Contents

• Last week
  • Napster
  • Skype
  • Gnutella
• This week:
  • Gnutella summary
  • Freenet
  • BitTorrent and analysis
• Next week
  • BitTorrent and incentives
  • Power laws and structured networks
Ultra nodes summarize keywords with Bloom filters (BF) and propagate them.

Ultra nodes > 32 connections, flat unstructured network, 32 leafs. Idea is to allow hubs to form.

Leafs connect to 3 or more ultra nodes, inform hashed keywords to ultra node.

Search is propagated by ultra nodes based on routing table (BF), TTL is used to adjust query results by ultra nodes.
Mapping the Gnutella Network

Map the network by crawling or monitoring hubs
Example: Gnutella v0.4 random topology has problems

Overlay networks can result in really bad application layer routing configurations unless the underlay is taken into account!

Hubs help here if they are chosen wisely.
Clustering can result in 3-5 orders of magnitude better performance than Gnutella v0.4
**Improvements**

- **Selective** flooding can be combined with spanning trees, random walks, etc. Good for bootstrapping search.
- GIA by Y. Chawathe et al. (SIGCOMM 2003) outperforms Gnutella v0.4 by 3-5 orders of magnitude

- **Design principles**
  - Explicitly account for node heterogeneity
  - Query load proportional to node capacity

- Make high-capacity nodes easily reachable
- Dynamic topology adaptation converts them into high-degree nodes
- Make high-capacity nodes have more answers
- Biased random walks and overload avoidance
Bloom filters are probabilistic structures used to store dictionaries. A bit-vector that supports constant time querying of keywords. Easy to merge two filters. Many variants. If space is at premium:

<table>
<thead>
<tr>
<th>Decrease</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of hash functions (k)</td>
<td>Number of hash functions (k)</td>
</tr>
<tr>
<td>- Less computation</td>
<td>- More computation</td>
</tr>
<tr>
<td>- Higher false positive rate</td>
<td>- Lower false positive rate</td>
</tr>
<tr>
<td>Size of filter (m)</td>
<td>Size of filter (m)</td>
</tr>
<tr>
<td>- Smaller space requirements</td>
<td>- More space is needed</td>
</tr>
<tr>
<td>- Higher false positive rate</td>
<td>- Lower false positive rate</td>
</tr>
<tr>
<td>Number of elements in the inserted set (n)</td>
<td>Number of elements in the inserted set (n)</td>
</tr>
<tr>
<td>- Lower false positive rate</td>
<td>- Higher false positive rate</td>
</tr>
</tbody>
</table>
Example Bloom filter
**Data:** \( x \) is the object key to insert into the Bloom filter.

**Function:** \( \text{insert}(x) \)

**for** \( j : 1 \ldots k \) **do**

```markdown
/* Loop all hash functions \( k \) */
\( i \leftarrow h_j(x); \)

**if** \( B_i == 0 \) **then**

```markdown
/* Bloom filter had zero bit at position \( i \) */
\( B_i \leftarrow 1; \)
```

**end**

**end**

**Algorithm 1:** Pseudocode for Bloom filter insertion
Data: $x$ is the object key for which membership is tested.
Function: $ismember(x)$ returns true or false to the membership test

$m \leftarrow 1$;
$j \leftarrow 1$;

while $m == 1$ and $j \leq k$ do

    $i \leftarrow h_j(x)$;
    if $B_i == 0$ then
        $m \leftarrow 0$;
    end

    $j \leftarrow j + 1$;
end

return $m$;

Algorithm 2: Pseudocode for Bloom member test
BF False positive probability is given by:

\[
\left( 1 - \left(1 - \frac{1}{m}\right)^{kn} \right)^k \approx \left(1 - e^{-kn/m}\right)^k.
\]

Optimal number of hash functions \(k\):

\[
k_{opt} = \frac{m}{n} \ln 2 \approx \frac{9m}{13n}.
\]

Size of filter given optimal number of hash functions:

\[
m = -\frac{n \ln p}{(\ln 2)^2}.
\]

Details in the survey paper available on course page.
If false positive rate is fixed, the filter size grows linearly with inserted elements.
<table>
<thead>
<tr>
<th></th>
<th>Gnutella v0.4</th>
<th>Gnutella v0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decentralization</td>
<td>Flat topology (random graph), equal peers</td>
<td>Random graph with two tiers. Two kinds of nodes, regular and ultra nodes. Ultra nodes are connectivity hubs</td>
</tr>
<tr>
<td>Foundation</td>
<td>Flooding mechanism</td>
<td>Selective flooding using the super nodes</td>
</tr>
<tr>
<td>Routing function</td>
<td>Flooding mechanism</td>
<td>Selective flooding mechanism</td>
</tr>
<tr>
<td>Routing performance</td>
<td>Search until Time-To-Live expires, no guarantee to locate data</td>
<td>Search until Time-To-Live expires, second tier improves efficiency, no guarantee to locate data</td>
</tr>
<tr>
<td>Routing state</td>
<td>Constant (reverse path state, max rate and TTL determine max state)</td>
<td>Constant (regular to ultra, ultra to ultra). Ultra nodes have to manage leaf node state.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Performance degrades when the number of peer grows. No central point.</td>
<td>Performance degrades when the number of peer grows. Hubs are central points that can be taken out.</td>
</tr>
</tbody>
</table>
Freenet

The unstructured P2P systems presented so do not offer good security and privacy features.

Many of these shortcomings are addressed in the Freenet file sharing system.

This system emphasizes anonymity in file sharing and protects both authors and readers.
Freenet II

The system works in a bit different way to Gnutella, because it allows users to **publish** content to the P2P networks and then disconnect from the network.

The published content will remain in the network and be accessible for users until it is eventually removed if there is not enough interest in the data.

The Freenet network is responsible for keeping the data **available and distributing** it data in a **secure** and **anonymous** way.
Overview of Freenet

The Freenet network is a decentralized loosely structured overlay network similar to Gnutella. The system is a self-organizing P2P network and creates a collaborative virtual file system by pooling unused disk space.

Prominent features of the system include emphasis on security, publisher anonymity, and deniability. Moreover, the system also focuses on data replication for availability and performance.

Each node maintains a dynamic routing table to be able to process requests for certain files. In order to obtain a file, a user sends a request message that includes a key for the desired file.
Freenet components

The Freenet network consists of three crucial parts:

- **Bootstrapping**, which pertains to how a new node enters the network
- **File identifier keys**, which are needed to be able to find files in the network. The keys can be derived using several different ways and each of them have their implications for the system and security
- **Key-based routing**, which is the process of finding a node that hosts the desired file
Freenet messages

Freenet has the following central messages:

- **Data insert.** This message allows a node to insert new data into the network. The message includes a key and the data file.

- **Data request.** A node requests for a certain file. The request contains the key of the file.

- **A reply.** The reply is sent by the node that has the requested file. The actual file is included in the reply message.

- **Data failed.** This operation denotes a failure to locate a file. The message will contain the location of the node where the failure occurs and the reason.
Search in Freenet

Depth-first search
With backtracking
Use the closest file key

Routing table:

<table>
<thead>
<tr>
<th>id</th>
<th>next_hop</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Object request
Reply
Failed request

Search and insert are similar. Insert uses search to find a closest node after TTL, collision gives a search result. If success, file cached at intermediate nodes.
Freenet versions

There are significant differences between Freenet protocol versions. Before version 0.7, the system used a heuristic algorithm where nodes did not have fixed locations and routing was based on finding the closest node that advertised a given key. Upon successful request, new shortcut connections were sometimes created between the requesting node and the responder, and old connections were discarded. This was changed to an algorithm that clusters nodes together and creates shortcuts (trying to leverage small world properties).
Routing in Freenet

The **new** algorithm introduced the notion of **node location**, which is a number between 0 and 1.

This location metric is used to **cluster** nodes.

File names are also transformed into numbers
- Easy to compare file number to node number

*Idea: place data to numerically closest node, cache data towards this node, locally greedy routing*

This kind of approach works well with popular data, the more a file is requested by clients, the more it will be cached by intermediate nodes.
Freenet Routing in Detail

1. When a client issues a request for a file, the node first checks if the file is locally available in the data store. If the file is not found, the file key is turned into a number in a similar fashion.

2. The request is then routed to the node that has the numerically closest location value to the key.

3. This routing process is repeated until a preset number of hops is reached.

4. If the desired file is found during the routing process, the file is cached on each node along the path (given that there is room).

Insertion of a file is similar.
Location Swapping in Freenet

Node swap is needed for clustering

Nodes swap location information in order to position its location in an optimal way to its peers (calculated based on distance to neighbours’ location)

A node **randomly** chooses a node in its proximity and sends a swap request

A swap is performed if the **swap reduces distances**, otherwise the swap is performed with a **probability** based on the calculated distances

Deterministic swap **always decreases** the average distances of a node to its neighbours, probabilistic swap is used to escape **local minima**
The routing and location algorithm result in four key properties:

- Over time nodes tend to specialize in requesting for similar keys as they receive search requests from other nodes for similar keys.
- As the consequence of the above, nodes tend to store similar keys over time. This stems from the caching of requested files.
- Keys are semantic free and the similarity of keys does not result in similarity of the files.
- Higher-level routing is independent of the underlying network topology.
Problems with Freenet Routing

The new Freenet routing algorithm is unable to provide performance guarantees with active malicious participants.

The algorithm also degenerates over time (even with passive adversaries) if the network experiences churn.

The recommended approach to address both problems is to periodically reset the locations of peers.

Also: no guarantee to locate data and the network can forget old data (no requests → no replication).
Privacy in Freenet

Privacy is realized using a variation of Chaum’s mix-net scheme for anonymous communication. Messages travel through the network through node-to-node chains. Each link is individually encrypted. Each node in this chain knows only about its immediate neighbours, the endpoints are decoupled from each other. This approach protects both the publishers and the consumers. It is very difficult for an adversary to destroy a file because it is distributed across the network.

Challenges: Location swapping exposes network topology.
MIX

MIX routes and forwards messages from several senders to several receivers in such a way that no relation between any particular sender and any particular receiver can be discerned by an external observer.

The classic application of MIX has been untraceable digital pseudonyms.

Other application cases are synchronous and asynchronous communication systems, and electronic voting systems. Most applications use a cascade of MIXes forming so-called MIX-net.

MIX-nets obfuscate the relation between the senders and receivers.

Onion routing is based on this idea.
1. User selects a sequence of mixes and a destination.
2. Onion-encrypt the message.
3. Send the message, removing a layer of encryption at each mix.

**Onion Encrypt**

1. Proceed in reverse order of the user’s path.
2. Encrypt (message, next hop) with the public key of the mix.
Privacy in Freenet II

MIX is used as a pre-routing phase in Freenet

A request goes through one or more MIX stages (with nested encryption) to the first Freenet node

Offers sender anonymity and security for the first hop
Freenet file types

**CHKs (Content Hash Key)** are useful for single non-mutable files, for example audio and video files (simply a hash of the description).

**SSKs (Signed Subspace Key)** are intended for sites with mutable data. A typical usage case involves a Web site. Hash of a public key, symmetric key (hash of the description), signature. Defines a personal namespace that anyone can read but can be written only with the private key.

**USK (Updatable Subspace Key)** are used for creating a link to the most current version of an SSK site. They are essentially wrappers around SSKs.

**KSK (Keyword Signed Keys)** are used for human-understandable links that do not require trust in the creator. The keypair is generated from the keyword (a string).

**Indirect files** allow metadata-based distributed pointers to a file.
KSK example (retrieval using strings)

String → Key generation → Public key
          ↓            ↓
          Private key File and Signature
          ↓
          Encrypt
          ↓
          Stored file

File key → Hash → File key
Example of KSK Usage

1. A deterministic algorithm is used to generate a cryptographic public/private key pair and a symmetric key based on the file description. The same description will result in the same keys irrespective of the node performing the computation.
2. The public key is stored with the data and it will be used to verify the authenticity of the data.
3. The file is encrypted using the symmetric encryption key.
4. The private key is used to sign the file.
5. In order to retrieve the file, a user needs to know the file description. This description can then be used to generate the decryption key.
SSK example (retrieval using strings and public keys)
Freenet indirect files (keyword CHKs pointers)
<table>
<thead>
<tr>
<th><strong>Freenet v0.7</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decentralization</strong></td>
</tr>
<tr>
<td><strong>Foundation</strong></td>
</tr>
<tr>
<td><strong>Routing function</strong></td>
</tr>
<tr>
<td><strong>Routing performance</strong></td>
</tr>
<tr>
<td><strong>Routing state</strong></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
</tr>
</tbody>
</table>
BitTorrent

BitTorrent is based on the notion of a torrent, which is a smallish file that contains metadata about a host, the tracker, that coordinates the file distribution and files that are shared.

A peer that wishes to make data available must first find a tracker for the data, create a torrent, and then distribute the torrent file. Other peers can then using information contained in the torrent file assist each other in downloading the file. The download is coordinated by the tracker. In BitTorrent terminology, peers that provide a complete file with all of its pieces are called seeders.
BitTorrent: Downloading Files

1. Upload torrent file
2. Provide first seed
3. Post search request and retrieve link to torrent file
4. Contact tracker
5. Contact seeder for pieces
6. Trade pieces with peers
Difference to HTTP

A BitTorrent file download differs from an HTTP request in the following ways:

– BitTorrent uses multiple parallel connections to improve download rates, whereas Web browsers typically use a single TCP Socket to transfer HTTP requests and responses
– BitTorrent is peer-assisted whereas HTTP request is strictly client-server
– BitTorrent uses the random or rarest-first mechanisms to ensure data availability, whereas HTTP is incremental
A solution to the broadcasting problem

BitTorrent attempts to solve the broadcasting problem, which has the goal of disseminating $M$ messages in a population of $N$ nodes in the shortest time.

In an environment in which the nodes have bidirectional communications and the same bandwidth, the lower bound on download time (rounds) is given by $M + \log_2 N$, the unit is the time it takes for two nodes to exchange a message.

This problem can be solved optimally with a centralized scheduler; however, BitTorrent lacks this centralized component and furthermore it does not have a completely connected graph as well.

BitTorrent therefore has a heuristic approach to solving this problem that works very well in practice.
Lower Bound

Assume bidirectional communications and the same bandwidth.

The lower bound on download time (rounds) is given by $M + \log_2 N$, the unit is the time it takes for two nodes to exchange a message.

Proof: [stat.haifa.ac.il/~gweiss/publications/p2pjos.pdf](stat.haifa.ac.il/~gweiss/publications/p2pjos.pdf)

Idea: in the first phase one client has the messages, and in the next phase $\log_2 N$ rounds are needed to inform the N-1 clients. The log comes from the P2P behaviour in which the clients utilize parallel data transfers to propagate the messages.
Peer selection is about selecting peers who are willing to share files back to the current peer
- Tit for tat in peer selection based on download-speed.
- The mechanism uses a choking/unchoking mechanism to control peer selection. The goal is to get good TCP performance and mitigate free riders

Optimistic unchoking
- The client uses a part of its available bandwidth for sending data to random peers
- The motivation for this mechanism is to avoid bootstrapping problem with the tit for tat selection process and ensure that new peers can join the swarm
Characteristics of the BitTorrent protocol II

- **Piece selection** is about supporting high piece diversity
  - Local Rarest First for piece selection (start with random, then finally use end game mode)
  - BITFIELD message after handshake with a peer, then HAVE messages for downloaded pieces
- **End game mode**
  - To avoid delays in obtaining the last blocks the protocol requests the last blocks from all peers
  - Sends cancel messages for downloaded blocks to avoid unnecessary transmissions
  - When to start the end game mode is not detailed in the specification
Tit-for-tat in Bittorrent

- Tit-for-tat is an effective strategy in game theory
  - Idea: cooperate first, and then respond in kind

- Peer has limited number of upload slots

- Upload bandwidth is exchanged for download bandwidth

- If peer is not uploading (only downloading) --> choke

- Upload slot to a random peer (optimistic unchoke)

- Searches for cooperative peers
TFT in more detail

1. Sort peers by incoming data rate
2. Reciprocate with top k, k is proportional to the square root of the upload capacity
3. Optimistically unchoke one other peer
4. Send each peer selected an equal split of capacity
Data transport in BitTorrent

Typically, BitTorrent uses **TCP** as its transport protocol for exchanging pieces, and it uses HTTP for tracker comms.

Possible to use HTTP port and real/fake HTTP headers for transport to avoid throttling (not in the specification)

The well known TCP port for BitTorrent traffic is 6881-6889 (and 6969 for the tracker port).

The DHT extension (peer-to-peer tracker) uses various UDP ports negotiated by the peers.

Web seeding (extension)
- Use HTTP to download pieces from Web sites

Security extensions (similar to TLS: message stream encryption)
NAT traversal

Open ports in firewall/NAT device

UPnP configuration

SSH tunnelling

HTTP tunnelling/proxying
  Any traffic through NATs
  Not necessarily efficient (with relay)
Distributed Tracker

BitTorrent Mainline DHT

Based on Kademlia DHT

Find peers through the DHT network

We will examine Kademlia later on this course
Altruism in BitTorrent

Seeders keep file available

A peer can choose to stay in the network and become a seeder, or leave

Upload activity is also example of altruistic behaviour
Biased neighbor selection

A technique called **biased neighbor selection** has been proposed for reducing cross-ISP traffic. A BitTorrent peer chooses most of its neighbors from the local ISP, and only a few peers from other ISPs. Essentially, the peer selection is biased towards local peers. A parameter $k$ represents the number of external peers from other ISPs. The tracker is modified to select $35 - k$ internal peers and $k$ external peers that are returned to the client requesting a peer list for a torrent. If there are less than $35 - k$ internal peers, the client is notified by the tracker to try again later.

The biased neighbour selection technique works well with the rarest first replication algorithm of BitTorrent; however, other piece selection algorithms, such as random selection, may not lead to optimal performance.
BitTorrent: Effects of Network Topology

Uniform random neighbor selection

Biased neighbor selection
Modelling BitTorrent

BitTorrent performance has been analyzed in the literature using analytical models, including stochastic and fluid models, extensive simulation experiments, experiments on distributed testbeds (PlanetLab), and by obtaining traces from real clients.

Both analytical and empirical evaluation and estimation are needed to dimension deployments to meet the service capacity demands.

Fluid models can be used to analytically estimate the protocol performance and understand the time evolution of the system by using differential equations.
Modelling aspects

- Dynamic population model
  - describing the evolution of the peer population in the P2P system
- Peer arrival process
  - steady arrival rate, smoothly attenuating arrival rate, or flash crowd?
- Efficiency of resource sharing
  - utilization of a peer’s upload capacity
  - effect of the piece/peer selection policy
  - number of parallel connections
- Selfishness / altruism
  - part of peers are free-riders that do not want to share upload capacity
- Download and upload rates
  - homogeneous or heterogeneous peer population?
- Number of permanent seeds
  - correspond to servers in the client-server architecture
Arrival processes

Various different arrival processes for new peers have been considered in the literature. The three key scenarios are as follows:

- The steady flow scenario used above assumes that new peers appear with a constant rate.
- The flash crowd scenario, considers the case where a (large) number of peers appear at the same time after which no new peers arrive.
- In a third scenario, the arrival rate is high in the beginning but smoothly attenuates as time passes.
Stochastic vs deterministic modelling

## Video-on-Demand

<table>
<thead>
<tr>
<th>Fixed-size Window</th>
<th>Piece 1</th>
<th>Piece 2</th>
<th>Piece ...</th>
<th>Piece w</th>
<th>Piece w+1..</th>
</tr>
</thead>
</table>

**Requested with probability p**

<table>
<thead>
<tr>
<th>BiToS</th>
<th>Piece 1</th>
<th>Arrived Piece</th>
<th>Piece 2</th>
<th>Piece w</th>
<th>Piece w+1..</th>
</tr>
</thead>
</table>

**BiToS**

w non-arrived pieces, probability p

Probability 1-p

w non-arrived pieces with absolute distance bounded (Bound b=2)

Outside stretching window

<table>
<thead>
<tr>
<th>Stretching Window</th>
<th>Piece 1</th>
<th>Arrived Piece 2</th>
<th>Piece 3</th>
<th>Piece ..w..</th>
</tr>
</thead>
</table>

Non-arrived: 1  Non-arrived: 1  Non-arrived: 2
Free-riding and tragedy of the commons

Users of P2P file sharing networks, such as Gnutella, face the question of whether or not to share resources to other peers in the community. They face essentially a social dilemma of balancing between common good and selfish goals. The selfish behaviour often encountered in P2P networks in which peers only download files and do not make resources available on the network is called free-riding. Free-riding occurs because the peers have no incentives for uploading files. Free-riding becomes a major problem when significant numbers of peers consume network resources while not contributing to the network. In the context of P2P this is often referred to as tragedy of the digital commons.
Preventing free-riding

BitTorrent has several mechanisms:
- Peer selection: tit-for-tat
- Optimistic unchoking
  - Two uses: find good peers and allow new peers to bootstrap

Other solutions have been proposed as well.
BitTyrant (NSDI 2007)

Observation: BitTorrent peers are altruistic

Incentives do not build robustness

A selfish BitTorrent client

Optimize return-on-investment (upload)
  Dynamically set the upload rate to maximize download rate

Can boost download speed by 70%
Building BitTyrant

Key idea: maximize return on investment (RoI)
   strategic peer selection
   strategic upload rate allocation

Cost: upload rate to peer p, $u_p$

Benefit: download rate from peer p, $d_p$

BitTyrant dynamically estimates these rates each tit-for-tat round

www.cs.utexas.edu/~yzhang/Teaching/cs386m-f10/Slides/3-2.ppt
Each TFT round, order and reciprocate with peers:

\[
\begin{align*}
&d_0, \quad d_1, \quad d_2, \quad d_3, \quad d_4 \\
&u_0, \quad u_1, \quad u_2, \quad u_3, \quad u_4, \quad \ldots
\end{align*}
\]

Choose \( k \) such that \( \sum_{i=0}^{k} u_i \leq cap \)

After each round, for each peer:

- If peer reciprocates:
  \[ d_p \leftarrow \text{direct observation} \]
  ...and continues to do so:
  Reduce \( u_p \)

- No reciprocation:
  Increase \( u_p \)

Unchokes

Does not unchoke
<table>
<thead>
<tr>
<th>BitTorrent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decentralization</strong></td>
</tr>
<tr>
<td><strong>Foundation</strong></td>
</tr>
<tr>
<td><strong>Routing function</strong></td>
</tr>
<tr>
<td><strong>Routing performance</strong></td>
</tr>
<tr>
<td><strong>Routing state</strong></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Decentralization</strong></td>
</tr>
<tr>
<td><strong>Foundation</strong></td>
</tr>
<tr>
<td><strong>Routing function</strong></td>
</tr>
<tr>
<td><strong>Routing performance</strong></td>
</tr>
<tr>
<td><strong>Routing state</strong></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
</tr>
</tbody>
</table>
Summary

We can summarize that unstructured P2P networks have favourable properties for a class of applications.

The applications need to be willing to accept best effort content discovery and exchange, and to host replicated content and then share the content with other peers.

The peers may come and go and the system state is transient (minimal assumptions on how long each peer participates in the network).

**Key point I**: data can be placed on an arbitrary node, typically no guarantees on finding the data.

**Key point II**: Structure and clustering is good.

The dominant operation in this class of applications is keyword-based searching for content.