Introduction to Peer to Peer systems

Andrea Marin

Università Ca' Foscari di Venezia
Dipartimento di Informatica
Corso di Sistemi Distribuiti

2009

Presentation outline

- What are Peer to Peer (P2P) distributed systems?
- Requirements of P2P systems
- Implementing P2Ps
  - Publishing, searching and accessing to resources
  - Fault tolerance
  - Performances
- The three generations of P2Ps
- Legal issues

Why studying P2P?

- Increased popularity
  - According to *ipoque*, in 2007 the percentage of Internet traffic due to P2P applications was 73.79% (10% Http, 8.2% streaming, 1% VoIP/Skype)
  - High availability of Internet connections and powerful computers at low price
- They allow for interesting features such as high scalability, good fault tolerance, anonymity, relatively easy load-balance ...
- New challenges for the definition of protocols

Note that P2P systems are more that the services offered by the well-know implementations (EMule, Kazaa...)!

What are P2Ps?

The following characteristics are fundamental for P2P systems:

1. Each user (node) contributes resources to the system
2. All the nodes have the same functional capabilities and responsibilities
3. There is not a centrally-administered system (or at least the whole P2P system keeps working if the central system fails)

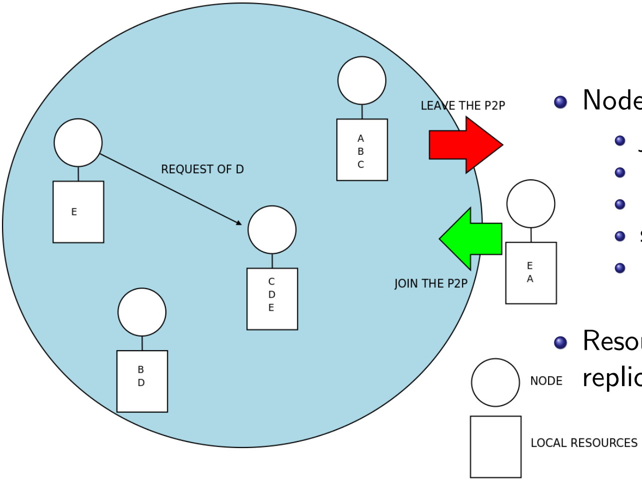
C. Shirky

P2Ps take advantages of resources at the edge of the network

Client/Server vs. P2P

Client/Server	P2P
• Asymmetry between client and server: they different roles • The server provides the resource and the client asks for it • Server infrastructure should not change very often • Implementations: Web, FTP, Web services, . . . • Problems: scalability, fault tolerance, load balance	• Symmetry among the nodes • All the nodes contribute resources • The node infrastructure changes: a node may enter the system or leave unpredictably • Implementations: Napster, Kazaa, Kademlia, . . . • Problems: localizing the resources, trusting nodes, . . .

Schema of P2P systems



- Nodes can...
  - join the system
  - leave the system
  - publish a resource
  - search a resource
  - unplublish a resource
- Resources are replicated

Problems

1. How to identify the resources?
   - Are they static or do they change along the time (e.g. different versions)?
2. Routing
3. Search vs lookup
   - Search for something by description, e.g., locate all documents about TCP/IP performances
   - Lookup: search for a well-know, uniquely identified object, e.g., retrieve the object with a specific identifier
4. Faults, e.g., nodes that suddenly leave the system, or physical connections are lost
5. How to store the resources in order to have a good load-balance of the network?

Generations of P2P

First generation

- A centralized cluster of server that store the index of the resource is required
- Nodes know the centralized server and may sen a search request. The server answers with the physical location (IP address) of a replica of the resource.
- Implementation: OpenNap used by Napster, eDonkey network

Second generation

- Searches do not depend on the central server
- Greater scalability
- First steps toward anonymity
- Better fault tolerance
- Implementations: Freenet, Gnutella, Kazaa, BitTorrent

Third generation

- Presence of middleware layer application-independent
- Implementations: Pastry, Tapestry, CAN, Chord, Kademlia

Napster: history

1. First P2P file sharing application
2. It allows users to share their storage and bandwidth
3. 1999: Shawn Fanning develops Napster
4. December, 1999: RIAA sues Napster for copyright violation
5. In 2000, 50 millions of downloaded Napster clients, Traffic of 7TB a day, 1.5 million of simultaneous users
6. April, 2000: Metallica group sues Napster
7. July, 2000: Granted the RIAA's request of injunction
8. January, 2001: Napster and Bertlesmann AG (one of the labels suing Napster) announce the service will not be free anymore, and will use the DRM technology
9. February, 2001: Napster is judged guilty
10. September, 2001: The subscription service takes the users of Napster from 50M to (almost) zero.

Napster architecture /1

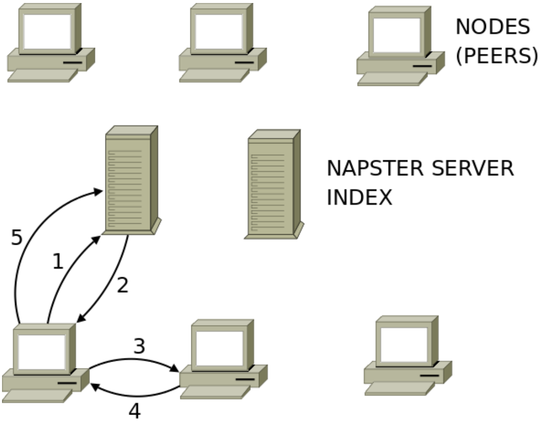
Napster consists of:

- Centralized indexes
- A (large) set of customers providing files to share

When a node performs a search:

- The index server returns the location (IP address) of the set of files matching the query conditions
- The real exchange occurs among the nodes

Napster architecture /2: phases



1. File location request
2. List of peers offering the file
3. File request
4. File delivered
5. Index update

Pro and cons of Napster

Pro

- Easy to implement
- Fast searches

Cons

- Large indexes in the servers
- Servers can become a bottleneck
- Reduced fault tolerance
- Works only with static resources
- Availability of a file is not guaranteed
- Napster servers know everything about the peers
- No anonymity

Legal issues with Napster

Napster developers' thesis

Napster is not liable for the infringement of the copyright owners' rights because it does not participate in the copying process.

Result of the debate

Indexing the resources is an essential part of the copying process, therefore the indexes maintainer are liable.

Note that:

- even if Napster developers' thesis is accepted, the copyright owners' rights would be not respected by peers
- however, it seems hard to persecute some millions of users

Other first generation P2P: eDonkey

Features:

- Hash identification: use of MD4 algorithm
  - Files are divided into chunks of 9,728,000 bytes + a remaining
  - A 128-bit MD4 is computed for each chunk
  - Subject to the birthday attack!
- Complex searches and lookups are available
- Client implementations: eDonkey (closed), eMule, Shareaza, aMule . . .
- Servers are still needed
  - same legal issues of Napster for the server maintainers

2nd generation: motivations

- Remove the need of a central index server $\Rightarrow$ distributed searches
  - Greater reliability
  - Greater fault-tolerance
  - No bottlenecks
  - No server maintainer can be persecuted
- Increased anonymity
  - anonymity is *not* needed for illegal actions
  - anonymity grants freedom of expressions in oppressive societies

Gnutella history

1. March, 1999: Gnutella is posted on the Web. It is developed by Justin Frankel
2. June, 1999: Nullsoft is acquired by AOL
3. June, 1999: Gnutella is retired by AOL
4. June, 1999: Independent programmers maintain Gnutella project
   - the protocol has been reversed engineered
   - open source clones appeared
5. 2001: Major protocol revision improving scalability

Note that, nowadays, Gnutella is a protocol specification rather than the original client!

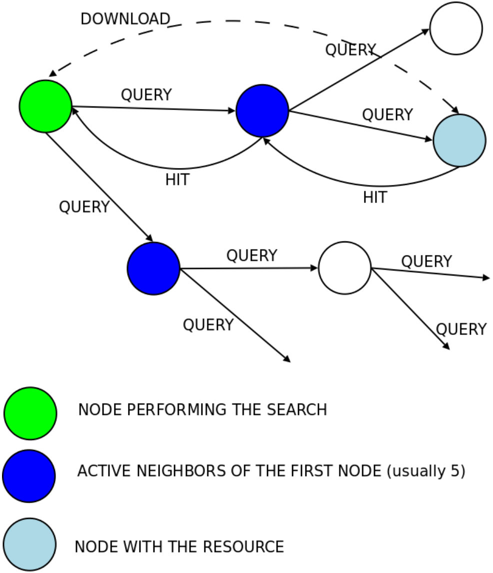
Main features

- No central authority or index server
- Search algorithm is based on flooding
  - at start-up a node knows few other nodes, i.e., the neighbors (problem how?)
  - the queries are forwarded to all node neighbors
  - the propagation is limited by a maximum number of hops
  - version 0.4: a node knows 5 neighbors, hop limit is 7
  - versione 0.6: nodes can be either a standard peer (leaf) or a ultrapeer. A standard peer is connected to 3 ultrapeer and a ultrapeer is connected to 32 other ultrapeers. The maximum number of hops is 4.

Phases for the network join operation

1. Node bootstrap: it must know at least one other node
    - pre-existing address list of possibly working nodes
    - web-caches of known nodes
    - IRC
2. The new node perform a PING request to the known neighbor(s)
3. PING messages are forwarded
4. Each node available for connection answer a PONG to the new node
5. The process continues until the new node is connected to a given number of nodes
6. The new node keeps in cache the addresses it has not pinged yet, and the addresses that were unavailable

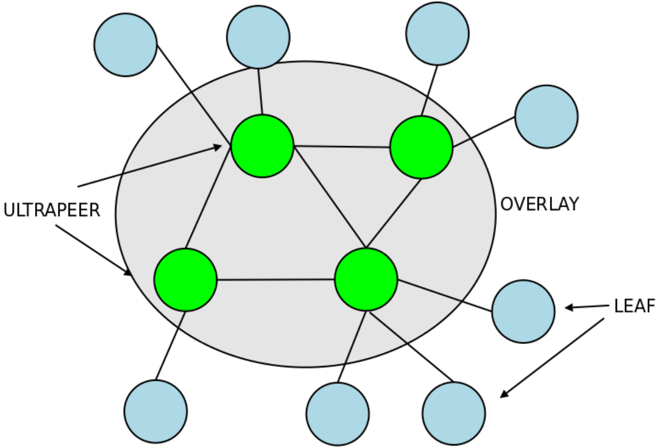
Search operation v0.4



- Fully de-centralized
- Distributed search cost
- Large amount of traffic due to queries
- Bottlenecks for slow peers or full communication channels
- Worst case of search time is $2D$ with D the TTL

Topology of Gnutella v0.6

Network with two layers:



Leaf vs. Ultrapeer

A ultrapeer node. . .

- It works as a proxy for the leaves
- It can answer to other nodes ping with a pong (possibly using a pong-cache)
- It must have a fast Internet conenction
- Must be able to accept TCP and UDP segments from the outside (no firewall)

A leaf node. . .

- It can just connect to ultrapeer nodes
- It can have a slow connection (even a dial-up)
- In order to become ultrapeer a node has to join the network again

Connection slots

A connection slot represents the possibility of a node to connect to other nodes.

- A leaf can connect to 3 ultrapeers
- A ultrapeer can connect to 32 other ultrapeers. This value is known as the network degree
- A ultrapeer can connect to 30 leaves

Lookup in Gnutella v0.6

Objectives

- Find a resource with the minimum number of messages
- Popular resources require less efforts

Definition (Dynamic querying)

The algorithm used by Gnutella is called dynamic querying and consists of 3 phases:

1. Search our leaves
2. Probe query
3. Controlled broadcast

The algorithm is run only by ultrapeers.

Search our leaves

1. The ultrapeer node sends the query to all its leaves setting TTL=1
2. It waits for a time $T = 2.4ms$
3. The leaves with that resource answer to the ultrapeer

Probe query

1. The ultrapeer sends to all the adjacent ultrapeers the query with TTL=1
2. The adjacent ultrapeers send the query to their leaves setting TTL=1
3. The answers are sent back to the first ultrapeer
4. If the Probe query finds more than 150 answers, then the algorithm terminates

The Probe query estimates the popularity of a resource.

Controlled broadcasting

1. Every 2.4ms the ultrapeers send the query to one adjacent ultrapeer setting TTL=2 or TTL=3 according to the number of answers obtained
2. Once 150 answers are obtained the algorithm terminates
3. After 200s from the beginning of the search the algorithm if stopped

GNutella: Query routing protocol (QRP)

Objective: reduce the number of messages sent for a query in the Dynamic querying algorithm

1. The leaf node send to the ultrapeer a description of the shared resources
2. The description is contained in a QRP table
3. QRP tables can be updated
4. Avoids to send queries to leaves that do not share resources or do not want to answer to queries

QRP table structure

- It is an array of $65,536$ Bytes
- If resource *filename.ext* is shared, then a hash of the name of the file is computed and the relative bit in the QRP table is set
- Then the ultrapeer will send to that leaf the queries for *filename.txt* (and maybe others)
- The ultrapeer can generate a composite QRP table using the OR of the QRP tables of its leaves
- Composite QRP tables can be exchanged among the ultrapeers