HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Overlay and P2P Networks

Unstructured networks II

Prof. Sasu Tarkoma

26.9.2011

Contents

- Unstructured networks
  - Napster
  - Skype
  - Gnutella
  - Freenet
- Summary

Unstructured networks

Unstructured networks are typically based on random graphs following flat or hierarchical organization

Unstructured networks utilize flooding and similar opportunistic techniques, such as random walks, expanding-ring, Time-to-Live (TTL) search, in order to locate peers that have interesting data items.

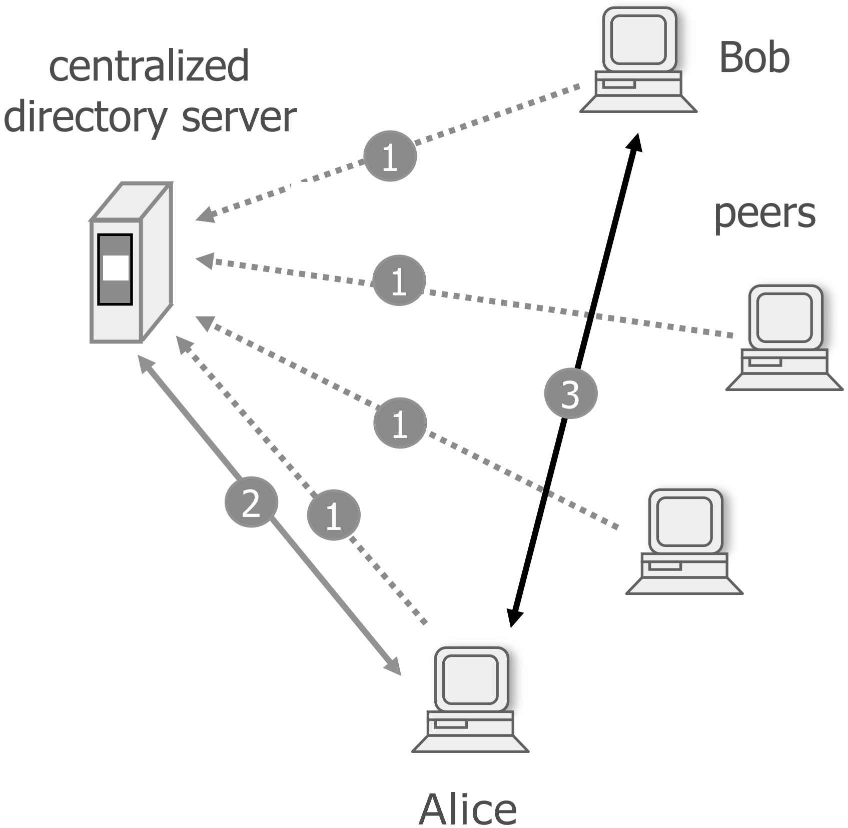
Napster

Napster was a centralized P2P music sharing service (mp3s)

Lauched in 1999 and made P2P popular and dubious from the legal viewpoint

Lawsuits from 1999, close-down in 2001, Chapter 7 in 2002, rebirth as a music store in 2003

Utilized a centralized index (server farm) for searching, transfers were peer-to-peer



User installing the software

- Download the client program
- Register name, password, local directory, etc.

1. Client contacts Napster (via TCP)
   - Provides a list of music files it will share
   - … and Napster's central server updates the directory
2. Client searches on a title or performer
   - Napster identifies online clients with the file
   - … and provides IP addresses
3. Client requests the file from the chosen supplier
   - Supplier transmits the file to the client
   - Both client and supplier report status to Napster

Napster Summary

Centralized server allows
- Consistent view of the P2P network
- Search guaranteed to find all files in the network

Limitations of this design are
- Centralized server is the weakest point of the system
  - Attacks, network partitions, …
- Limited scalability

Skype

Skype is a well-known Internet telephony service
- Calls between peers
- Interface to traditional telephony services (costs money)

Skype architecture is similar to KaZaa and Gnutella
- Supernodes and regular nodes

A proprietary protocol, protocol uses encryption

A centralized server for logging and billing

Supernodes and regular nodes maintain a distributed directory of online peers

Supernodes forward calls and call traffic (mostly for firewalled/natted peers)

A number of built-in techniques for traversing firewalls and NAT boxes
- STUN-like behaviour

What is NAT

Expand IP address space by deploying private address and translating them into publicly registered addresses

Private address space (RFC 1918)

10.0.0.0 - 10.255.255.255 (10.0.0.0/8)

172.16.0.0 - 172.31.255.255 (172.16.0.0/12)

192.168.0.0 - 192.168.255.255 (192.168.0.0/16)

First described in RFC 1631

Technique of rewriting IP addresses in headers and application data streams according to a defined policy

Based on traffic source and/or destination IP address

NATs and Firewalls

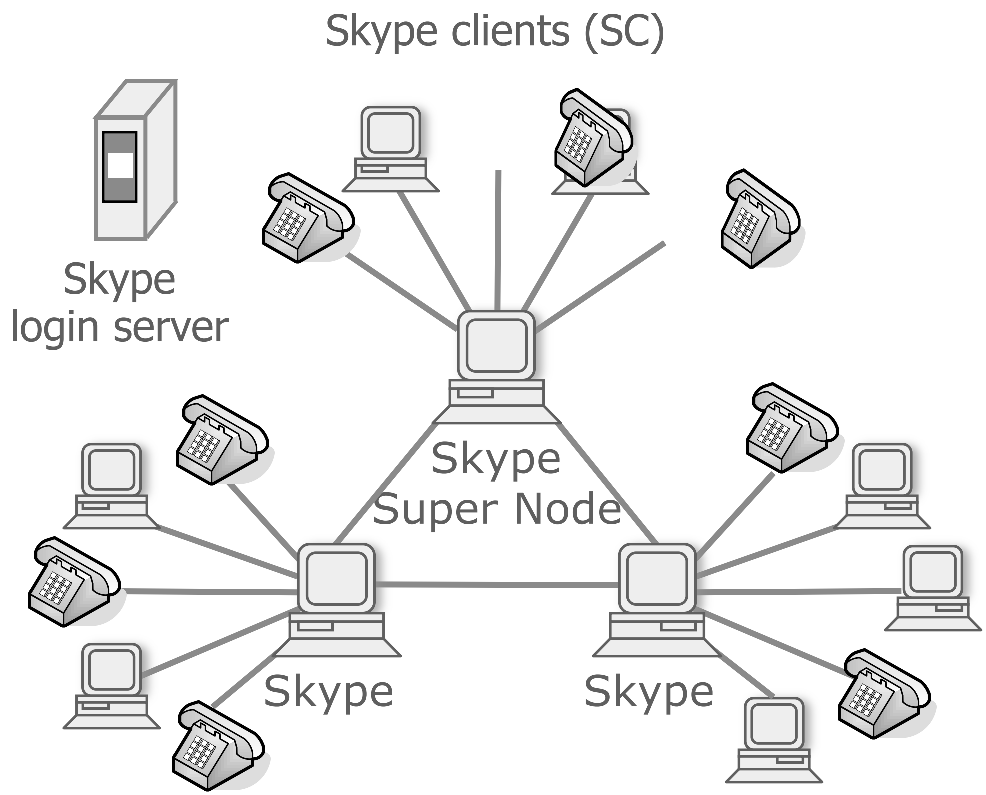
- Firewalls
  - Security main concern
  - Demilitarized zone
  - Increasingly complex rules (what is filtered, how)
- NATs
  - Lightweight security devices
    - Topology hiding and firewalling
  - Increasing number in deployment
    - Solves some of the address space problems of IPv4 (Port Translation, NAPT)
  - IPv6 solves the addressing problem so NATs are not needed for this

Skype

- Skype is P2P
- Proprietary application-layer protocol
- Hierarchical overlay with super nodes

- Index maps usernames to IP addresses; distributed over super nodes
- Peers with connectivity issues use NAT traversal or communicate via super node relays

- Encryption: RC4 and AES (data), public keys verified at login



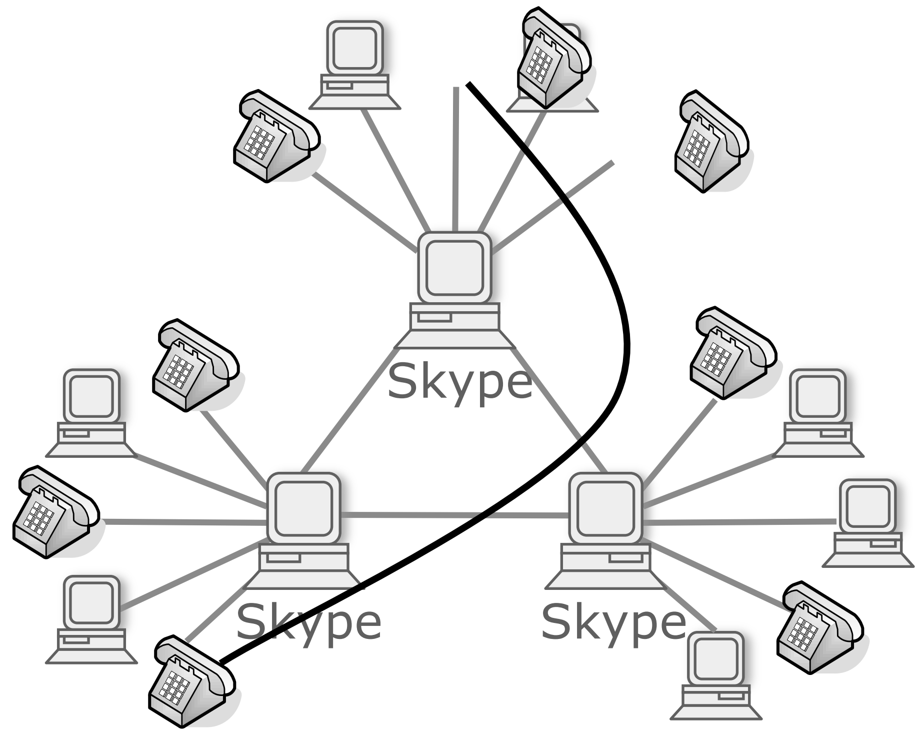
Skype peers as relays

Problem when both Alice and Bob are behind "NATs".

- NAT prevents an outside peer from initiating a call to insider peer

Solution:

- Using Alice's and Bob's SNs, Relay is chosen
- Each peer initiates session with relay.
- Peers can now communicate through NATs via relay



Skype Login Process

Comparison of three network setups

- Exp A: both Skype users with public IP address
- Exp B: one Skype user behind port-restricted NAT
- Exp C: Both Skype users behind port-restricted NAT and UDP-restricted firewall

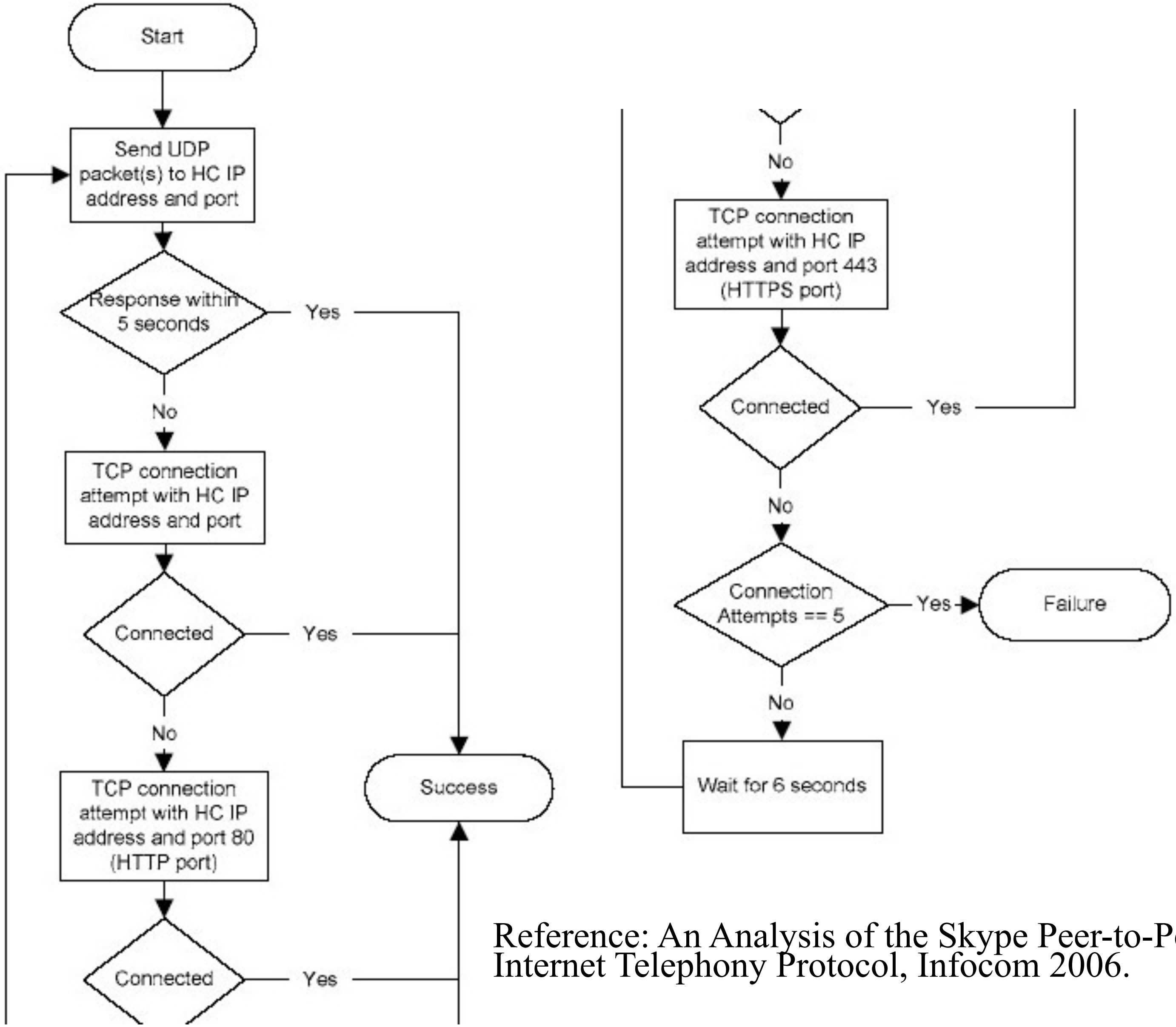
Message flows for first time login process

- Exp A and Exp B are similar
- Exp C only exchange data over TCP

	Total data exchanged	Login process time
Exp A	Approx 9 KB	3-7 secs
Exp B	Approx 10 KB	3-7 secs
Exp C	Approx 8.5 KB	Approx 34 secs

Reference: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Infocom 2006.

Login algorithm (HC is host cache)



Reference: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Infocom 2006.

Session keys

Skype client has a built in list of Skype servers

On each login session, Skype client generates a session key from 192 random bits

Skype client also generates a 1024-bit private/public RSA key pair.

Signed RSA public key is disseminated to Super Nodes

Skype clients can authenticate this information and then agree on a session key by using the RSA key

Blocking skype

Firewall rules

Skype traffic detection

http://www.tml.tkk.fi/Publications/C/23/papers/Santolalla_final.pdf

User Search

Skype uses the global index to search for a user

Skype claims that search is distributed and is guaranteed to find a user if it exists and has logged in during last 72 hours

Search results are observed to be cached at intermediate nodes

Establishing a Call: Three cases

Case 1: Public IP addresses. Caller establishes TCP connection with callee Skype client.

Case 2: Caller is behind port-restricted NAT, callee has public IP. Caller uses online Skype node to forward packets over TCP/UDP.

Case 3: Both caller and callee behind port-restricted NAT and UDP restricted firewall. Exchange info with a Skype node using TCP. Caller sends media over TCP to an online node which forwards to callee via TCP.

Conclusion on Skype

Successful overlay technology

Call forwarding with self-organizing network of nodes

Gnutella

Gnutella addresses some of Napster's limitations

A decentralized P2P system based flooding the queries
- Queries are flooded and responses are sent on the reverse path
- Downloads directly between peers

Open protocol specification, originally developed by Nullsoft (bought by AOL)

Differs between versions
- 0.4 is the original version (simple flooding)
- 0.7 is more advanced (similar to KaZaa)
  - More structure (hierarchy is good for scalability!)

Gnutella v0.4 protocol messages

A peer joining the network needs to discover the adress of a peer who is already a member of the network

New peer sends GNUTELLA CONNECT message

A peer then uses PING messages to discover peers and receives PONG messages. PONGs include data regarding peers and follow the reverse path of PINGs.

A peer uses the QUERY message to find files, and receives QUERYHIT messages as replies (again on reverse path)

Peers forward QUERY messages (flooding)

The QUERYHIT contains the IP address of the node that can then be used for the file transfer (HTTP)

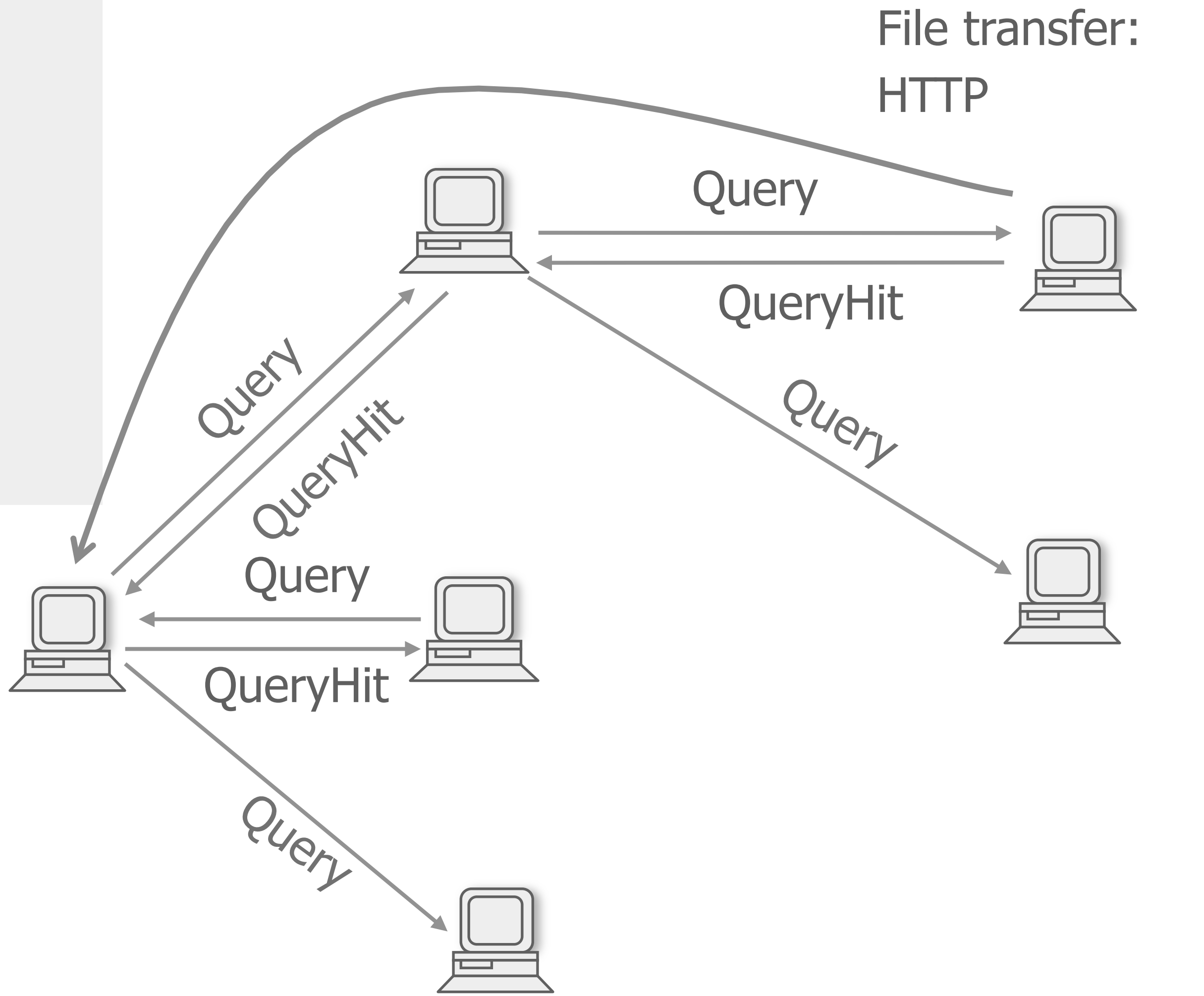
The Gnutella Protocol

Query message sent
- over existing TCP connections

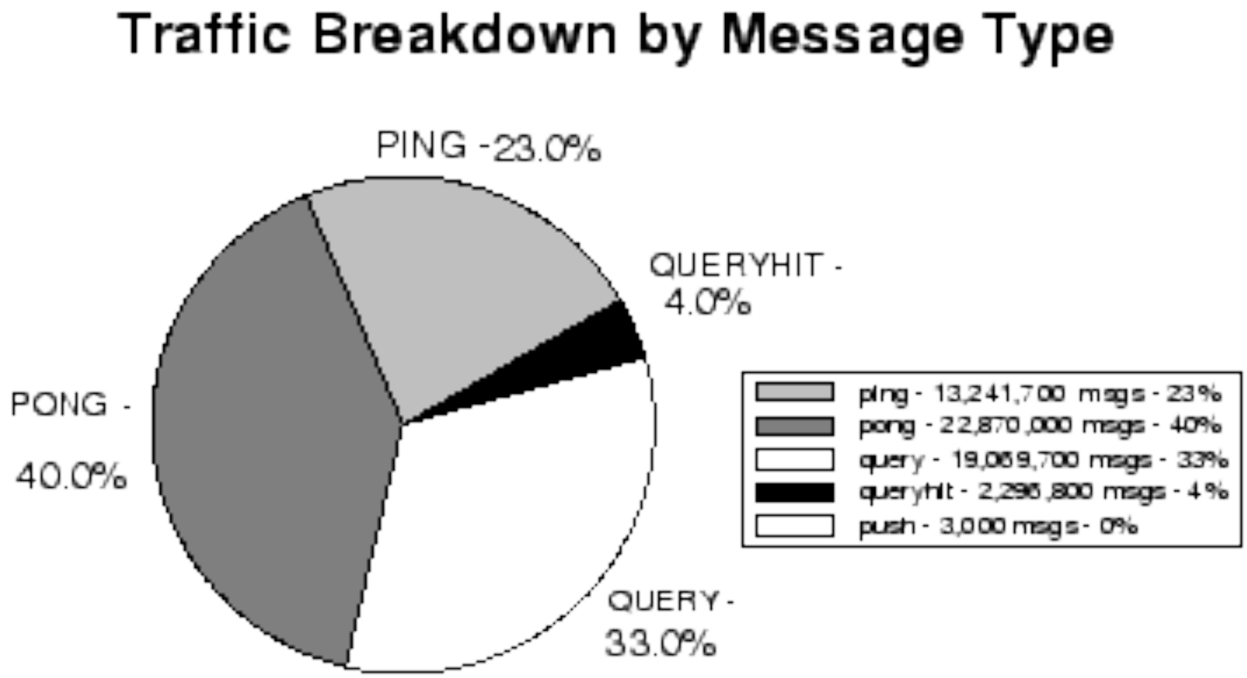
Peers forward
- Query message

QueryHit sent over
- reverse path

Scalability: limited scope flooding



Traffic breakdown



From "A Quantitative Analysis of the Gnutella Network Traffic"

Trees vs graphs

Tree

N nodes, N-1 links

Network with N hosts and M connections, M >= N-1 then (M – (N -1)) loop connections

These make the network more robust, but increase communications overhead

Loops result in infinite message loops (unless specific loop prevention measures are implemented)

Looping and message processing

Gnutella network is based on a cyclic graph

Loops are problematic

Two key solutions:

1. TTL (Time-To-Live): reduces flooding (7 by default)
2. Duplicate detection with unique request identifier

Gnutella uses both (v0.7 is not using flooding anymore so the problem is alleviated)

Even with duplicate detection cannot prevent receiving the same message many times (but can prevent propagation)

Request messages

Each peer keeps track of all messages it has seen

Can forget about that after some time period

Remember who first sent you a message

If a second copy or subsequent copy of a message arrives, ignore it

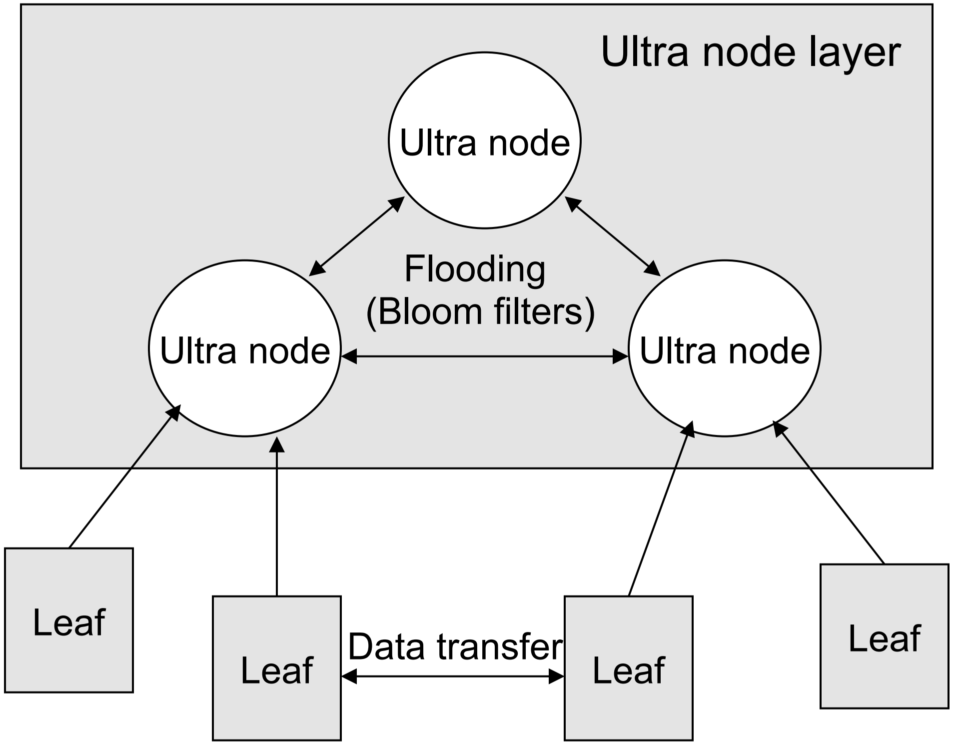
Response messages

Use the same GUID as the message they are in response to

Each peer routes a response msg to the peer from which it first received the original msg

Drop message if did not see original

The Gnutella v0.7 Architecture



Gnutella v0.7 routing

Since version 0.6, Gnutella has been a composite network consisting of leaf nodes and **ultra nodes**. The leaf nodes have a small number of connections to ultra nodes, typically three

The ultra nodes are hubs of connectivity, each being connected to more than 32 other ultra nodes.

When a node with enough processing power joins the network, it becomes an ultra peer and establishes connections with other ultra nodes

This network between the ultra nodes is flat and unstructured. Then the ultra node must establish a minimum number of connections with client nodes in order to continue acting as an ultra node. These changes attempt to make the Gnutella network reflect the **power-law distributions** found in many natural systems.

Query Routing Protocol

In Gnutella terminology, the leaf nodes and ultra nodes use the **Query Routing Protocol** to update routing tables, called **Query Routing Table (QRT)**

The QRT consists of a table **hashed keywords** that is sent by a leaf node to its ultra nodes

Ultra nodes merge the available QRT structures that they have received from the leaf nodes, and exchange these merged tables with their neighbouring ultra nodes

Query routing is performed by hashing the search words and then testing whether or not the resulting hash value is present in the QRT of the present node

The classical Gnutella protocol used reverse path routing to send a message back to this origin peer. Later incarnations of the protocol use UDP to directly contact the origin peer

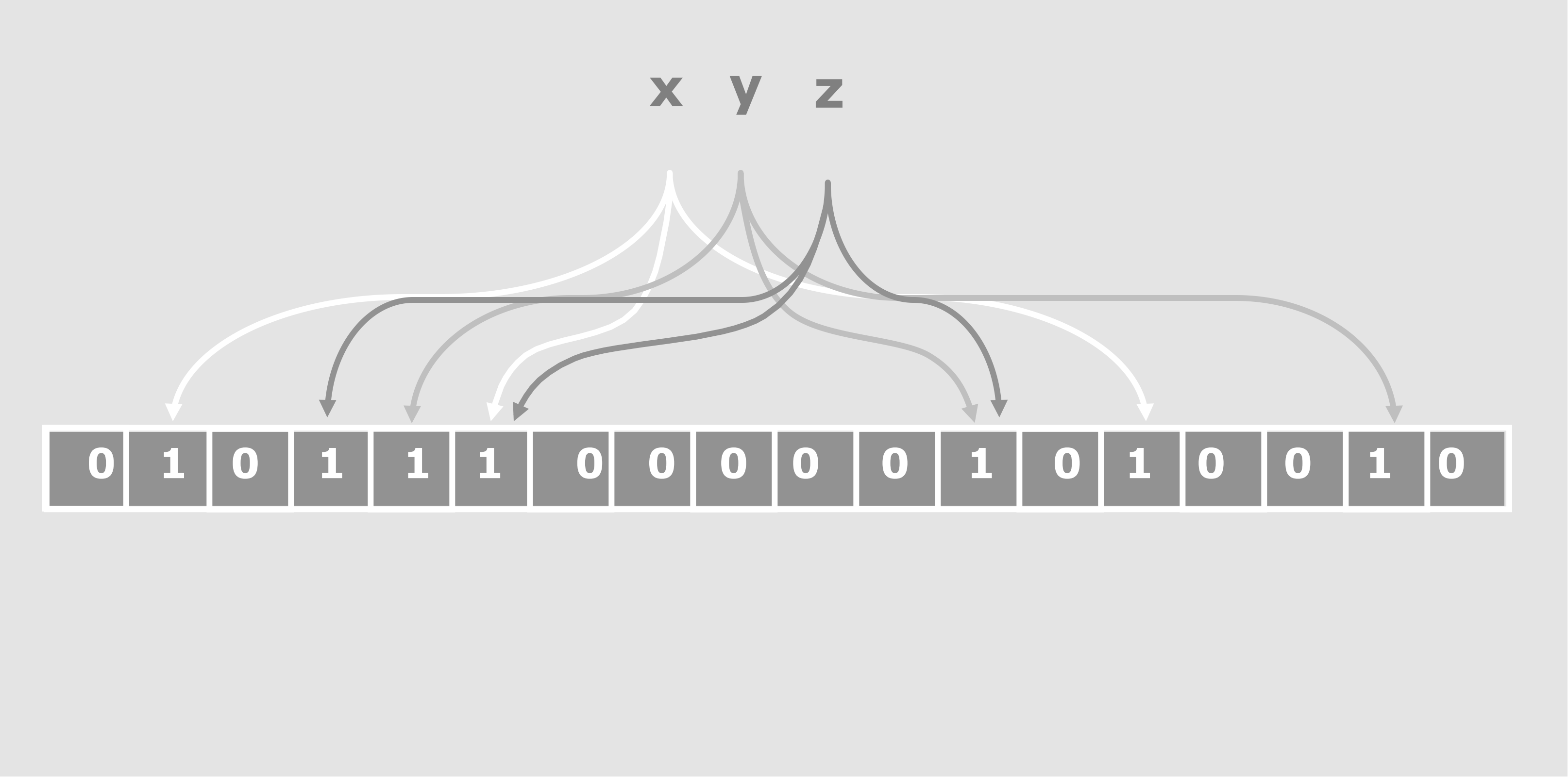
Bloom filters in Gnutella v0.7

Bloom filters are probabilistic structures used to store dictionaries

A bit-vector that supports constant time querying of keywords

	Decrease	Increase
Number of hash functions (k)	Less computation Higher false positive rate	More computation Lower false positive rate
Size of filter (m)	Smaller space requirements Higher false positive rate	More space is needed Lower false positive rate
Number of elements in the inserted set (n)	Lower false positive rate	Higher false positive rate

Example Bloom filter



Data: x is the object key to insert into the Bloom filter.
Function: *insert*(x)

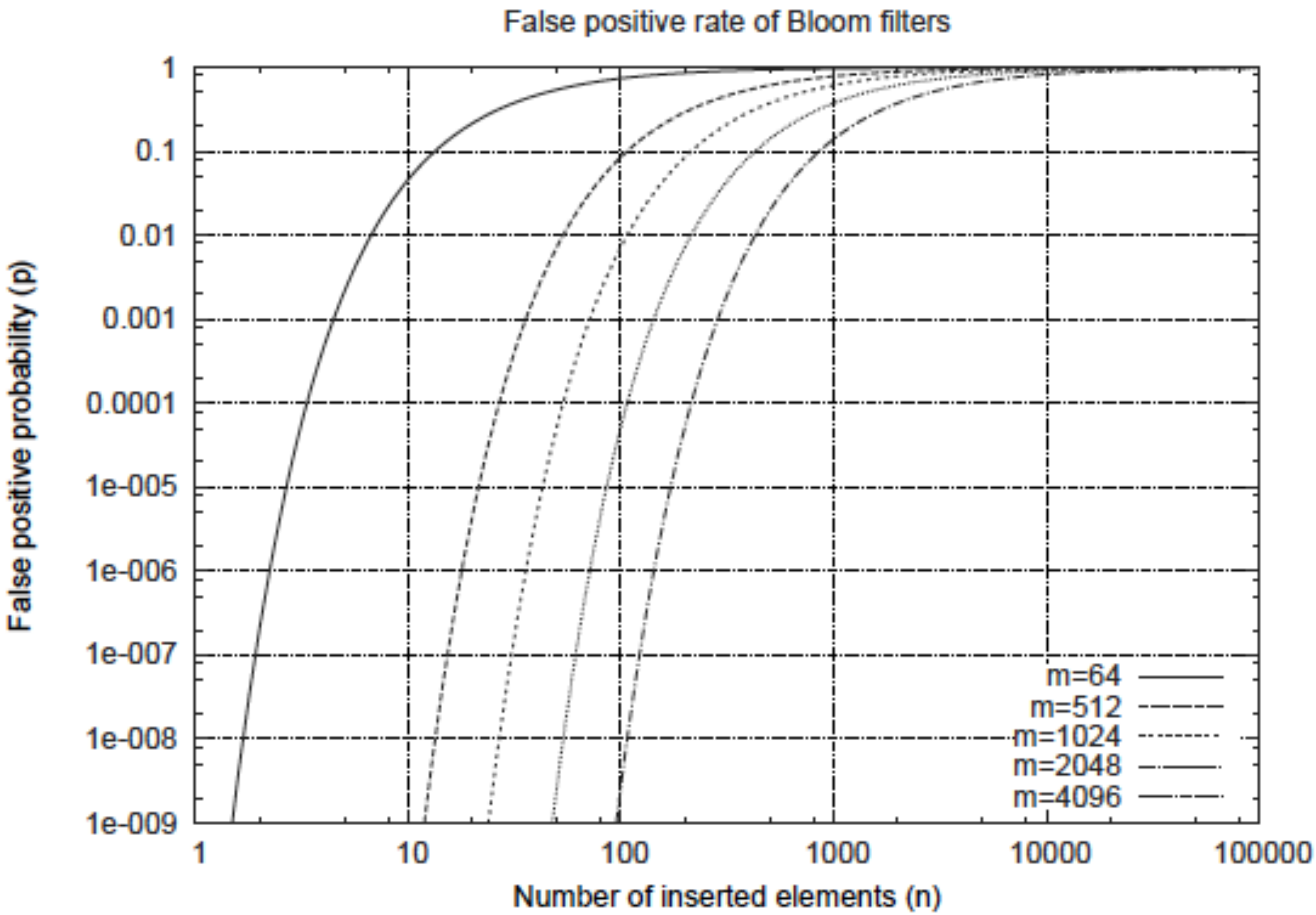
```
for j : 1 ... k do
    /* Loop all hash functions k                         */
    i ← h_j(x);
    if B_i == 0 then
        /* Bloom filter had zero bit at
        position i                                       */
        B_i ← 1;
    end
end
```

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

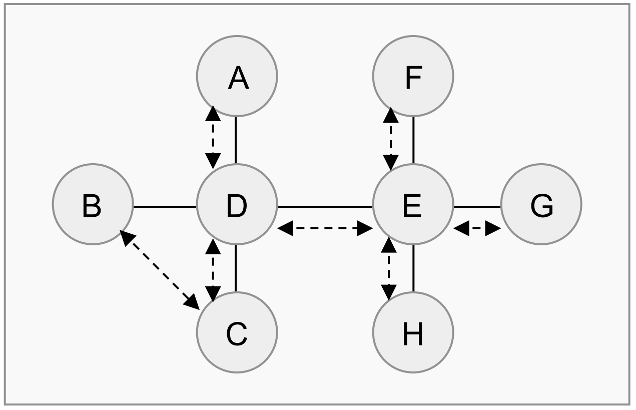
```
m ← 1;
j ← 1;
while m == 1 and j ≤ k do
    i ← h_j(x);
    if B_i == 0 then
        m ← 0;
    end
    j ← j + 1;
end
return m;
```

Algorithm 2: Pseudocode for Bloom member test

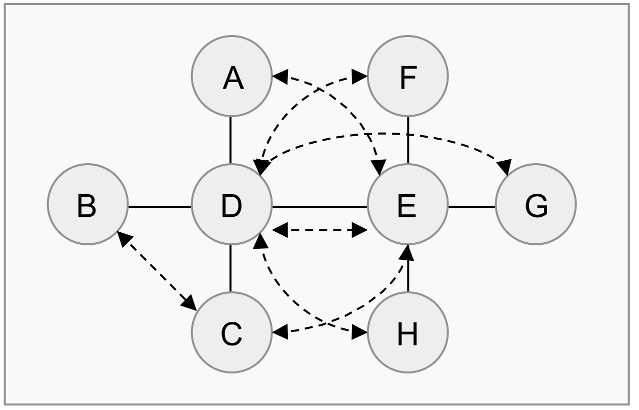


False positive rate of Bloom filters
False positive probability (p)
1
0.1
0.01
0.001
0.0001
1e-005
1e-006
1e-007
1e-008
1e-009
1
10
100
1000
10000
100000
Number of inserted elements (n)
m=64
m=512
m=1024
m=2048
m=4096

Mapping the Gnutella Network



Perfect mapping for message from A.
Link D-E is traversed only once.



Inefficient mapping that results in link D-E
being traversed six times

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

	Gnutella v0.4	Gnutella v0.7
Decentralization	Flat topology (random graph), equal peers	Random graph with two tiers. Two kinds of nodes, regular and ulta nodes. Ultra nodes are connectivity hubs
Foundation	Flooding mechanism	Selective flooding using the super nodes
Routing function	Flooding mechanism	Selective flooding mechanism
Routing performance	Search until Time-To-Live expires, no guarantee to locate data	Search until Time-To-Live expires, second tier improves efficiency, no guarantee to locate data
Routing state	Constant	Constant
Reliability	Performance degrades when the number of peer grows	Performance degrades when the number of peer grows

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:

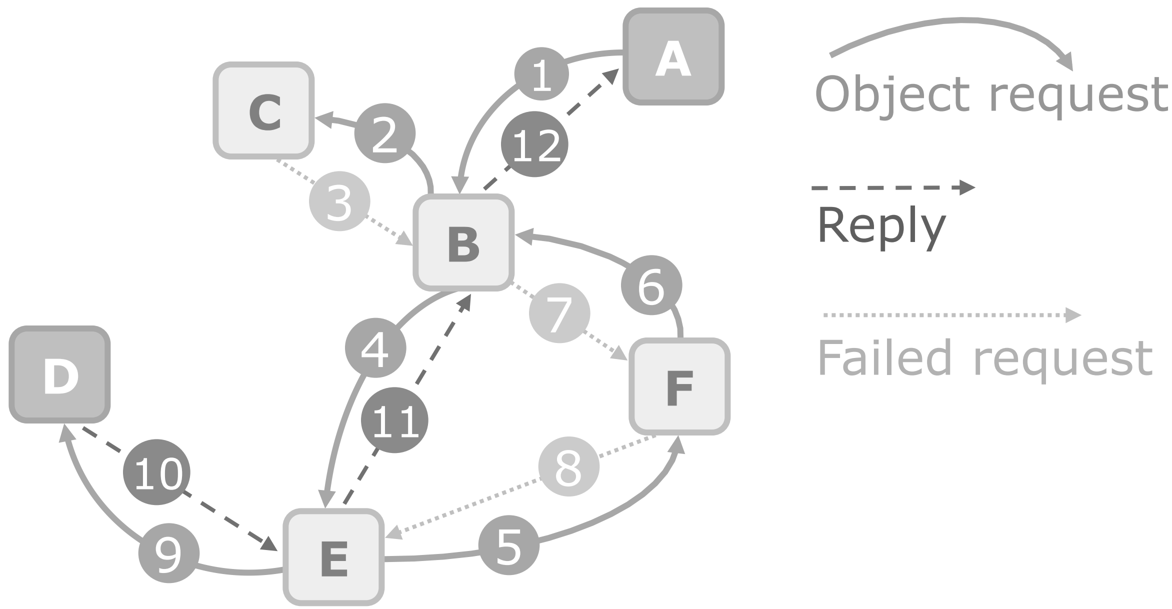
- **Data insert.** This message allows a node to insert new data into the network. The message includes a key and the data file.
- **Data request.** A node requests for a certain file. The request contains the key of the file.
- **A reply.** The reply is sent by the node that has the requested file. 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

Routing table:

id	next_hop	file
	⋮	
	⋮	



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 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 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).

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

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

Freenet routing properties

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

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

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)

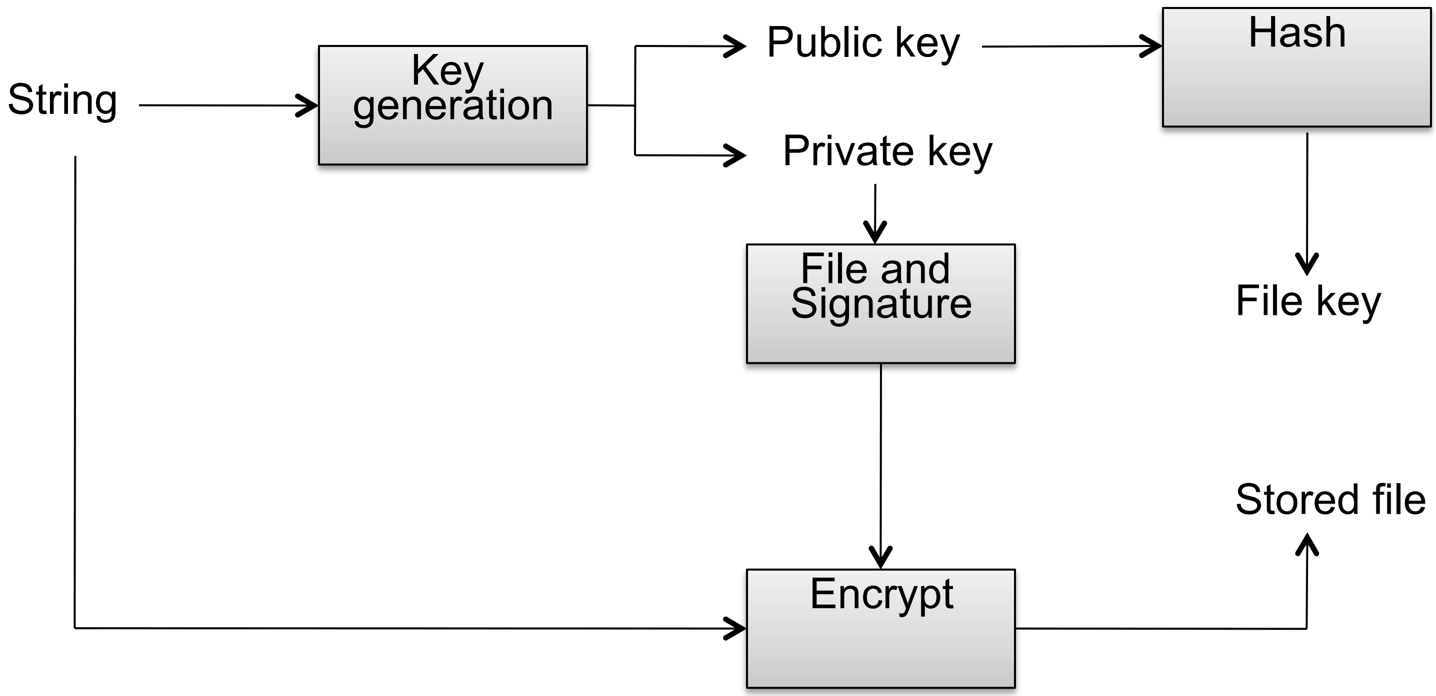
SSKs (Signed Subspace Key) are intended for sites with mutable data. A typical usage case involves a Web site. Hash of a public key, symmetric key (hash of the description), signature. Defines a personal namespace that anyone can read but can be written only with the private key.

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

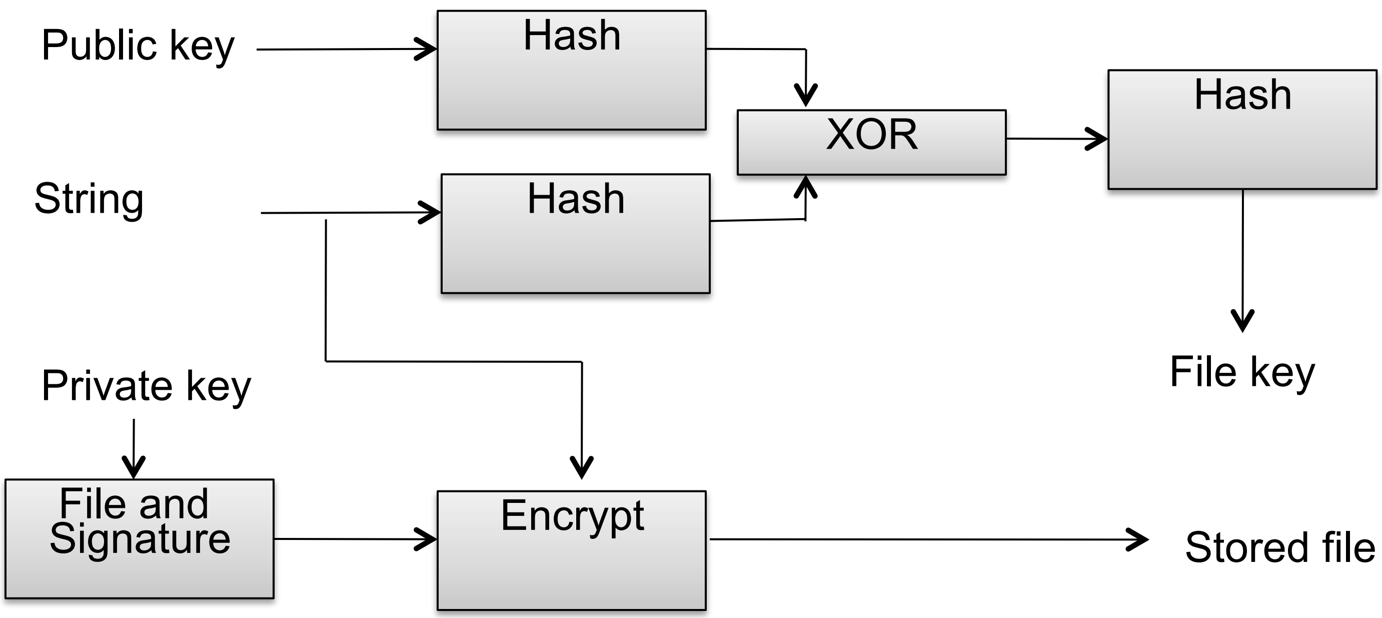
KSK example (retrieval using strings)



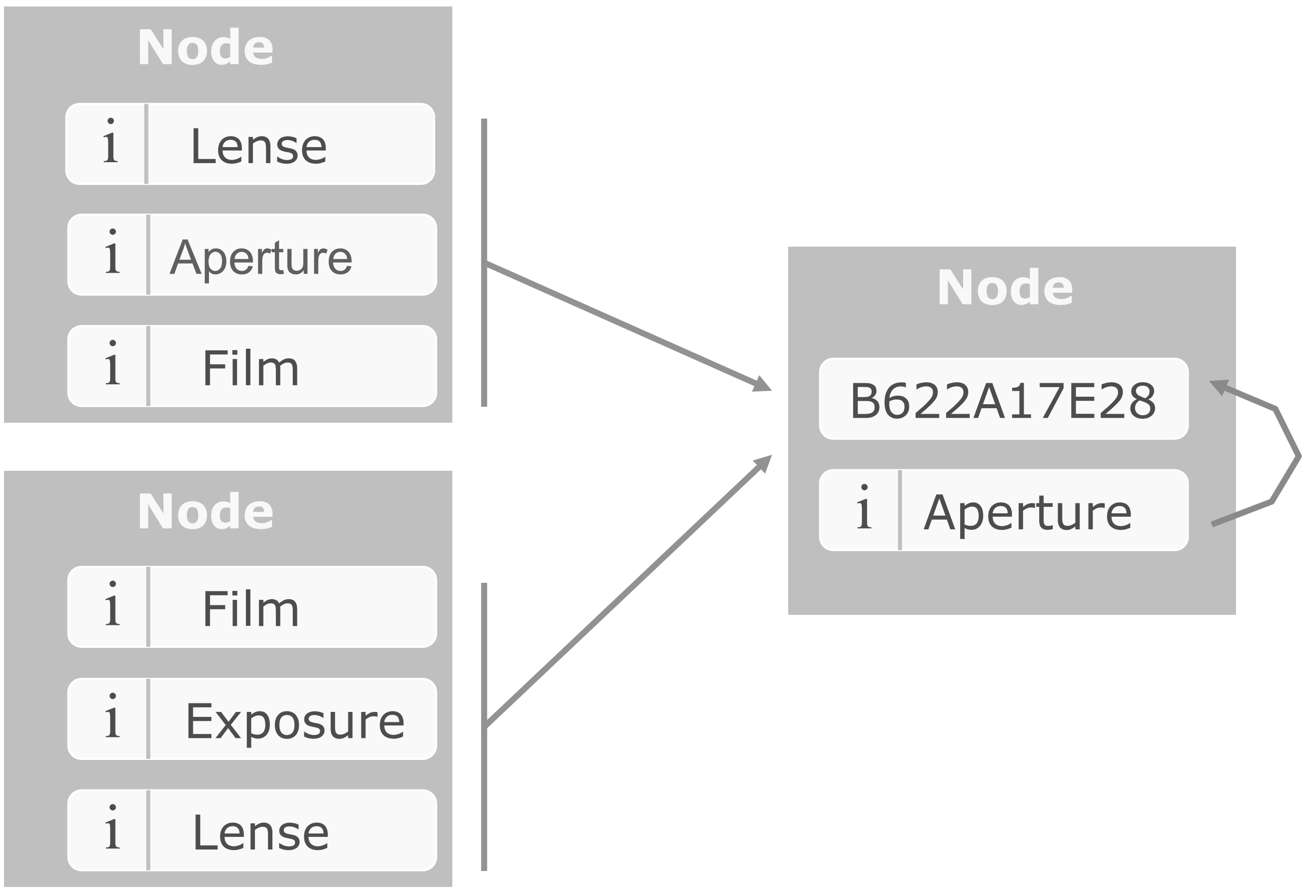
Example of KSK Usage

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 results 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)



	Freenet v0.7
Decentralization	Similar to DHTs, two modes (darknet and opennet), two tiers
Foundation	Keywords and text strings are used to identify data objects. Assumes small world structure for efficiency
Routing function	Clustering using node location and file identifier. Searches from peer to peer using text string. Path folding optimization
Routing performance	Search based on Hop-To-Live, no guarantee to locate data. With small world property $O(\log(n)^2)$ hops are required, where n is the number of nodes.
Routing state	With small world property $O(\log(n)^2)$
Reliability	No central point of failure

	BitTorrent	Freenet v0.7	Gnutella v0.4	Gnutella v0.7
Decentralization	Centralized model	Similar to DHTs, two modes (darknet and opennet), two tiers	Flat topology (random graph), equal peers	Random graph with two tiers. Two kinds of nodes, regular and ulta nodes. Ultra nodes are connectivity hubs
Foundation	Tracker	Keywords and text strings are used to identify data objects. Assumes small world structure for efficiency	Flooding mechanism	Selective flooding using the super nodes
Routing function	Tracker	Clustering using node location and file identifier. Searches from peer to peer using text string. Path folding optimization	Flooding mechanism	Selective flooding mechanism
Routing performance	Guarantee to locate data, good performance for popular data	Search based on Hop-To-Live, no guarantee to locate data. With small world property $O(\log(n)^2)$ hops are required, where n is the number of nodes.	Search until Time-To-Live expires, no guarantee to locate data	Search until Time-To-Live expires, second tier improves efficiency, no guarantee to locate data
Routing state	Constant, choking may occur	With small world property $O(\log(n)^2)$	Constant	Constant
Reliability	Tracker keeps track of the peers and pieces	No central point of failure	Performance degrades when the number of peer grows	Performance degrades when the number of peer grows

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: data can be placed on an arbitrary node, typically no guarantees on finding the data

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