

- Strengths
  - Good response time, scalable
  - No infrastructure, no administration
  - No single point of failure
- Weaknesses
  - High network traffic
  - No structured search
  - Free-riding



Self-organizing System

Gnutella: no servers

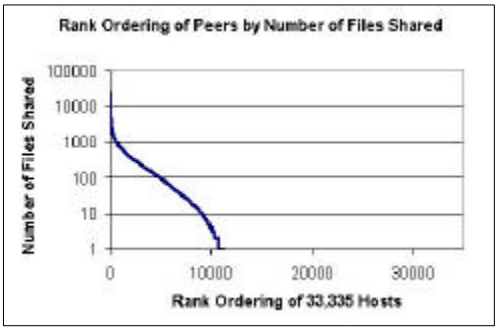
When Napster was shut down, very quickly music file sharing enthusiasts found alternative ways of sharing music files. In particular Gnutella was receiving a lot of attention as an alternative to Napster. One might wonder why Gnutella was not shut down in the same way Napster was.

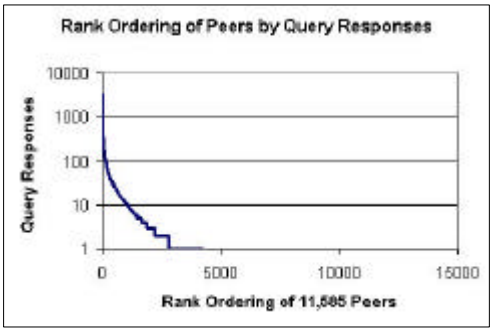
As in Napster in Gnutella music files are stored at the users of the system. But different to Napster there exists no central server that is used to locate music files. So, how can music files be found at all ? The idea is to have the users organizing among themselves. The users form a community in which incoming search requests are forwarded to neighboring users, till the request arrives at a node where the data is available. This approach is called message flooding or gossiping. Actually gossiping works fairly well and replaces thus the search service provided in Napster by a central server. Since the whole functioning of the system depends on the local cooperation of the participating nodes only, and the global properties of the system, such as being able to find a music file in the whole network, emerges from the collective local behaviors, this system is also considered as a self-organizing system.

The advantages of the approach are fairly clear. Since no central server is required the system is very robust (again legal action, but as well against other threats such as server failure or denial of service attack). Setting up the system requires neither special investments nor administration.

But, the approach has also some problems: first gossiping the search messages to all neighboring nodes creates many messages and thus consumes substantial network bandwidth. Since in Gnutella only file names are used to identify files (and not structured annotations as in Napster), no structured searches are possible (e.g. for the author of a song). And since no central control ensures the quality and well-behavior of the participants, social problems such as free-riding are likely to occur.

Free-riding Statistics [Adar00]





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

Many servents share files nobody downloads

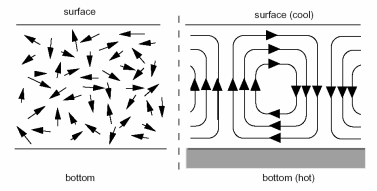
Of 11,585 sharing hosts:

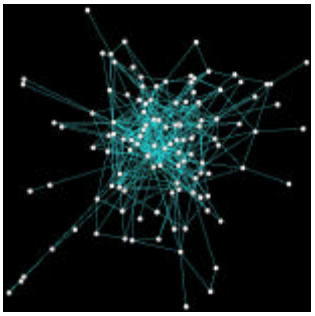
- Top 1% of sites provide nearly 47% of all answers
- Top 25% of sites provide 98% of all answers
- 7,349 (63%) never provide a query response

That social problems occur in Gnutella has actually been shown by studies of the Gnutella network. Here we give one exemplary result. It shows that relatively few peers provide most of the files. And even more dramatic is the fact that among the hosts that share files, most of them share files that nobody is interested in.

Self-Organization

- Self-organized systems well known from physics, biology, cybernetics
  - distribution of control (= decentralization = symmetry in roles = P2P)
  - local interactions, information and decisions
  - emergence of global structures
  - failure resilience





- Self-organization in information systems
  - appear to be costly (if done wrong)
  - Example: search efficiency in Gnutella

We have mentioned that Gnutella builds on the self-organization of the users. In fact, it is an extremely interesting development in the area of computer science that with the upcoming P2P systems, the principle of self-organization starts to come into play at the level of applications, in particular for information management.

Self-organization is a well-known, and intensively studied phenomenon in various disciplines. Self-organization can be characterized in the first place by distribution of control. Equivalently, one can say that there exists no component in the system that has global information on the system or can globally interact with the systems. Each component acts locally and autonomously based on local information on the state of the system. Self-organizing systems exhibit structures and structured behavior, that are not enforced by some outside entity, but that emerge from the aggregation of the local interactions of the components of the systems. Since no distinguished roles can be identified in a self-organizing system, they tend to be very robust against failures. Failures of single components are simply compensated by others. A famous example from physics is the so-called Benard phenomenon, where a magnetic substance, when heated from one side starts to exhibit regular structures (rolls). Another famous example are insect colonies, such as termites, which produce complex structures without any central coordination. Comparable phenomena occur with P2P systems. For example on the image on the right one sees the Gnutella "backbone", a network of server-like nodes that emerges from the interactions in the Gnutella system.

Self-organization is a very powerful principle to organize complex systems, however, this can come at a substantial cost. For Gnutella we have already remarked that this cost is related to the high message traffic that is generated. Therefore in the following we want to understand whether this is unavoidable or whether more efficient forms of self-organization

P2P Architectures

- Principle of self-organization can be applied at different system layers

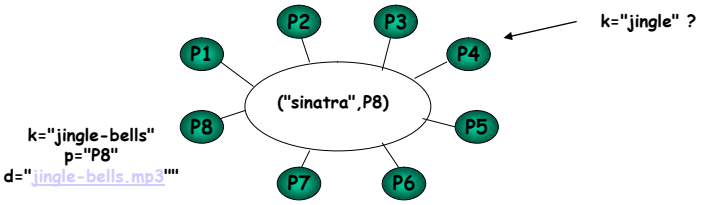
Networking Layer	Internet	Routing	TCP/IP, DNS
Data Access Layer	**Overlay Networks**	**Resource Location**	**Gnutella, FreeNet**
Service Layer	P2P applications	Messaging, Distributed Processing	Napster, Seti, Groove
User Layer	User Communities	Collaboration	eBay, Ciao

- Original Internet designed as decentralized system:
  - P2P overlay networks ~ application-level Internet on top of the Internet
  - support application-specific addresses

The principle of self-organization cannot only be applied to the problem of search, but at many different layers of a computer systems. Among the first self-organizing systems was the Internet itself. The networking layer of the Internet is largely a self-organizing system. We can interpret the first generation of P2P systems for file sharing, more generally, as resource location systems, that provide application-specific addressing capabilities on top of the Internet. Therefore these systems are also often called overlay networks. On the other hand we have seen that, as in Napster, services can be provided in a P2P style. Other examples of such services are SETI (distributed computation, search for extra-terrestrial life) and Groove (messaging, collaboration). Finally human societies are themselves self-organizing systems, and thus applications supporting communities, also build on the principles of self-organization.

Resource Location in P2P Systems

- Problem: Peers need to locate distributed information
  - Peers with address **p** store data items **d** that are identified by a key **k**
  - Given a key **k** (or a predicate on **k**) locate a peer that stores **d**, i.e. locate the index information **(k, p)**
  - Thus, the database we have to manage consists of the key-value pairs **(k, p)**
- Can such a distributed database be maintained and accessed by a set of peers without central control ?



In the following we focus our attention on the problem of resource location, which is considered as a main issue P2P systems address. The problem is to locate in a network nodes that hold a specific resource, a data item. The data item is identified by some key (where a key can be structured, e.g. an XML annotation, or unstructured, i.e. an identifier). For locating data items we have therefore to know the binary relationship between keys and peer addresses, the *index information*. This gives a binary relation, that we can consider as a database. The problem is of how to make this database available in a distributed environment, in particular, how this can be achieved without having central control (Thus a server-based database is excluded).

Although the problem of resource location might appear of being a rather limited one, in its most general form it is nothing else than the problem of data management in a distributed environment. Therefore it is a fundamental building block for higher service layers.

Resource Location Problem

- Operations
  - search for a key at a peer: p->search(k)
  - update a key at a peer: p->update(k,p')
  - peers joining and leaving the network
- Performance Criteria
  - search latency: e.g. searchtime(query) ≈ Log(size(database))
  - message bandwidth, e.g. messages(query) ≈ Log(size(database))
    messages(update) ≈ Log(size(database))
  - storage space used, e.g. storagespace(peer) ≈ Log(size(database))
  - resilience to failures (network, peers)
  - degree of data consistency
- Qualitative Criteria
  - complex search predicates: equality, prefix, containment, similarity search
  - use of global knowledge
  - peer autonomy
  - peer anonymity and trust
  - security (e.g. denial of service attacks)

When comparing different solutions to the resource location problem a number of issues are to be taken into account:

There exist different types of operations that are to be supported: search and update, but also changes in the network structure by having peers joining and leaving the network.

Then certain performance criteria are of interest. From the user perspective it is mainly search latency, that should be low. Also low storage cost is an issue for individual nodes. From a more global perspective the amount of bandwidth consumed for searches and updates is important, since it relates to the total throughput a system can achieve. Finally, resilience to failures is important, since we have to assume that peers act autonomously, and can therefore, for example, any time decide to leave the network or to deny service.

Finally, there are several important qualitative criteria:

•For search it is important to distinguish what search predicates are supported. This has important consequences on the type of applications the resource location mechanism can support.

•Even if no central server is used, it may occur that some global knowledge is involved in the setup of the system. A typical example are approaches that rely on the existence of unique IP numbers, that they use to organize the components among each other. In such an approach this global knowledge takes over the role of a global server and provides global control. This may reduce the degree of autonomy peers have.

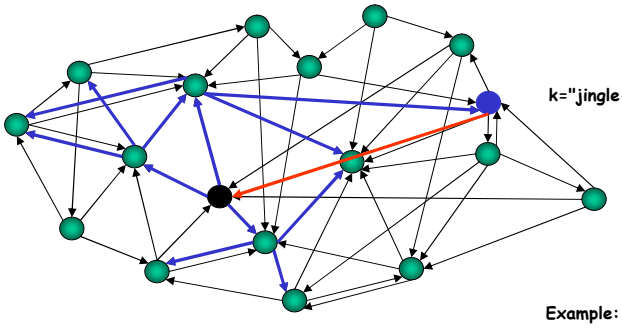
•Since peers interact in a P2P system typically with unknown peers and no central instance guards the proper behavior of the system, issues of anonymity, trust and security play frequently an important role in the design of resource location approaches. We will not further go into those.

Summary

- What is a P2P System ?
- What is emergence ?
- At which layers can the P2P architecture occur ?
- How do we define efficiency for a P2P resource location system ?

2. Unstructured P2P Networks

- No index information is used
  - i.e. the information (k, p) is only available directly from p
- Simplest approach: Message Flooding (Gossiping)
  - send query message to C neighbors
  - messages have limited time-to-live TTL
  - messages have IDs to eliminate cycles



Example: C=3, TTL=2

In an unstructured P2P network the index information is only known to the nodes that keep the data items in question. In other words it is not replicated, which is advantageous from the perspective of updates, since no inconsistent indexes can occur.

The basic way resource location is performed in an unstructured network is by message flooding. Message flooding consists of sending a search request to a fixed number of neighbors C, and messages are forwarded in this manner a maximal number of times only (TTL). In addition peers keep identifiers of forwarded messages, such that they can recognize the case where the same message arrives multiple times, and thus avoid cycles.

Gnutella

- Developed in a 14 days "quick hack" by Nullsoft (winamp)
  - Originally intended for exchange of recipes
- Evolution of Gnutella
  - 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
- Based on message flooding
  - Typical values C=4, TTL=7
  - One request leads to $2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26{,}240$ messages
  - 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)
  - Neighbors are found using a basic discovery protocol (ping-pong messages)

An interesting observation on the origin of Gnutella is that it evolved itself in a very unpredictable and "emergent" way. It was originally never intended for the purpose of music file exchange, and that this happened was rather an accident.

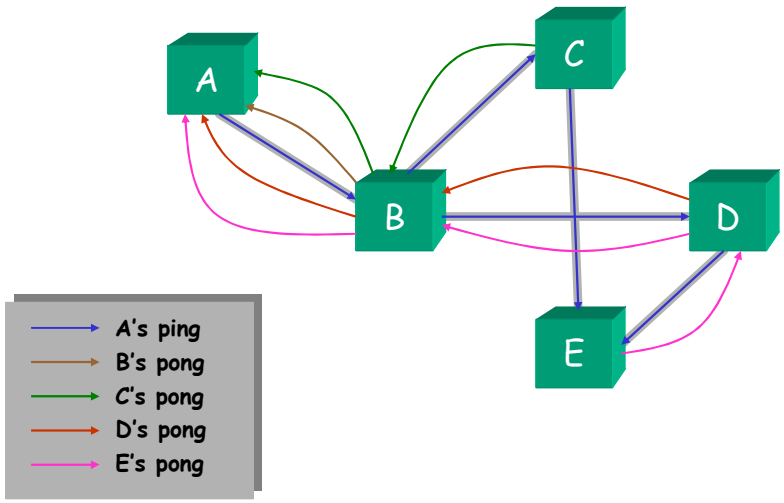
Gnutella is based on simple message forwarding. An important aspect of Gnutella is of how the neighbor network (the overlay network) is constructed, that is required in order to enable the flooding of search requests. First, any node entering the Gnutella system has to know at least one Gnutella node. In principle, this could be any node, in practice some standard nodes that are available through the Internet, are used. By contacting this node the new node can start to discover the network, by sending ping messages, which are answered by other nodes with pong messages. In that way the node learns about other nodes and starts to build up it's connections to neighbors.

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

Actually, an interesting issue is: what is Gnutella ? Gnutella is not a specific software, but it is a protocol (standard), for which many different "clients" exist. The protocol consists of 5 message types that are required to ensure the basic functioning of a Gnutella network: the ping and pong messages are used to establish the network, such that peers can find each other and build their list of neighbors. Search requests (query message) contain besides the search predicate also further information used for routing, such as minimal bandwidth desired. Query hit messages are returned along the same path the query message arrived at a peer. For downloads that require delivery from behind a firewall a special push message allows to forward the information required for sending the file. The problem is that a client cannot pull the information from behind a firewall, since no IP address of the delivering servent is available.

Gnutella: Meeting Peers (Ping/Pong)



This small example illustrates a couple of points:

•ping messages are also flooded through the network.

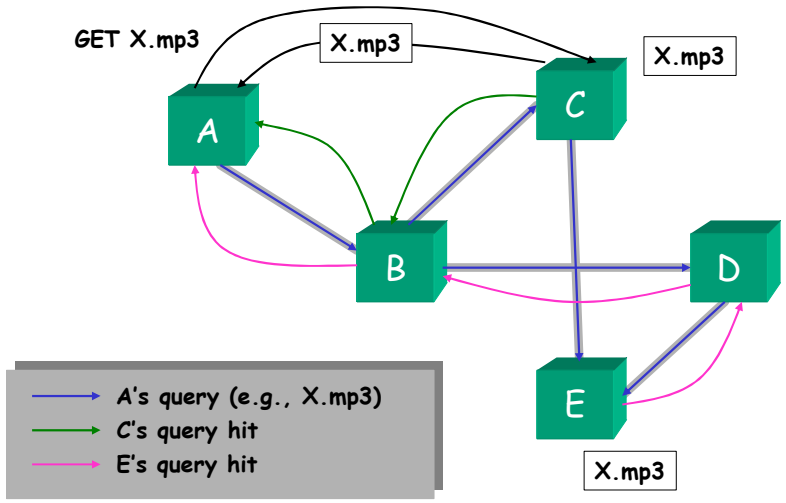
•peers wait till a response to a message arrives. In this manner the response is propagated back along the request path.

•the asynchronous nature of message distribution.

•the detection of cycles.

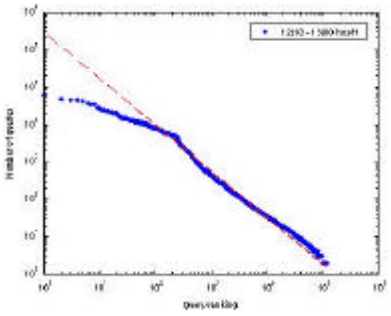
Peers that have been responding to a ping message, can be used subsequently as new neighbors. In that way a peer maintains always a list of potential future neighbors, in case one of the current neighbors is no more reachable, an event that occurs frequently.

Gnutella: Searching (Query/QueryHit/GET)



When a peer sends out a query message, it may obtain several responses indicating which peers provides the requested file. Among those it selects one, and directly contacts it in order to download the file.

Popularity of Queries [Sripanidkulchai01]

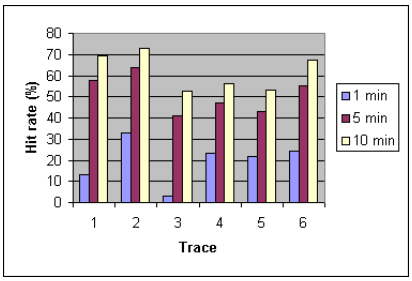


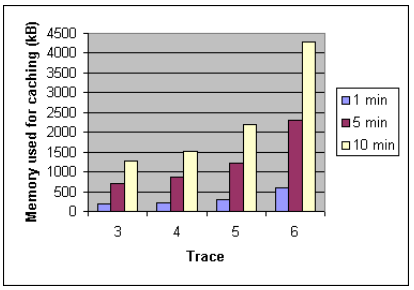
- Very popular documents are approximately equally popular
- Less popular documents follow a Zipf-like distribution (i.e., the probability of seeing a query for the i^{th} most popular query is proportional to $\mathbf{1/(i^{alpha})}$
- Access frequency of web documents also follows Zipf-like distributions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

Gnutella is a system very suitable for experimental evaluation. For example, traces have been produced showing the distributions of queries. It turns out that the distribution is multimodal. The very popular documents are approximately uniformly distributed whereas the less popular documents follow a Zipf distribution. The Zipf distribution occurs very frequently in many practical systems: in a Zipf distribution the frequency of the i-th item is proportional to 1/1^alpha, where alpha is a constant. Since certain documents are particularly popular in Gnutella, one possibility to improve search performance is to cache documents, which increases the chance that both the more frequent documents are cached and then also found in the cache.

Caching in Gnutella [SripanidkulchaiO1]

- Average bandwidth consumption in tests: 3.5Mbps



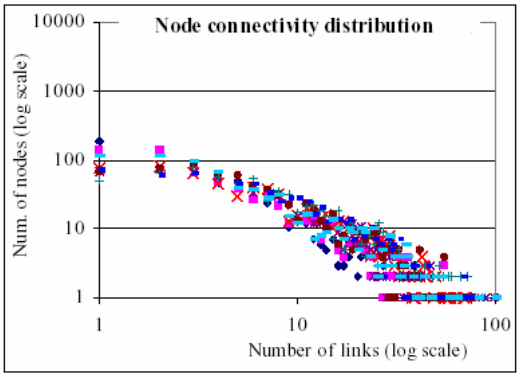


- Experiments cache query and their results at nodes for 1, 5 and 10 minutes based on Gnutella traces
- Result
  - 50% - 70% hit rate for the maximal cache time
  - requires up to 4 MB cache

Simulations based on real traces of Gnutella show that caching can help. With a moderate amount of cache storage (1-4 MB) one can decease the amount of traffic up to 70%.

Connectivity in Gnutella

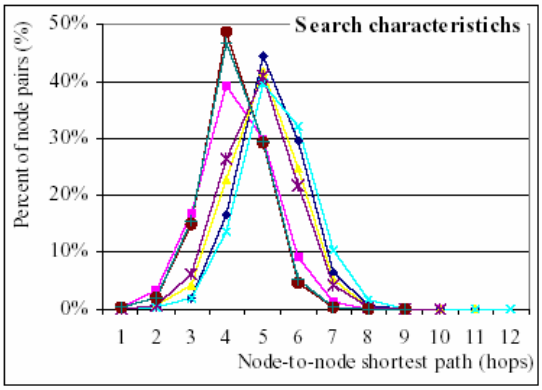
- Follows a power-law distribution: $P(k) \sim k^{-g}$
  - k number of links a node is connected to, g constant (e.g. g=2)
  - distribution independent of number of nodes N
  - preferential attachment
  - low connected nodes follow a constant distribution (?): more robust



A recent discovery on the structure of real networks (social, technical, natural) has been made regarding the distribution of the links to a node. If one assumes that new nodes prefer to attach to well-connected nodes, then one obtains a power-law distribution. The power-law distribution has the interesting property, that it is "scale-free", which means that it does not change with a changing number of nodes N in the graph.

Path Length in Gnutella

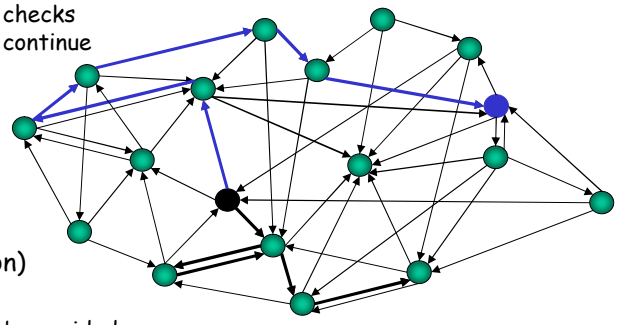
- Average distance between nodes approx. 7



Experimental evaluations have also shown that the diameter of the Gnutella network is about 7. This explains why searches with TTL=7 work well.

Improvements of Message Flooding

- Expanding Ring
  - start search with small TTL (e.g. TTL = 1)
  - if no success iteratively increase TTL (e.g. TTL = TTL +2)
- k-Random Walkers
  - forward query to one randomly chosen neighbor only, with large TTL
  - start k random walkers
  - random walker periodically checks with requester whether to continue



- Experiences (from simulation)
  - adaptive TTL is useful
  - message duplication should be avoided
  - flooding should be controlled at fine granularity

The basic message flooding technique is however limited in what it can do, and in particular in how flexible it can react to the environment. Variations have been proposed, that impose a better control on the flooding. One possibility is to stepwise increase the TTL till the data is found. In that way one avoids unnecessary messages in case the desired data is close in the network. Another alternative is not to flood but to perform random walks in the graph in parallel. In this way no exponential explosion in the number of messages generated with each step of the TTL occurs. If the TTL and the number of random walks is chosen such that the probability of reaching each node is close to 1, then most of the redundant messages in Gnutella (e.g. multiple messages arriving at the same node) can be avoided, and therefore the total bandwidth consumed drops significantly (in simulations up to 2 orders of magnitude). This advantage comes however at another cost: the search latency increases as well, since substantially larger TTL values are used.

Discussion Unstructured Networks

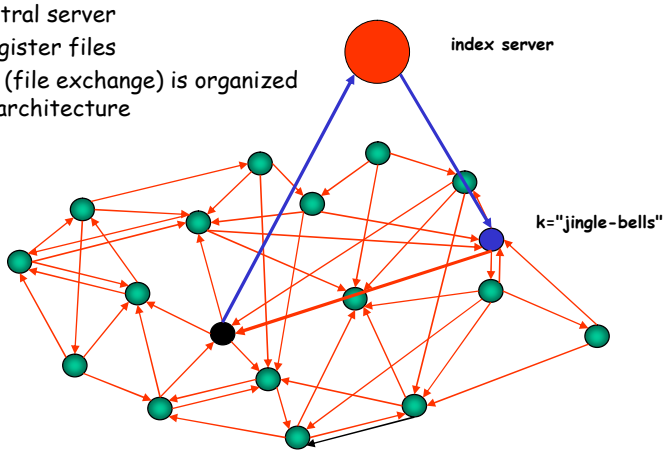
- Performance
  - Search latency: low (graph properties)
  - Message Bandwidth: high
    - improvements through random walkers, but essentially the whole network needs to be explored
  - Storage cost: low (only local neighborhood)
  - Update cost: low (only local updates)
  - Resilience to failures good: multiple paths are explored and data is replicated
- Qualitative Criteria
  - search predicates: very flexible, any predicate is possible
  - global knowledge: none required
  - peer autonomy: high

Summary

- How are unstructured P2P networks characterized ?
- What is the purpose of the ping/pong messages in Gnutella ?
- Why is search latency in Gnutella low ?
- Which are methods to reduce message bandwidth in unstructured networks ?

3. Hierarchical P2P Networks

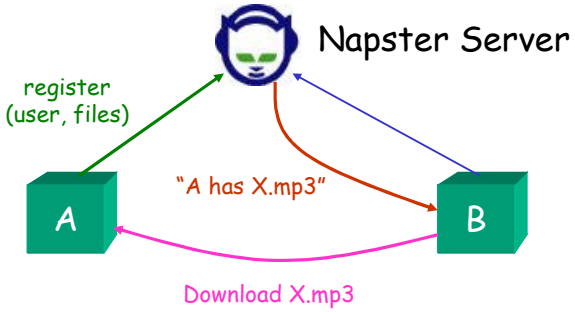
- Servers provide index information, i.e. the information (k, p) is available from dedicated servers
- Simplest Approach
  - one central server
  - user register files
  - service (file exchange) is organized as P2P architecture



The main obstacles for reducing the bandwidth consumed in unstructured P2P networks is the lack of indexing information. Data can only be reliably detected by visiting all nodes. Therefore the use of index information can substantially reduce the bandwidth. The best example for this is Napster, which provided a single index server, that contains information on every data item. The resulting situation is depicted in the figure. For searching a data item a peer has to contact the index server, whereas the service (the file download) is organized in a P2P network. The drawback of such an architecture is the bottleneck resp. single-point-of-failure introduced by the server.

Napster

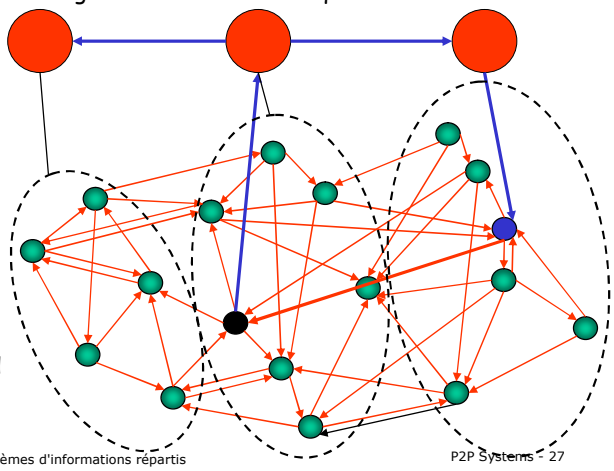
- Central (virtual) database which holds an index of offered MP3/WMA files
- Clients connect to this server, identify themselves (account) and send a list of MP3/WMA files they are sharing (C/S)
- Other clients can search the index and learn from which clients they can retrieve the file (P2P)
- Additional services at server (chat etc.)



The Napster system requires users to identify themselves and to register their data at the server. In addition the server provides additional (community) services, like a chat room. Thus it also supports P2P interaction at the user layer.

Superpeer Networks

- Improvement of Central Index Server (Morpheus, Kaaza)
  - multiple index servers build a P2P network
  - clients are associated with one (or more) superpeers
  - superpeers use message flooding to forward search requests



- Experiences
  - redundant superpeers are good
  - superpeers should have high outdegree (>20)
  - TTL should be minimized

In the meanwhile a new generation of P2P music file sharing systems has appeared. Systems based on the Morpheus technology, such as Kaaza, try to reconcile the efficiency obtained by using an indexing service, with the robustness of a P2P architecture. The idea is not to use a single index server but to organize a set of index servers in a P2P network. Clients attach then to one of the servers. They interact with "their" server in the same way it is done in Napster, i.e. they request from the server index information and register their files at the server. The servers interact among each other like in an unstructured P2P network, by message flooding. In that way clients obtain access to all data from all clients, independent of the server to which they are attached to. The servers are dynamically designated, depending on their reliability and available bandwidth. Experimental studies give some guidelines of how the performance of such a system can be optimized. Multiple servers could be responsible for the same domain, the superpeers should have substantially higher out-degrees than peers in a Gnutella network, such that the TTL can be minimized.

Discussion

- Performance
  - Search latency: very low (index)
  - Message Bandwidth: low
    - with superpeers flooding occurs, but the number of superpeers is comparatively small
  - Storage cost: low at client, high at index server
  - Update cost: low (no replication)
  - Resilience to failures: bad (system has single-point of failure)

- Qualitative Criteria
  - search predicates: very flexible, any predicate is possible
  - global knowledge: server
  - peer autonomy: low

- But: complete decentralization is not an asset per se, everything depends upon how well the overall system works !