Overlays

Semi-decentralized system *(Napster)*

- Intensive part left to users
- Bootstrap and easy part left to servers

Fully-decentralized system

- No server-like things at all

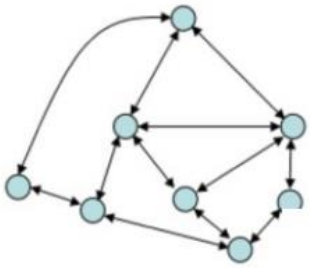
Through Napster's experience, we learned that resource sharing is good, since it's every node "pays" its participation by providing access to its resources *(disk, network, files, ..)*, avoiding resource bottlenecks. But since there still was a server *(used to locate the users that could provide a file)*, there still was a single point of failure and a bottleneck in the design.

Gnutella was an attempt to Fully-decentralized system where, as in Napster, music files were stored at the users of the system. Differently though, there was **no central index**, meaning there was no central server to locate these files. To find them, peers would establish **non-transient direct connections** between themselves, defining a **network overlay**. Through Gnutella's experience, we learned that this way we reached good response time *(scalability)*, no infrastructure/administration needed so no single point of failure, **but** the network traffic was really high, there was no way to do complex queries and *free-riding*.

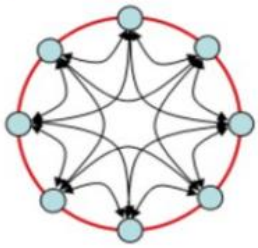
Overlay Network = P2P networks. A logic network connecting peers laid over the IP infrastructures (app lvl abstraction).

A **P2P protocol** defines the set of messages that the peers exchange, their format, semantics, … .
It usually defines a routing strategy and has some sort of identification of the peer through a unique identifier *(hash)*.

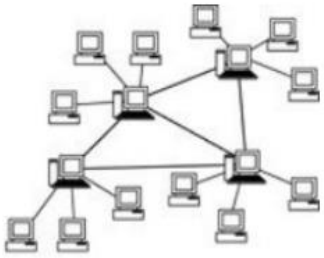
- Unstructured overlays



- Structued overalys: distributed hash tables



- Hybrid overlays: SuperPeer



Unstructured overlays: Peers are arbitrarily connected, which means that look-up algorithms are:

- **Flooding:** *TTL messages [false negatives!], ID to each msg to prevent cycles*
- **Expanding Ring:** *flood with increasing TTL, choosing a random subset of neighbors)*
- **Random Walk:** *a path constructed by taking random directions each step. It can be described by a Markov chain (memory-less) -> K-RandomWalk: K query messages, each path is called walker*
  The (K-)RandomWalk reduces the number of messages to (K*)TTL instead of exponential-like in Flooding, but search latency increases. *It's possible to bias the walk towards high-degree nodes!*

The advantages are that it's easy to maintain and highly resilient, but the lookup cost is linear in N which implies very low scalability. *Examples: Gnutella, BitTorrent, BitCoin, …*

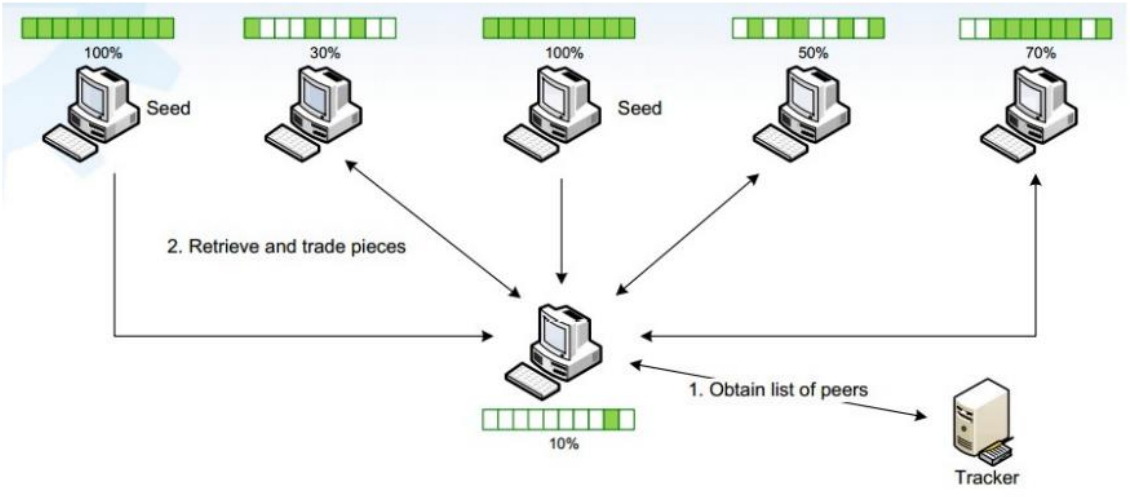
In this type of network, the bootstrap is solved via a **WebCache**, known web servers storing the IP address of a set of stable peers, then each client has an **internal cache** where they store the peers contacted in the previous and current sessions *(dynamically updated by gossiping with neighbor peers)*.

Structured overlays: The choice of the neighbors is defined by a criteria in order to guarantee scalability *(key-based lookup).* *Examples: CAN, Chord, Pastry, Emule (Kademlia), …*

Hierarchical overlays: The presence of peers *(P)* and super-peers *(SP)*. *P* are connected to *SP* and *SP* know the *P* resources. The lookup is made by flooding **only** the *SP*, so there are lower lookup costs and improved scalability at the cost of lower resistance to *SP churn*. *Examples: Skype, Kazaa, ..*

Case study: BitTorrent

It's a **Content Distribution Network** (CDN). A distributed set of hosts cooperating to distribute a large set of data to end-users. The peer which is going to share a file generates a **descriptor** of the file *(file.torrent)* and publishes it on a server. The descriptor includes a reference to a **tracker**, an active entity that coordinates the peers sharing the file. The set of peers collaborating to the distribution of the same file coordinated by the same tracker is called **swarm**, which includes **seeders** *(own 100% of the file)* and **leechers** *(own part or no parts of the file)*



The picture shows how the content is split into **pieces** *(256KB – 2MB)*, the smaller unit of retransmissions. Once downloaded a piece, you check it with the SHA1 algorithm against what the *file.torrent* and it if fails you request the retransmissions. This implies the existence of a SHA-1 hash per piece in the torrent. Furthermore, pieces are split into **blocks** *(16KB).*

To bootstrap, the peer with the *file.torrent* retrieves the tracker's URL and connects to it, sends to the tracker infos about its identity *(identifier, port, …)*, .., then the tracker returns a random list of peers already in the torrent *(typically 50 peers)*. The peer can finally connect to some of them *(at most 40 outgoing connections)*.

Note: *Now BitTorrent is trackerless through Kademlia DHT (avoid centralization point == tracker)*

Free Rider: Peers that do not put their bandwidth at disposal of the community. Complex problem because there's no centralized entity to control that they exist, there's no way to impose behaviors due to modified clients, etc. But a good behavior for a P2P network is when everybody cooperates, which implies that Free Riders must be removed.

A possible approach is **reciprocity**: give to a peer if and only if the peer gave already. Another strategy is a dynamic one where we monitor its connection, or even one based on the *prisoner dilemma*. The one we will see is the **choking/unchoking** algorithm.

Divide the time in **rounds** *(10s)*. For each round, decide who will receive the data. We take into accounts these variables:

- ***Interested / Uninterested:*** *wants / doesn't want a piece*
- ***Chocked / Unchocked:*** *wants / doesn't want to send you data*

All connections start as *Uninterested* and *Choked*. To reach a state where a local peer can receive data, it must send a message *Interested* to the wanted peer. Then he waits for an *Unchoke* message and only then starts to ask for pieces with *Request* messages.

There are many versions of this, the most used is where each peer evaluates periodically for each neighbor the download speed in the previous round and decides to either choke him or not, then proceeding to unchoke a fixed number of peers (3) every 10s *(avoid wasting resources by rapidly choking and unchoking peers)*. Finally, optimistic unchoking (*one random peer is unchoked)* to allow newcomers to download resources (every 30s).

Peer Wire Protocol *(the used P2P protocol)*

Handshake: *two-way handshake to initiate a connection between two peers (symmetric connection)*

Keep Alive: *sent every 2 mins if no msg received from the connection*

Messages (PWP)

Bitfield: *first msg sent after the handshake. I-th bit to 1 if the peer has the piece, 0 otherwise*

Have: *msg sent from a peer to the swarm, indicating that a piece is completely downloaded*

Interested: *sent to indicate that it's interested in a specific piece (contains the piece index)*

Not Interested: *after receiving a HAVE msg to say that it's not interested in that piece anymore*

Choke: *sent from A to B when A chokes B (== A won't accept B's requests)*

Unchoke: *same as before, but B requests are allowed*

Request <index, begin, length>: *request a specific block in a specific piece (only after UNCHOKE)*

Piece <index, begin, block>: *to send blocks*

Cancel: *msg sent to indicate that a piece is already there and it's not interested anymore. Used only in end-game mode.*

To **select** the pieces of the file to download there are different possibilities, like *Strict Priority, Random First Piece, Rarest First* and *End Game [broadcast requests when close to the full download].*