Freenet

Parin Chheda (07CS3023)
Sarthak Jain (07CS1026)
Ravi Niranjan (07CS3010)
Teacher: Prof. Niloy Ganguly
Department of Computer Science and Engineering
IIT Kharagpur

September 10, 2010

1 Introduction

Freenet is a peer-to-peer network application that permits the publication, replication, and retrieval of data while protecting the anonymity of both authors and readers. Freenet operates as a network of identical nodes that collectively pool their storage space to store data files, and cooperate to route requests to the most likely physical location of data. The five main design goals are:

- Anonymity for both producers and consumers of information.
- Deniability for storers of information.
- Resistance to attempts by third parties to deny access to information.
- E cient dynamic storage and routing of information.
- Decentralization of all network functions.

The system is designed to respond adaptively to usage patterns, transparently moving, replicating, and deleting les as necessary to provide e cient service without resorting to broadcast searches or centralized location indexes. Freenet does not, however, explicitly try to guarantee permanent data storage. Because disk space is finite, a tradeoff exists between publishing new documents and preserving old ones.

2 Architecture

Freenet participants each run a node that provides the network some storage space. To add a new file, a user sends the network an insert message containing the file and its assigned location-independent globally unique identifier (GUID), which causes the file to be stored on some set of nodes. During a files lifetime, it might migrate to or be replicated on other nodes. To retrieve a file, a user sends out a request message containing the GUID key. When the request reaches one of the nodes where the file is stored, that node passes the data back to the requests originator.

2.1 GUID Keys

Freenet GUID keys are calculated using SHA-1 secure hashes. There are two main types of keys: content-hash keys and signed-subspace keys which are analogous to inodes and filenames in a conventional file system.

1. **Content-hash keys (CHKs):** The CHK is a low-level data-storage key and is generated by hashing the contents of the file to be stored. Hence, CHK is unique absolute indentifier for every file. Also, identical copies of a file inserted by different users would automatically be coalesced because every user will calculate the same key for the file.

2. **Signed-subspace keys (SSKs):** The SSK sets up a personal namespace that anyone can read but only its owner can write to. You could create a subspace by first generating a random public-private key pair to identify it. To add a file you first choose a short text description. You would then calculate the files SSK by hashing the public half of the subspace key and the descriptive string independently before concatenating them and hashing again.

Signing the file with the private half of the key provides an integrity check as every node that handles a signed-subspace file verifies its signature before accepting it. To retrieve a file from a subspace, you need only the subspaces public key and the descriptive string, from which you can recreate the SSK. Adding or updating a file, on the other hand, requires the private key in order to generate a valid signature. SSKs thus facilitate trust by guaranteeing that the same pseudonymous person created all files in the subspace, even though the subspace is not tied to a realworld identity.

2.2 Messaging and Privacy

Rather than move directly from sender to recipient, messages travel through node-to-node chains, in which each link is individually encrypted, until the message finally reaches its recipient. Each node knows only about its immediate neighbours and the identities of receiver and sender are protected. Hence it prevents an adversary from destroying a file by attacking all of its holders.

2.3 Routing

Freenet uses steepest-ascent hill-climbing search: Each node forwards queries to the node that it thinks is closest to the target. This makes is scalable and robust.

2.4 Requesting files

Every node maintains a routing table that lists the addresses of other nodes and the GUID keys it thinks they hold. When a node receives a query, it first

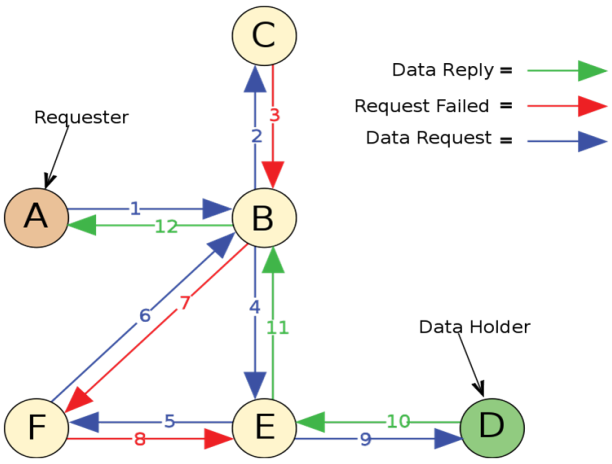

Figure 1: Request Sequence

checks its own store, and if it finds the file, returns it with a tag identifying itself as the data holder. Otherwise, the node forwards the request to the node in its table with the closest key to the one requested. If the request is successful, each node in the chain passes the file back upstream and creates a new entry in its routing table associating the data holder with the requested key.

To conceal the identity of the data holder, nodes will occasionally alter reply messages, setting the holder tags to point to themselves before passing them back up the chain. Later requests will still locate the data because the node retains the true data holders identity in its own routing table and forwards queries to the correct holder. Routing tables are never revealed to other nodes. To limit resource usage, the requester gives each query a time-to-live limit that is decremented at each node. If the TTL expires, the query fails, although the user can try again with a higher TTL (up to some maximum). Because the TTL can give clues about where in the chain the requester is, Freenet offers the option of enhancing security by adding an initial mix-net route before normal routing. This effectively repositions the start of the chain away from the requester.

If a node sends a query to a recipient that is already in the chain, the message is bounced back and the node tries to use the next-closest key instead. If a node runs out of candidates to try, it reports failure back to its predecessor in the chain, which then tries its second choice, and so on.

2.5 Inserting files

To insert a file, a user assigns it a GUID key and sends an insert message to the users own node containing the new key with a TTL value that represents the number of copies to store. An insert message follows the same path that a request for the same key would take, sets the routing table entries in the same way, and stores the file on the same nodes. Thus, new files are placed where queries would look for them.
The insert may fail either because the file is already in the network (for CHKs) or the user has already inserted another file with the same description (for SSKs). If the TTL expires without collision, the final node returns an all clear message and each node along the path verifies the data against its GUID, stores it, and creates a routing table entry that lists the data holder as the final node in this chain.

2.6 Data Encryption

The node operators might wish to remain ignorant of the contents of their data store. So, the publisher can encrypt the data before insertion. Thus, node operators cannot read their own files, but users can decrypt them after retrieval.

3 Network Evolution

3.1 Adding Nodes

To join the network, a new node first generates a public-private key pair for itself and sends an announcement message including the public key and physical address to some existing node with a user-specified TTL. The receiving node notes the new nodes identifying information and forwards the announcement to another node chosen randomly from its routing table. The announcement continues to propagate until its TTL runs out. At that point, the nodes in the chain collectively assign the new node a random GUID.

3.2 Training Routes

Nodes routing tables should specialize in handling clusters of similar keys so that it would answer them successfully more often. Nodes data stores should also specialize in storing clusters of files with similar keys. Because inserts follow the same paths as requests, similar keys tend to cluster in the nodes along those paths.

3.3 Key Clustering

Because GUID keys are derived from hashes, the closeness of keys in a data store is unrelated to the corresponding files contents. This lack of semantic closeness is unimportant, however, because the routing algorithm is based on the locations of particular keys, rather than particular topics.

4 Conclusion

The Freenet network provides an effective means of anonymous information storage and retrieval. By using cooperating nodes spread over many computers in conjunction with an efficient routing algorithm it keeps information anonymous and available while remaining highly scalable.