

Freenet: A Distributed Anonymous Information Storage and Retrieval System

<http://freenetproject.org>

Geoffrey Fairchild

Goals

- Anonymity for both producers and consumers of information
- Deniability for storers of information
- Resistance to attempts by 3rd parties to deny access to information
- Efficient dynamic storage and routing of information
- Decentralization of all network functions

Basic Idea

- Each node maintains a dynamic routing table
- This routing table contains addresses for other nodes and keys which these nodes are thought to hold
- Requests for keys are passed 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 for requests will vary

More About Requests

- Each request has a HTL (*hops-to-live*)
- Each request has a pseudo-random ID to prevent loops
- No node is privileged over any other node

File Keys

- 3 different types of file keys
 - KSK (*keyword-signed key*)
 - SSK (*signed-subspace key*)
 - CHK (*content-hash key*)

KSK (*keyword-signed key*)

- Derived from a short description the user provides
 - text/philosophy/sun-tzu/art-of-war
- Asymmetric key pair
 - Public half – hashed to yield the file key
 - Private half – used to sign the file to prevent others from tampering with the file
- Symmetric key
 - Used to encrypt the file for plausible deniability

KSK (cont.)

- Publish description to allow others to retrieve/search for the file
- Collisions are possible (and inevitable) since they're based on user text input
- “Key-squatting” possible – inserting junk values under popular descriptions to cause collisions

SSK (*signed-subspace key*)

- Public/private key pair created by user
- Public key and descriptive text (like the text used for the KSK) hashed separately and then XORed together, then hashed again
- Private key used to sign the file (like with the KSK)
- To allow others to retrieve the file, simply publish the public key and descriptive text
- Users can create their own namespace for files by using the SSK

CHK (*content-hash key*)

- Directly hash the file to get a CHK
- Designed for files that aren't going to change (such as audio/video files)
- Files encrypted with randomly generated encryption key
- To allow others access to the file, simply publish the file's hash and the decryption key

CHK (cont.)

- CHK are useful for splitting files (the author suggests 2^n kilobyte chunks)
- Easily accomplished by splitting file, putting those files on Freenet, and creating an indirect file that references each chunk

How to Find Keys?

- Hypertext spider
 - Centralization problem
- Special class of indirect files
 - Named according to search keywords, each file would contain the relevant hash(es)
 - Multiple files with the same search keyword are allowed
 - Managing these is an open problem
- Users create their own compilations of keys and publish them on the internet

Retrieving Data

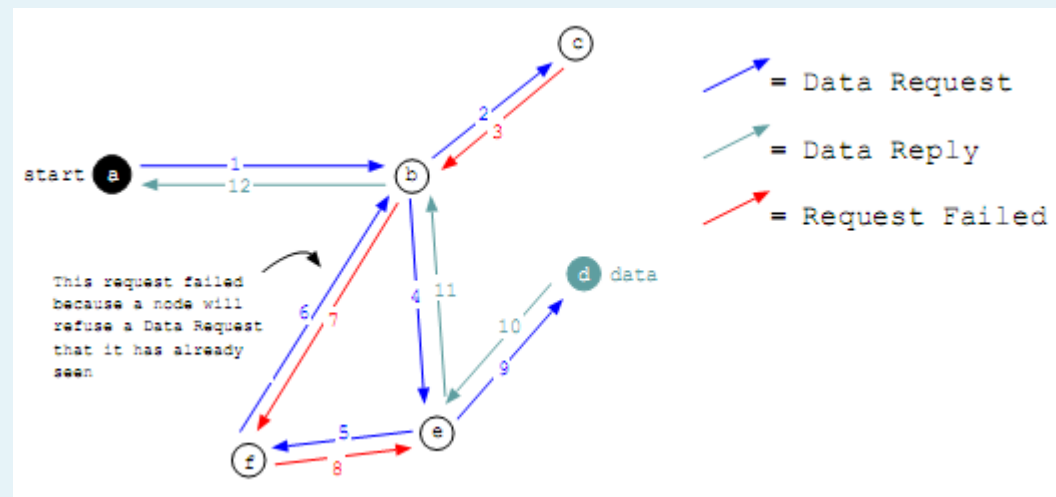
- User sends request to own node with key and HTL
- Node checks its own data store
 - If found, return data along with note saying it was the source
 - If not found, look up nearest key (lexicographically) in its routing table and forward request to that node
- If request is ultimately satisfied, pass the data to requester, cache data in own data store, and add data to own routing table

Retrieving Data (cont.)

- Subsequent requests for same file will be served from own data store
- Subsequent requests for “similar” keys (lexicographically) will be forwarded to previously successful data source
- Due to anonymity concerns, any node along the way can unilaterally decide to claim itself or another arbitrarily-chosen node as the data source

Retrieving Data (cont.)

- If a node can't forward a request (node down, possible loop), it tries the 2nd closest match, 3rd closest match, etc
- If HTL is reached, report failure



Storing Data

- Must check for key collisions
 - Use same method as in retrieval for finding if a hash exists
 - If it exists, nodes return data just like a retrieval
 - Each node between the initiator and the data source will cache this data
 - If the HTL count is reached, the initiator is given the “all clear” which means it's OK to insert

Storing Data (cont.)

- Once the “all clear” is received, propagate the data along the path established by the initial query
 - Each node in between will cache the data
- Each node can again unilaterally decide to change the insert message to claim itself as the data source to provide anonymity

Effects

- Newly inserted files are placed on nodes with similar keys
- New nodes can use inserts as a supplementary means of announcing themselves to the network
- Attacks to supplant existing files will further spread existing files since originals are propagated on collision

Managing Data

- Storage is not infinite
- The data store is implemented like an LRU cache
 - Older unused files will be deleted in favor of newer more popular files
- Since files are encrypted, node operators can claim plausible deniability since there's no way for them to know the contents of the data

Adding Nodes

- Must join the network out-of-band
- New node chooses random seed and hashes it
- New node sends hash of seed and address to the known node
- Known node generates a random seed, XORs with the received hash, and hashes that value to create a “commitment”

Adding Nodes (cont.)

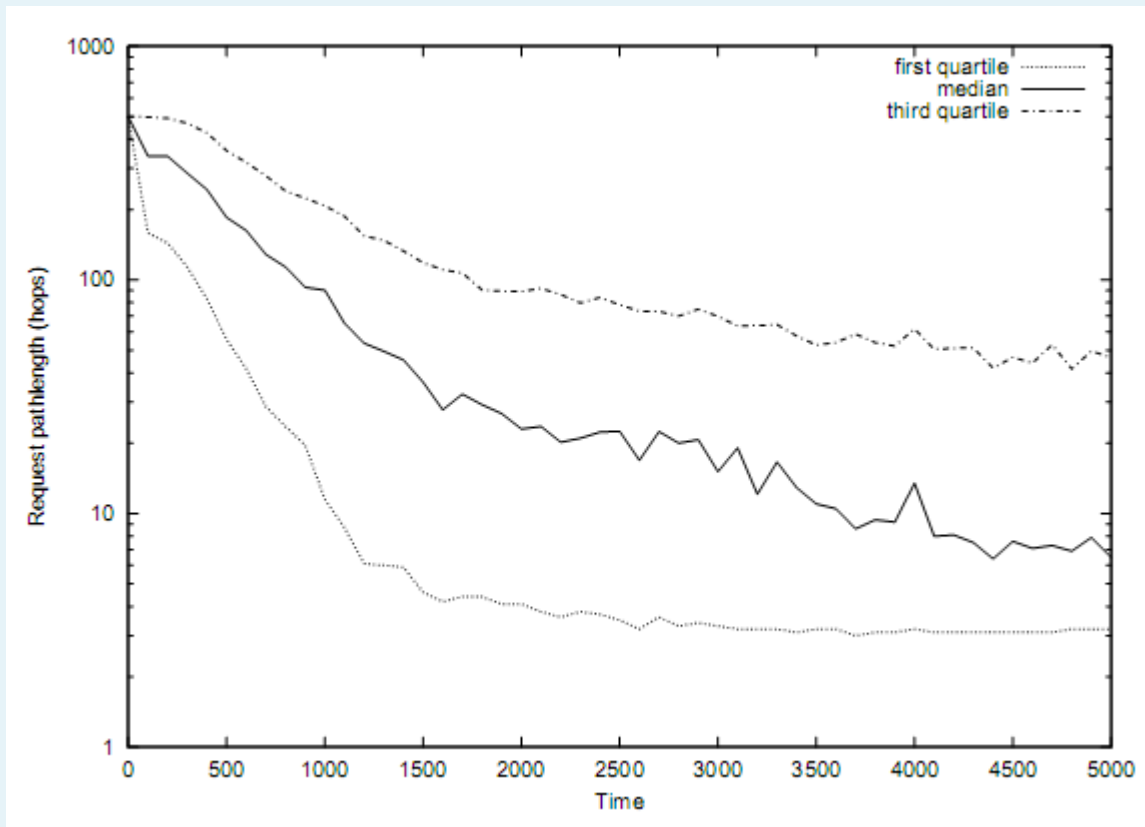
- Each node forwards this commitment to a randomly chosen node in his routing table
- Each node in turn performs this computation and forwards its commitment
- This process continues until the HTL expires

Adding Nodes (cont.)

- The last node just generates a seed
- All nodes in the chain reveal their seeds
- The key for the new node is assigned to be the XOR of all the seeds
- The commitment value is used so that each node can verify that every other node told the truth to know that there're no malicious nodes
- Each node in the chain adds the new node to his routing table

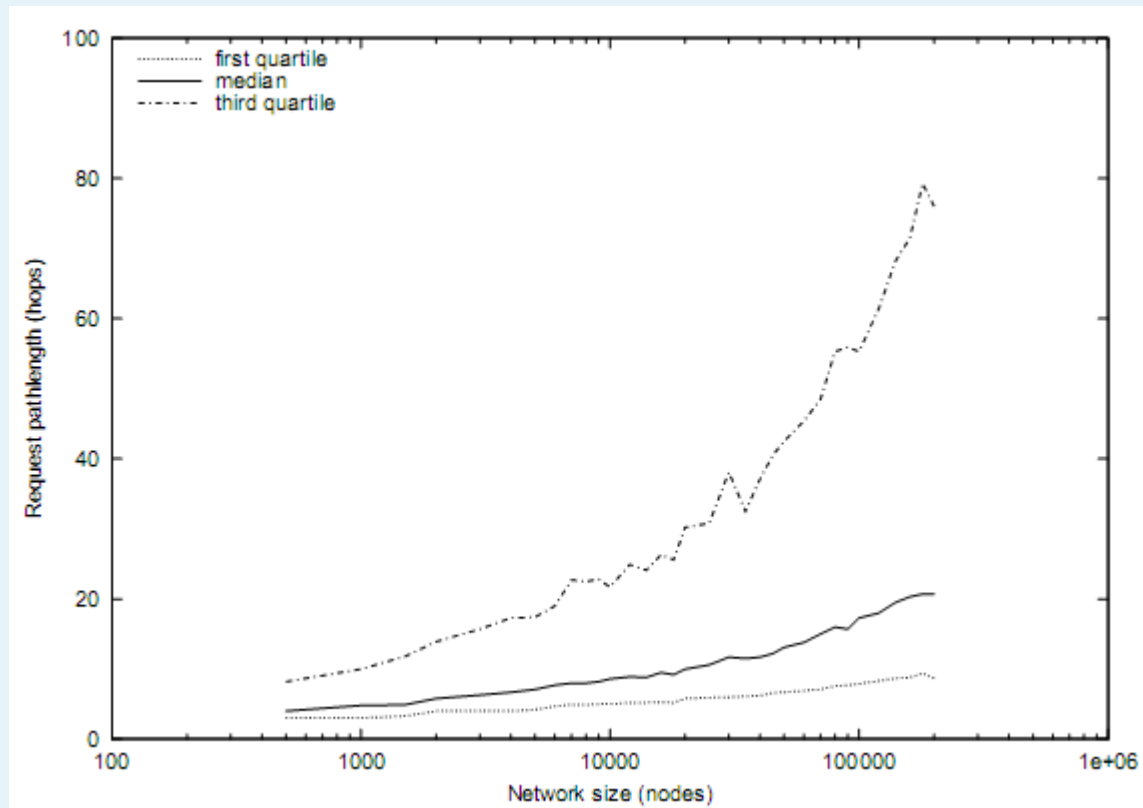
Network Convergence

- As time goes on and data gets inserted/requested, # hops goes down



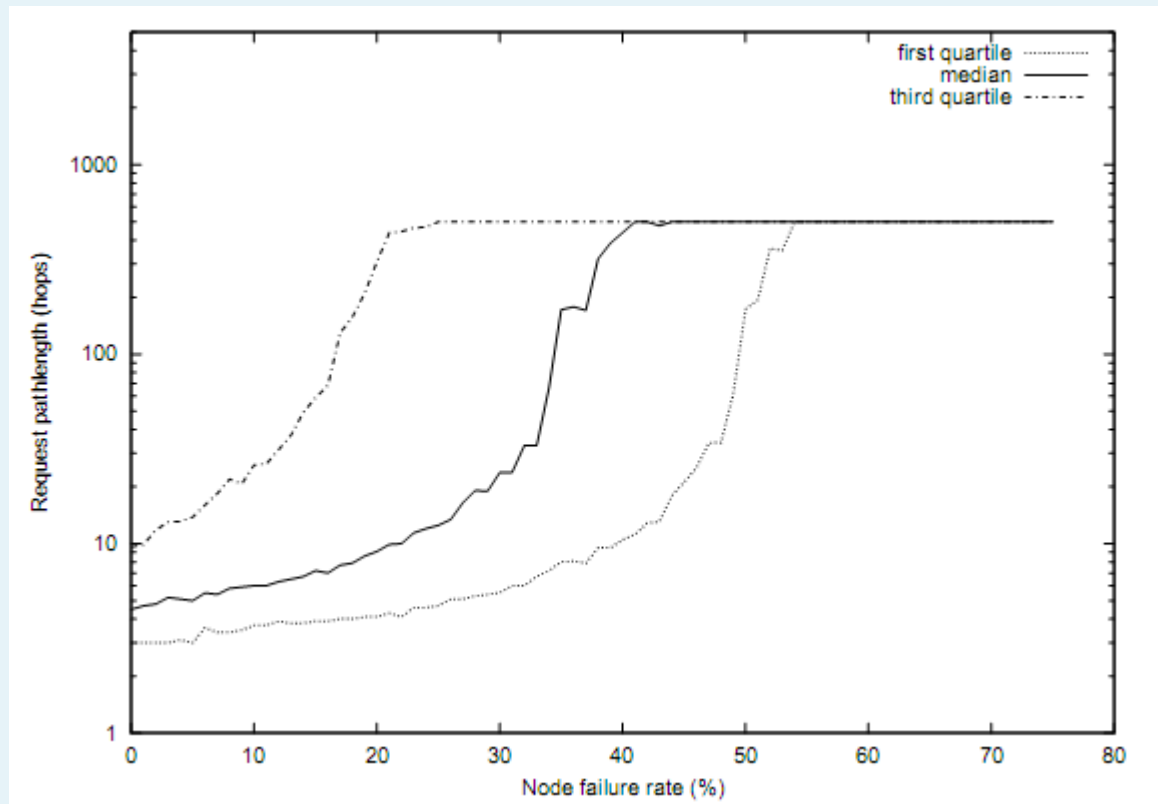
Scalability

- Even with a small routing table (250 entries), # hops grows logarithmically with respect to the number of nodes



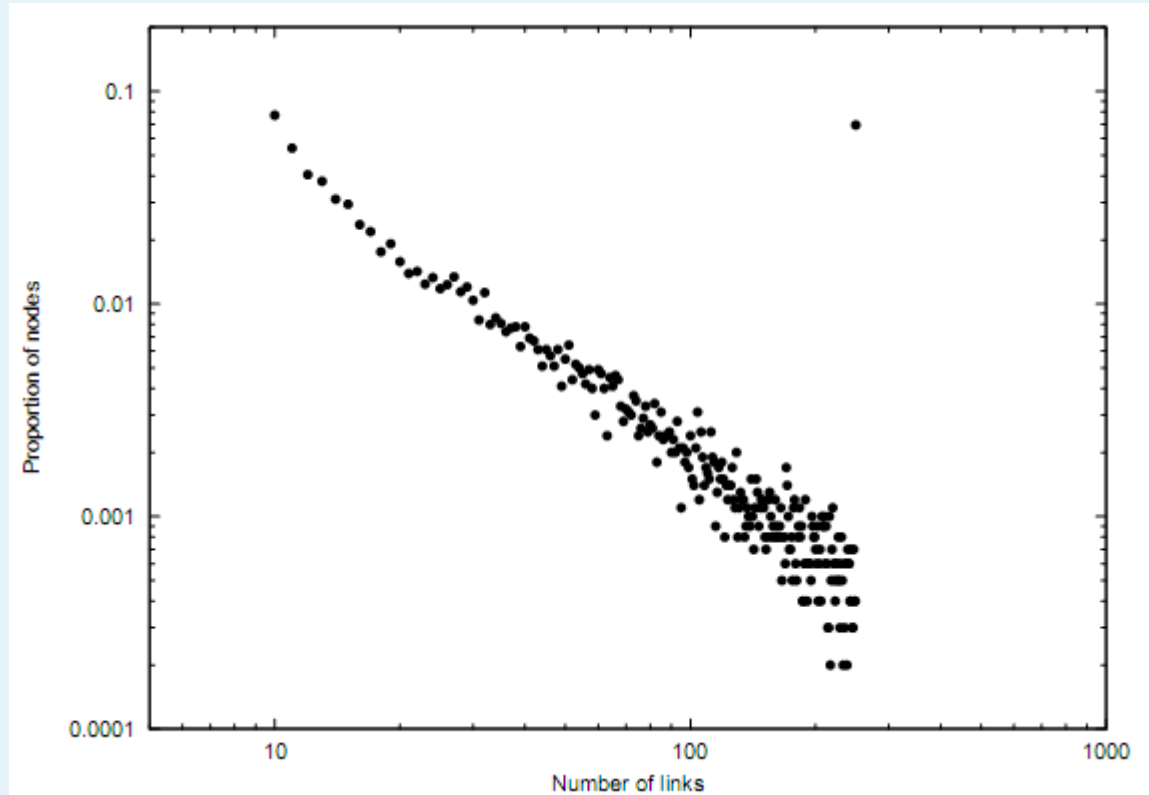
Fault Tolerance

- The median # hops remains below 20 even when up to 30% of the nodes fail



Small World Model

- As a result of the clustering of hashes, there is indeed a power-law distribution of links



Small World Model (cont.)

- The power-law distribution yields a high degree of fault tolerance
 - Random failures are most likely to knock out nodes with contain a small number of connections
 - Loss of poorly connected nodes won't greatly affect routing

Uses

- Currently being developed at <http://freenetproject.org/>
- <http://freekiwiki.sourceforge.net/> and <http://friki.berlios.de/> – anonymous wiki built using Freenet and MediaWiki
- Thaw is a file sharing application included with Freenet