

# Routing in a distributed anonymous P2P system - Freenet

Farhad Mehta  
(mehta@in.tum.de)

Seminar Internet Routing,  
Technische Universität München

SS 2003 (Version 2)

## Abstract

Freenet proposes a novel way of routing information through a peer to peer network. This paper describes the Freenet system, with a specific focus on the peer to peer routing techniques employed by it to facilitate its major design goals, total distributivity, anonymity, and deniability.

## 1 Introduction

Over the years, the Internet has been invaluable in its function as an information storage and retrieval system. The last couple of years have seen a tendency to change its organisation from a server-client based model, where a sharp divide is made between information providers and requesting hosts, to a peer to peer model, where both communicate as equals. Since the Internet, along with its facilitating protocols are built around a server-client model, it proved a nontrivial exercise to extend it to support peer to peer interaction.

One of the major non-technical reasons for the shift to peer to peer networks was to have free information flow. In a server-client model, this was not possible as the server had total control of what information it was delivering. Early attempts at P2P networks proved that freedom to publish, without anonymity, was not sufficient to guarantee free speech. Additionally, the slightest centralised network function opened the door for centralised information control. For a P2P system to be potentially used to store sensitive information, one has to guarantee total anonymity for users and data, along with complete decentralisation. In addition, the network should have efficient routing characteristics, a somewhat opposing goal.

The Freenet project aims to provide such a P2P information storage and retrieval system. Its motto: Information wants to be free.

Section two formally states the design goals of Freenet project. Section three gives an overview of the Freenet system. Section four goes over how basic network functions such as data insertion and retrieval work. Section five shows how the Freenet protocol fares when simulated for routing

performance. Section six concludes the paper with a look at how far the protocol manages to meet its design goals.

## 2 Design goals

Formally laid out, the goals of Freenet are:

- Anonymity for information producers and consumers
- Deniability for storers of information
- Resistance to attempts by third parties to deny access to information
- Decentralisation of all network functions
- Efficient dynamic storage and routing of information

For the end user, the system should serve as a distributed network file system. The files as such have no permanent home and float around in the network, being replicated near requesting hosts, and deleted in areas of no interest. The lifetime of a file on the network is not guaranteed, but it is hoped that enough nodes contribute enough space for most files to remain in the system indefinitely.

## 3 System overview

Each user of the Freenet system runs a node. These nodes form an adaptive peer to peer network and query each other to store and access data. Files on the network are indexed using location independent keys. Each node maintains a data-store (i.e. a set of files indexed using keys) and a dynamic routing table containing other known nodes, along with the keys of files they are known to hold. Files on the network are lazily replicated on demand.

Requests for keys are relayed from node to node through a chain of proxy requests in which each node makes a local decision about where to send the request next. Routes will vary on the key requested. The routing mechanisms for data storage and retrieval are designed to adaptively adjust routes over time to provide efficient performance while using only local knowledge. To maintain privacy nodes only have knowledge of their immediate upstream and downstream neighbours.

In order to prevent routing loops, each request is given a *hops-to-live* value which is decremented at each routing step. Each request is also assigned a pseudo-unique random identifier. Nodes reject requests they have seen before.

No hierarchy or central point of failure exists, since no node is privileged over any other node. Nodes join the network by first discovering the address of one or more existing nodes through external means, then start sending messages.

The binary file keys used to index files on the network are obtained using a hash function. Freenet uses three types of file keys. Since their details do not aid in the discussion on routing, only the simplest of the the three file key mechanisms will be outlined here. The simplest type of file key is the keyword-signed key (KSK), which is derived from a short descriptive text string chosen by the user when storing a file in the network.

This string is used as input to deterministically generate a public/private key pair. The public half is then hashed to yield the file key.

The private half of the asymmetric key pair is used to sign the file, providing a minimal integrity check that a retrieved file matches its file key. The file is also encrypted using the descriptive string itself as a key.

To allow others to retrieve the file, the user need only publish the descriptive string. With the descriptive string in hand, the requestor generates the identical public/private key pair used by the publisher, sends the request for the file using the hash of the public key. When the file arrives, he can use descriptive string to decrypt the file, and the the public half of the asymmetric key to check its integrity.

## 4 Network functions

### 4.1 Data retrieval

A normal sequence of events for data retrieval:

1. User calculates the key of the file to be requested.
2. He then sends a request message to the local node, specifying the key, and a *hops-to-live* value.
3. The local node checks local store for the requested key.
4. If found, it returns file and mentions that it was the source.
5. If not found, the node then looks up its routing table for the next node with key nearest<sup>1</sup> to the one being requested, and forwards the request.
6. If the request is ultimately successful and returns with the requested data, the node will:
  - (a) Pass the data back to the upstream requestor (or user).
  - (b) Cache the data in its own data-store.
  - (c) Create a new entry in its routing table, associating the actual data source with the requested key.

A subsequent request for the same key will be satisfied from the local cache, but the search for a similar key will be forwarded to the last successful data source. For security concerns, any node along the request path is free to modify the reply message to claim itself, or another arbitrarily chosen node as the data source.

7. In case the search fails, or if a node cannot forward a request to its preferred downstream node because the target is down or a loop would be created, the node having the next nearest key will be tried, until it runs out of choices, upon which it would report the failure to the upstream node.
8. If the *hops-to-live* limit is exceeded, a failure result is propagated back to the original requestor without any further nodes being tried.

Figure 1 depicts a typical sequence of request messages used in [Jan\_01] to illustrate the file request mechanism. The user initiates a request at

---

<sup>1</sup>Nearness with respect to keys can be measured with respect to an arbitrary distance metric. Note that nearness in key distances is unrelated to any notion of closeness in the document space.

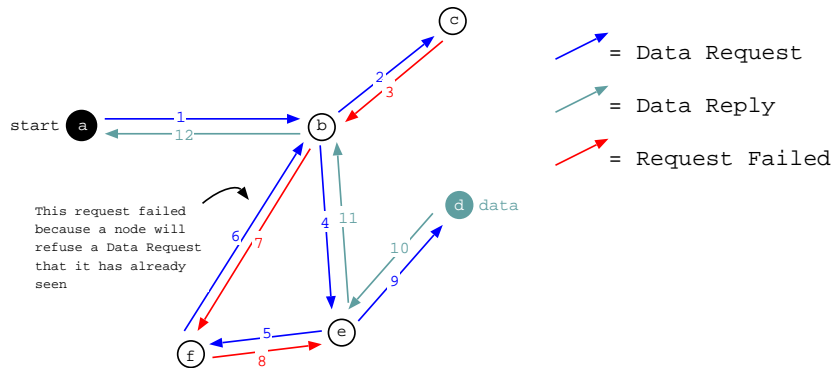


Figure 1: A typical request scenario.

node a. Node a forwards the request to node b, which forwards it to node c. Node c is unable to contact any other nodes and returns a backtracking “request failed” message to b. Node b then tries its second choice, e, which forwards the request to f. Node f forwards the request to b, which detects the loop and returns a backtracking failure message. Node f is unable to contact any other nodes and backtracks one step further back to e. Node e forwards the request to its second choice, d, which has the data. The data is returned from d via e and b back to a, which sends it back to the user. The data is also cached on e, b, and a

This request mechanism has a number of desirable effects:

1. The quality of routing improves over time as:
  - (a) Nodes become more and more specialised in locating similar sets of keys.
  - (b) Nodes become similarly specialised in storing clusters of files with similar keys.

These two events should improve the efficiency of future requests in a self-reinforcing cycle.

2. Popular data is transparently replicated by the system and mirrored closer to requesting hosts, providing increased efficiency and data redundancy.
3. Nodes build up their routing tables as they process requests, increasing connectivity.

## 4.2 Data storage

In order to insert a file, the user calculates the key for it and forwards it, with a *hops-to-live* value, to the local node. The *hops-to-live* roughly determines the number of nodes to store the file on. Similar to a data request, the local node checks its data-store for the key, and if not found, forwards the query to the next node with similar key. At the end of the query cycle, the inserting host either gets an all-clear message (analogous

to a failure in the case of information retrieval) or a response with a copy of the file with the conflicting key. In the case of a conflicting key, the user must try again with a different key. In the case of an all-clear, the inserting host is then allowed to send the appropriate data, which is propagated along the initial query path, and stored in each node on this path. Each of these nodes will also create an entry in their routing tables associating the inserting host with the new key. Any node can chose to alter the insert message to claim itself, or any arbitrary node to be the data source in order to maintain anonymity of the actual data source.

This mechanism has the following desirable effects:

1. Newly inserted files are placed on nodes already possessing files with similar keys, reinforcing their specialisation in that key cluster.
2. New nodes can use inserts to announce their existence to the rest of the network.
3. Malicious attempts to replace existing files by inserting dummy files under existing keys are likely to spread the real files further, as they are propagated on collision.

### 4.3 Data management

To deal with the limitations of finite storage, Freenet uses a LRU (least recently used) policy to manage data-stores and routing tables at each node. When a new file arrives which would cause the data-store to exceed the designated size, the least recently used files are deleted in order until there is room. The resulting impact on availability is toned down by the fact that the routing table entries created for these deleted files will remain for a longer time, potentially allowing the node to later get new copies from the original data sources. Since routing table entries are smaller than the files themselves, it is very probable that they will outlive their corresponding files.

This does mean that Freenet cannot guarantee the duration of time a file exists on the network. Files exist on the network only as long as they are requested for. This has the advantage that outdated or unneeded files gracefully leave the network without the arbitration of a single entity that judges their outdatedness or insignificance.

Since files are encrypted, node operators have no way of explicitly managing the content stored at their node's data-store.

### 4.4 New nodes

Although the request and insert mechanisms allow a new node to discover more of the network over time, in order for the rest of the network to discover them, they must somehow announce their presence. The announcement would add an entry corresponding to the new node, along with a key. The new node can now be sent requests, and takes an active part in the network.

In order for effective routing to take place, all existing nodes must be consistent in deciding which keys to send the new node. Given the security and peer to peer constraints of Freenet, this is not a trivial task.

Freenet uses a cryptographic protocol to reach consensus over a key for this new node. Details of this protocol can be found in [Ian\_01].

## 5 Routing performance

This section contains a summary of some of the results obtained by one of the authors [Hong\_01] while simulating the Freenet peer to peer protocol. Simulations were made to test network convergence, scalability and fault tolerance. The authors have tried to explain the validity of their results using the small-world network model, which is also summarised.

### 5.1 Network Convergence

Network convergence is a measure of how much time a routing network takes to identify and use optimal paths between any two individual nodes.

- *Setting:* The simulation consisted of a test network of 1000 nodes. Each node had a data-store size of 50 items and a routing table size of 250 addresses. All data-stores were initialised to be empty. All routing tables were initialised to connect the network in a regular ring-lattice topology. Each node had routing entries for its two nearest neighbours on either side. The keys associated with these routing entries were set to be hashes of the destination nodes' addresses.

Inserts of random keys were sent to random nodes in the network, interspersed randomly with requests for randomly-chosen keys known to have been previously inserted, using a *hops-to-live* of 20 for both. Every 100 time-steps, a snapshot of the network was taken and its performance measured using a set of probe requests. Each probe consisted of 300 random requests for previously-inserted keys, using a *hops-to-live* of 500. The resulting distribution of request path lengths was recorded, the number of hops actually taken before finding the data. If the request did not find the data, the path length was taken to be 500.

- *Results:* Figure 2 shows the evolution of the first, second, and third quartiles<sup>2</sup> of the request path length over time, averaged over ten trials. Initially high path lengths decrease rapidly over time. As nodes know more about the rest of the network, using requests to fill up their routing tables, routing is seen to converge, resulting in a rapid decrease in path lengths.

### 5.2 Scalability

Scalability is a measure of how the network copes with an increase in the number of individual nodes.

- *Setting:* The network starts at twenty nodes, similarly initialised as in the case for network convergence. New nodes are added to this network and the change in request path length is measured over time. Inserts and requests are simulated randomly as before. Every five time steps, a new node is created and added to the network by simulating a node announcement message with *hops-to-live* of 10 sent from it to a randomly-chosen existing node.
- *Results:*

---

<sup>2</sup>first quartile = 25<sup>th</sup> percentile ; second quartile = 50<sup>th</sup> percentile = median ; second quartile = 75<sup>th</sup> percentile.

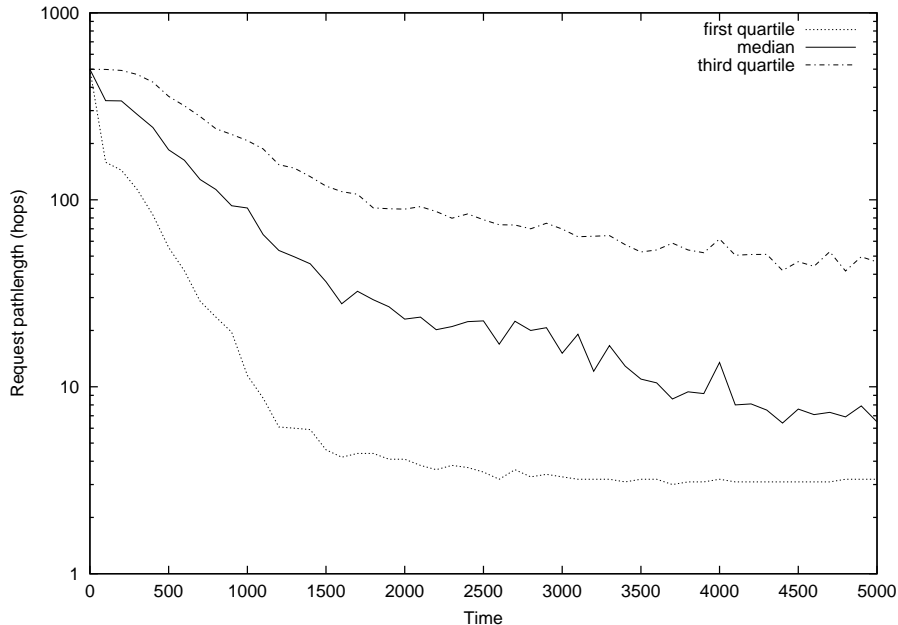


Figure 2: Decrease in request path length over time.

Figure 3 shows the evolution of the first, second, and third quartiles of the request pathlength versus network size, averaged over ten trials. The authors claim that pathlength scales approximately logarithmically, with a slope change near 40,000 nodes. They explain this slope change as a result of routing tables becoming filled and overflowing, resulting in subsequent loss of routing information. The simulated network appears capable of scaling to one million nodes with a median pathlength of 30.

### 5.3 Fault tolerance.

Fault tolerance is a measure of how well a routing network is able to cope in the presence of node failure.

- *Setting:* Node failure is simulated by removing randomly selected nodes from a 1000 node network.
- *Results:*

Figure 4 shows the resulting evolution of the request pathlength, averaged over ten trials. The simulated network seems robust in spite of quite large failures. Even with 30 percent node failure, the median pathlength remains below 20.

In [Ian\_01], the authors attempt to explain the stability and fault tolerance results on the Freenet network, using the small-world network model [Milg\_67][Wat\_67]. In a small-world network, the majority of nodes have only relatively few, local, connections to other nodes, while a small number of nodes have large, wide-ranging sets of connections.

Small-world networks permit efficient short paths between arbitrary points because of the shortcuts provided by the well-connected nodes.

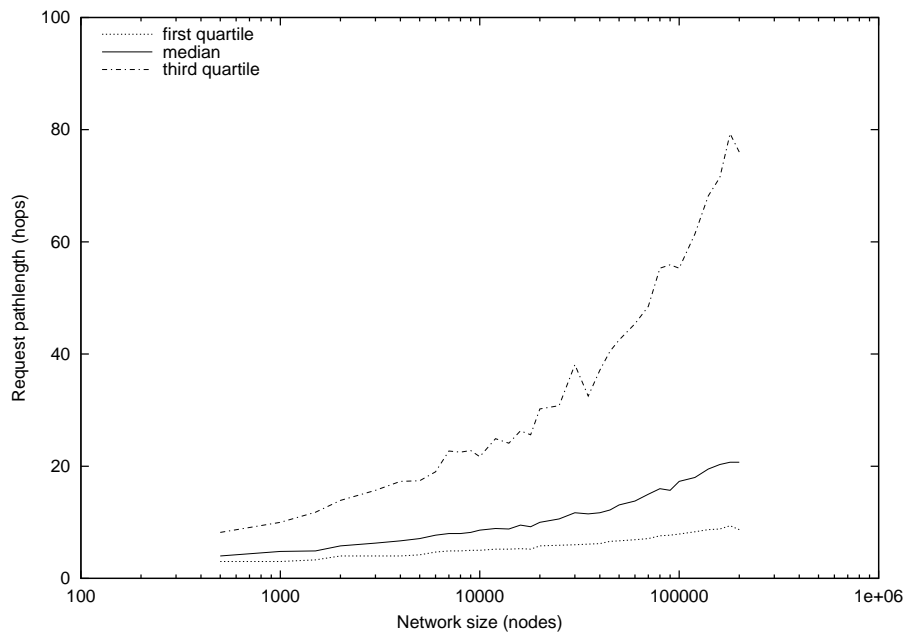


Figure 3: Increase in request path length with network size.

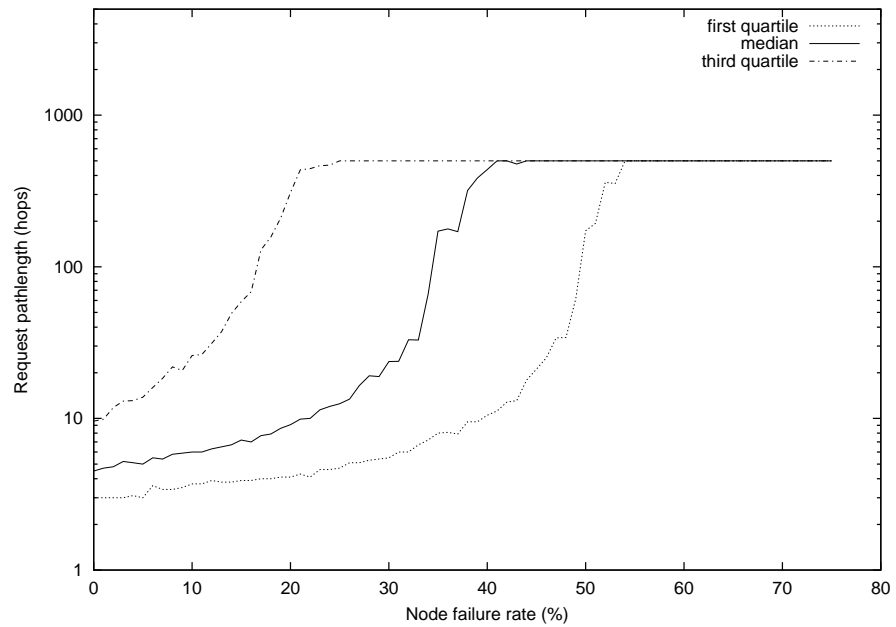


Figure 4: Increase in request path length under node failure.

They also have a high degree of fault-tolerance as random failures are most likely to disable poorly connected nodes, whose loss will not greatly affect routing in the network. Routing performance is only noticeably affected when the number of random failures becomes high enough to disable a significant number of the well connected nodes.

The authors claim [Ian\_01] that the simulated Freenet network conforms to the small-world model with respect to the distribution of the number of links.

How are these well connected nodes chosen? It may be the case that these nodes are the oldest nodes existing in the network, given that they have had more time to fill up their routing tables, in comparison to more recently added nodes. This explanation is not entirely justified. Requests fill up the routing table, and since requests do not favour an older node to a newer node once a network has reached a stable state, it is also likely that a newer node will have greater connectivity when compared to an older one. This is a desired effect, as it makes it harder for an outsider to identify the well connected nodes for malicious attack.

## 6 Conclusion

So how far does Freenet meet its design goals?

- *Anonymity for information producers and consumers:* The method of indexing data using file keys makes it extremely difficult for an outsider to guess the contents of a file from its hash key. The identities of information producers and consumers are further kept anonymous by the replicate on request, and replicate on insert mechanisms, since any node along the request path has the choice to change the source node field in a routing message.
- *Deniability for storing hosts:* Using key indexing, the user at a node has no way of knowing what files reside at his node apart from a brute force dictionary search.
- *Resistance to attempts by third parties to deny access to information:* Since files are duplicated on collision, malicious attempts at inserting a bogus file under a given name will result in spreading the real file further.
- *Decentralisation of all network functions:* The commonly chosen hash function, along with Freenet's routing mechanism eliminates the need of centralised routing information.
- *Efficient dynamic storage and routing of information:* Simulations done by the authors seem to suggest that this form of routing shows favourable characteristics, with respect to network convergence, stability and fault tolerance.

In my opinion, the Freenet system proposes a novel way of routing information through a network. Traditionally, routes in a network are chosen using a lowest cost path computation from one node to another. Information had to be indexed with the node that possessed it. In order to keep information sources anonymous, Freenet directly indexes the information stored by them. Its use of a commonly agreed hash function to calculate file keys, and their use in routing requests is a smart way of achieving fast network convergence without sacrificing on distributivity.

Although Freenet seems to meet its proposed design goals, there do exist some problem areas:

- *Load balancing*: All nodes are treated identically, even the slowest ones. There exists nothing to prevent a slow node from becoming well-connected, slowing the entire network down.
- *Search strategy*: The user needs to enter an exact description string in order to access a file. Boolean or wildcard searches are not feasible. Slight variations or spelling errors result in failed searches.
- *Accuracy of information*: Since sources of information are anonymous, there is no way of accessing the credibility of information accessed from the Freenet network. It cannot be assumed that frequently accessed information needs to be accurate.
- *Information consensus*: It is possible for two different files indexed with the same key to be present on two different areas of the network. Therefore information accessed from the network can depend on the position of the node in the network.

## References

- [Ian\_01] Ian Clarke, Oskar Sandberg, Brandon Wiley and Theodore W. Hong: *Freenet: A Distributed Anonymous Information Storage and Retrieval System*; Lecture Notes in Computer Science (vol 2009), 2001.
- [Hong\_01] Theodore W. Hong: *Performance*; Peer-to-Peer, ed. A. Oram, 2001.
- [Milg\_67] S. Milgram: *The small world problem*; Psychology Today 1(1), 1967.
- [Wat\_67] D. Watts and S. Strogatz : *Collective dynamics of small-world networks*; Nature 393, 1998.