Protecting Free Expression Online with Freenet

Ian Clarke[1], Theodore W. Hong[2], Scott G. Miller[1], Oskar Sandberg[1], and Brandon Wiley[1]

[1] The Freenet Project, 2554 Lincoln Blvd. #712, Venice, CA 90291, USA
{ian,scgmille,oskar,brandon}@freenetproject.org
[2] Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, United Kingdom
Tel +44 20 7594-8233 / Fax +44 20 7581-8024
t.hong@doc.ic.ac.uk

Introduction

Freedom of expression in the digital age is coming increasingly under threat with the growth of censorship and the erosion of privacy on the Internet. Governments around the world, including those of Australia, France, Germany, China, Saudi Arabia, and Singapore, have undertaken a range of efforts to force Internet service providers to block access to content deemed unsuitable or to make them liable for such material hosted on their servers, as documented by Human Rights Watch (http://www.hrw.org/advocacy/internet/) and the Global Internet Liberty Campaign (http://www.gilc.org/). In several cases, bans extend to cultural and political content. The United States itself recently mandated the installation of content filters in public libraries receiving federal funds.

In addition, various state and corporate actors have gone further and tried not just to block but to destroy certain materials altogether: from the Church of Scientology's lawsuits against websites hosting copies of their religious texts, to the Motion Picture Association of America's efforts to halt dissemination of the DeCSS cryptographic algorithm, to the British government's attempts to suppress the revelations of former MI6 agent Richard Tomlinson.

Privacy, too, is threatened as personal information flows are increasingly subject to monitoring and surveillance. The Electronic Privacy Information Center (http://www.epic.org) has raised important civil liberties questions about the FBI's Carnivore system, which can reportedly scan in real time all email passing through an ISP, and the EU's new Convention on Cybercrime, which gives European authorities broad powers to intercept and record digital communications. Recent incidents such as the dismissal of the dean of the Harvard Divinity School for downloading pornography to his home computer, the publication of Monica Lewinsky's deleted personal emails in a Congressional report, and the move by DoubleClick to link names and addresses to profiles of individual web browsing activity, all point to an unprecedented level of intrusion into private life[12].

These trends are of serious concern not only to whistleblowers or political dissidents, but to anyone disturbed by the thought of others reading her mail or watching her do a web search for, say, "Aids support groups." Indeed, free

expression and privacy are closely entwined in the process of communication: controversial information may need to be published anonymously to protect its author against retaliation; it must be preserved against removal or blocking at the locations where it is held; and it may need to be retrieved anonymously to protect its readers from adverse consequences.

Fortunately, concurrent advances in the power of personal computers have made possible the development of new peer-to-peer technologies to respond to these challenges. For this reason, we are developing Freenet[5], a distributed information storage system designed to address concerns of information privacy and survivability. The system operates as a self-organizing peer-to-peer network that pools unused disk space across hundreds of thousands of desktop computers to create a collaborative virtual file system.

Our main design goals are:

- Privacy for producers, consumers, and holders of information
- Resistance to censorship of information
- High levels of availability and reliability through decentralization
- Efficient, scalable, and adaptive storage and routing

Privacy for producers and consumers means the ability to create and retrieve files anonymously. Since this means little without protecting the survival of the files themselves, we also seek privacy for information holders, meaning concealing the identities of which computers are storing which files. Together with redundant replication of data, holder privacy makes it extremely difficult for censors to block or destroy files on the network.

To increase the robustness of the network itself, Freenet strives for a completely decentralized architecture free from any single points of failure that might be attacked or overloaded, while maintaining efficiency and scalability. The peer-to-peer environment is inherently untrustworthy and unreliable, and we must assume that participants may operate maliciously or fail without warning at any time. Therefore, Freenet implements strategies to protect data integrity and prevent privacy leaks in the former instance, and provide for graceful degradation and redundant data availability in the latter. The system is also designed to adapt to usage patterns, automatically replicating and deleting files to make the most effective use of available storage in response to demand. These characteristics additionally help prevent server overload under sudden demand spikes (the so-called "Slashdot effect").

Freenet does not, however, explicitly seek to guarantee permanent data storage. Since disk space is finite, a tradeoff exists between publishing new documents and preserving old ones. The typical solution is to require payment (e.g. in disk space or money) for publishing, but we would like to encourage publishing, rather than keep out authors who might be unable to run peer nodes themselves or too poor to pay for storage. On the other hand, we need to guard against flooding by junk documents. These considerations have led to a probabilistic storage policy discussed in more depth later. We hope, however, that Freenet will attract sufficient resources from participants for most files to last indefinitely.

The software is currently under open-source development, and a beta version can be downloaded from `http://www.freenetproject.org/`.

Freenet Architecture

On an abstract level, Freenet works like this: Each file (or piece of a file) in the system is assigned a location-independent globally unique identifier (GUID) by its creator. Everyone who wants to participate in Freenet runs a node which provides some storage space to the network. To add a new file, a user finds a node to store it on and sends that node an insert message containing the file and its GUID. During a file's lifetime, it may migrate to or be replicated on other nodes. To retrieve a file, a user sends a request message containing its GUID key to some node where it is stored, and receives the data back.

Keys

Freenet GUID keys are calculated using SHA-1 hashes. Two main types of keys are used, *signed-subspace keys* and *content-hash keys*.

Signed-Subspace Keys. The signed-subspace key (SSK) sets up a personal namespace that can only be written to by its owner. A user creates a subspace by randomly generating a public/private key pair to identify it. To add a file, she first chooses a short text description. For example, a user inserting a Vietnam War archive might assign it the description, `politics/us/pentagon-papers`. She then hashes the public half of the subspace key and the descriptive string independently, concatenates them, and hashes again to yield the SSK. The private half of the subspace key is used to sign the file as an integrity check. Every node that handles a signed-subspace file will verify its signature before accepting it.

To retrieve a file from a subspace, you only need to know the subspace's public key (perhaps stored on your keyring) and the descriptive string, from which you can recreate the signed-subspace key. Adding or updating a file requires the private key, however, which is kept secret by the owner. In this way, even though a subspace is not tied to a real-world identity, it enables trust by guaranteeing that all the files in it were created by the same person. This mechanism can be used for example to send out a newsletter under a pseudonym, to publish a website, or (operated in reverse) to receive email.

Various layers can be built on top of signed-subspace keys. For example, a hierarchical structure can be created using directory files containing hyperlinks to other files. A directory under the key `politics/us` might contain a list of keys such as `politics/us/pentagon-papers`, `politics/us/mcintyre-v-ohio`, and `politics/us/constitution`, plus recursive pointers to other directories. Another use for SSKs is to implement an alternative Domain Name System for nodes that change addresses frequently. Each such node would have its own subspace, regularly updated with its current location. The subspace's public

key—an *address-resolution key*—would take the place of a fixed address. To contact one of these nodes, you would simply look up its key to retrieve its current address.

Content-Hash Keys. The content-hash key (CHK) is the primary data-storage key. As the name suggests, a content-hash key is generated by hashing the contents of the corresponding file. This has the useful property of giving every file a unique absolute identifier that is easy to authenticate. If you send someone a CHK reference, you know that they will see the exact file you intended (unlike the behavior of URLs). Content-hash keys can also automatically coalesce identical copies of the same file inserted by different people, since the file will be assigned the same key each time.

To retrieve a file stored under a content-hash key, the full binary key is needed, making CHKs inconvenient for human use. Hypertext links solve this problem, but content-hash keys can also be used in direct communication by combining them with signed-subspace keys. To do this, a user first inserts a file under its content-hash key. She then inserts an indirect file under a signed-subspace key that points at the content-hash key. This enables others to retrieve the file from the signed-subspace key in two steps.

Indirect files are useful because they support updating in a convenient way. First the owner inserts a new version of the data under a content-hash key, which should differ from the old key since the contents are different. The signed-subspace key can then be updated to point to the new version, while the old version will still remain accessible by content-hash key to anyone who wants it.

Content-hash keys can also be used to split large files into multiple parts. A split file can be inserted by inserting each part under its own content-hash key and creating an indirect file to point to all the parts.

Messaging and Privacy

Privacy in Freenet is maintained using a variation of Chaum's mix-net[4] scheme for anonymous communication. In this scheme, messages are not sent directly from sender to recipient, but travel through chains in which node A sends a message to node B, node B passes it on to node C, and so on, until the message finally reaches its recipient. Each node-to-node link is individually encrypted. Since each node in the chain only knows about its immediate neighbors, the endpoints of the chain could be anywhere among hundreds of thousands of nodes in the network continually exchanging indecipherable messages. Even the node immediately after the sender can't tell whether its predecessor was the true originator of the message or was merely forwarding a message from someone else. Similarly, the node immediately before the receiver can't tell whether its successor was the true recipient or continued to forward it on. This arrangement is intended to protect not only information producers and consumers (at the beginnings of chains), but also information holders (at the ends of chains). It is important to protect the latter in order to prevent an adversary from destroying a file by attacking all of its holders.

Routing

Ensuring privacy is not enough, of course; queries must actually find the data as well. The routing of messages is the key element that defines the Freenet protocol.

The simplest routing method, used by services like Napster, is to maintain a central index of all files so requests can be sent directly to information holders. Apart from the obvious lack of security, centralization creates a single point of failure that is easy to attack. For example, if you were trying to phone George Clooney, calling directory assistance would be the most direct way to get his number, but you're stuck if the service goes down or someone removes his entry.

Systems like Gnutella broadcast queries to every connected node within some radius. Using this method, you would ask all of your friends if any of them knew Clooney's number, and get them to ask *their* friends, and their friends' friends, and so on. Within a few steps, the entire country would be looking for him, which would find the answer but is clearly wasteful and unscalable.

Freenet avoids both problems by using a steepest-ascent hill-climbing search. Each node forwards queries to the node that it thinks is closest to the target. For example, you might start searching for Clooney by asking a friend studying film, who might pass you on to someone she knows in L.A., who might pass you on to Clooney's agent, who could put you in touch with the man himself.

Requesting Files. The way this works is as follows: Every node maintains a routing table listing the addresses of other nodes together with the GUID keys it thinks they hold. When a node receives a query, it first checks its own store for the file, and if found, returns the file with a tag identifying itself as the data source. If not found, it looks up the numerically-closest key in its table to the key requested and forwards the request to the corresponding node. That node then checks *its* store, and so on. If the request successfully finds the data, each node in the chain will pass the file back upstream and create a new entry in its routing table associating the ultimate data source with the requested key. Each node may also cache a copy locally, depending on its distance from the source. Security can be enhanced by preceding chains with a random mix-net route before routing normally.

To limit resource usage, queries are given a time-to-live (TTL) limit that is decremented at each node. If the TTL expires, the query fails and an error is returned indicating that the key could not be found within the specified distance. (The user can try again with a higher TTL, up to some maximum.) If a node sees a query looping back, it rejects the message and the sender tries its next-closest key instead. If a node runs out of candidates to try, it reports failure back to its predecessor in the chain, which will then try *its* second choice, and so on.

Figure 1 depicts a typical request sequence. The user initiates a request at a. Node a forwards the request to b, which forwards it to c. Node c is unable to contact any other nodes and returns a "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 a loop and rejects the message. Node f is unable

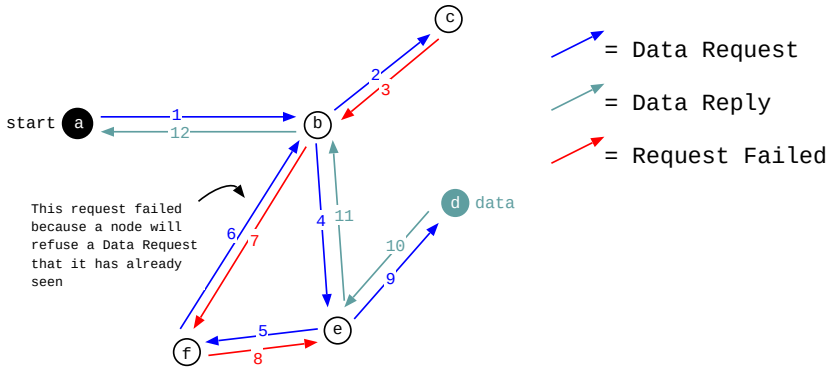

Fig. 1. A typical request sequence.

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 file. The file is returned from d via e and b back to a, which sends it to the user. The file may also be cached on e, b, and a.

The idea is that the request homes in closer with each hop until the key is found. A subsequent query for the same key will tend to approach the path that the first request took; once the two converge, the query can be satisfied by a locally-cached copy without going all the way to the original source. Subsequent queries for similar keys will also be speeded up by jumping over intermediate nodes to a node known to have previously supplied similar data. Hence nodes that reliably answer queries will gain routing table entries and be contacted more often than nodes that do not.

Inserting Files. Inserts work in a very similar way to requests. To insert a file, a user first assigns it a GUID key as previously described. She then sends an insert message containing the new key with a TTL value that represents the number of copies she wants to store. When a node receives an insert, it first checks its datastore to see if the key already exists. If so, the insert fails. For CHKs, since SHA-1 collisions are extraordinarily unlikely, this means that the file is already in the network and does not need to be inserted. For SSKs, this means that the user previously inserted another file with the same description. The user should either choose a different description, or perform an update rather than an insert. (Updates are not yet implemented because we are still working on a mechanism to make sure all old copies get replaced.)

Otherwise, the node looks up the closest key and forwards the message to the corresponding node in the same way as for queries. If the TTL expires without collision, an "all clear" result will be returned upstream. The user then sends the data down the path established by the initial insert message. Each node along

the path will verify the data against its GUID, store it, and create a routing table entry for it that sets the data source to be the final (farthest downstream) node in the chain. If a loop or a dead end is encountered, the insert backtracks to the second-nearest key, then the third-nearest, and so on, in the same way as for requests.

The idea here is that an insert follows the same path that a query for the same key would take, sets the routing table entries in the same way, and stores the file on the same nodes. Thus, the new file will be placed in precisely the places that a request would look for it in.

Data Encryption

For political or legal reasons, it may be desirable for node operators to remain ignorant of the contents of their datastores. To this end, users are strongly encouraged to encrypt all data before insertion. The network proper knows nothing about this level of encryption; it only ships around already-encrypted bits. In particular, data encryption keys are not used in routing and are not included in any network messages. They are only distributed directly to end users by inserters at the same time as the corresponding GUIDs. This arrangement makes it impossible for node operators to read their own files while still permitting users to decrypt them after retrieval. Node operators cannot gain any information by looking at GUIDs, either, since the hashes used to generate them scramble any identifying characteristics. From a node operator's point of view, her datastore consists only of random GUIDs attached to opaque data.

Network Evolution

Over time, the network evolves, both through new nodes joining the network and through existing nodes creating new connections following queries. As more requests are handled, local knowledge about other nodes in the network improves and routes adapt to become more accurate without needing global directories.

Adding Nodes

A node is logically identified by a public/private key pair, which is used to sign a physical address reference. It may move from one physical address to another, but will be recognized as the same node from its public key. The address-resolution key mechanism previously described can be used to inform other nodes of a physical move. Note that nodes' public keys are not certified; this is unnecessary since we don't need to link a key to a real-world identity—the public key *is* the node's identity. Certification may of course become useful in the future for deciding whether or not to trust a new node, but for now there is no trust mechanism.

To join the network, a new node first generates a public key for itself. It must then find an existing node through some out-of-band means, to which it sends an

announcement message giving its key and physical address. When a node receives such an announcement, it notes the new node's 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 in the keyspace using a cryptographic fair coin flip. This enables the new node to become responsible for a consistent region of keyspace while ensuring that no participant can bias the choice. In particular, a malicious node cannot influence the assignment of responsibility for a specific key that it might want to attack.

The announcement procedure also induces a *small-world* topology in the network, as described later. Small-world topologies frequently occur in nature and permit highly scalable routing even in very large networks.

Training Routes

The network's routing should train itself as more requests are processed, for two reasons. First, each node's routing table should become specialized in handling a cluster of similar keys as time goes by. Whatever key a node is associated with in others' routing tables—even if randomly assigned at first—will make it receive mostly requests for keys similar to that key. When those requests succeed, the node will learn about previously-unknown nodes that can supply such keys and create new routing entries for them. As it gains more "experience" in handling queries for those keys, it will succeed in answering them more often—and hence get asked about them more often, in a positive feedback loop. For example, if people kept asking you where to find bowling alleys, you would start to learn about them even if you knew nothing at first. As more people found out that you could answer bowling-related questions, you'd get asked about bowling more and more until you became an expert on the subject.

Second, nodes' datastores should also become specialized in storing clusters of files having similar keys. Because inserts of similar keys follow the same paths, similar keys will tend to cluster in the nodes in those paths. Files cached by nodes after requests should likewise be clustered since most requests will be for similar keys.

Taken together, the twin effects of clustering in routing tables and datastores should improve the effectiveness of future queries in a self-reinforcing cycle, as nodes begin to focus on particular clusters of keys that are precisely the keys that they are asked about. While we do not yet have a good mathematical model to analyze the training and convergence of the Freenet algorith, simulations show that the networkis in practice able to locate files quickly, with a median pathlength of just 8 hops in a 10,000-node network.

Key Clustering

Since keys are derived from hashes, the closeness of keys in a datastore will be unrelated to the closeness of the corresponding files' contents. This lack of semantic

closeness is unimportant, however, since the routing algorithm is based on knowing where particular keys are located, not where particular topics are located. For example, suppose a descriptive string such as `politics/us/pentagon-papers` yields the key `AF5EC2`. Requests for this file can be satisfied by creating clusters containing the keys `AF5EC1`, `AF5EC2`, and `AF5EC3`, as opposed to creating clusters containing works about U.S. politics. In fact, hashes are useful *because* political works will be scattered across the network, since this lessens the chances that the failure of a single node will make all politics unavailable. Similarly, the contents of any given subspace will be scattered across different nodes, increasing robustness.

Searching

One issue that remains open is how users can search for keys relevant to their needs. In many ways this is similar to the problem of searching the web, and the same solutions are possible. Like the web, Freenet can be spidered, or individuals can publish lists of bookmarks. One simple possibility for a native Freenet search would be to create a special public subspace for indirect keyword files. When an author inserted a file, she could also insert a number of indirect files pointing at the original file, whose names corresponded to search keywords. For example, the Pentagon Papers archive might have indirect files named `keyword:politics`, `keyword:united-states`, and `keyword:vietnam` pointing at it. These keyword files would differ from normal files in that multiple files having the same key would be permitted to coexist, and requests for such keys could return multiple matches. A search for "politics" might return a keyword file `keyword:politics` pointing to the Tiananmen Papers as well as one pointing to the Pentagon Papers. However, managing a large number of indirect files for common keywords would be difficult, since all the files having the same name would be attracted to the same nodes. A more sophisticated approach might use a distributed search over detailed metadata descriptors inserted along with the original files. Much work still needs to be done in this area.

Managing Storage

To encourage publishing, Freenet does not require payment for inserts or impose restrictions on the amount of data a publisher can insert. However, as a result the system may have to decide which files to keep, given finite disk space. The current system prioritizes space allocation by popularity, as measured by the frequency of requests per file. Within each node's datastore, files are kept ordered by time of last access. When a new file arrives that cannot fit in the free space available, the least-recently-accessed (i.e. most unpopular) files are deleted until there is room.

Evicted files do not completely disappear right away, since their associated routing table entries are smaller and will stay around for longer. If an evicted

file is requested at a later date, the node can use the routing table to fall back to the original data source, which may be able to supply another copy.

Why would the original source be any more likely to have it? Data source pointers in Freenet, considered as graphs, have a tree-like structure. Nodes at the leaves may only see a few local requests for a file, but nodes higher up the tree receive requests from a larger part of the network, making their copy more popular.

The distribution of files is therefore determined by two competing forces. When files are requested from one part of the network, the query-routing mechanism automatically creates more copies there and the tree grows in that direction. This improves response time and prevents Slashdot-type overloading. When files go unrequested in another part, they become subject to deletion and that part of the tree shrinks, freeing up space for other files. The net effect is that the number and location of copies of each file adjusts to the demand for it.

An important problem that still needs further work is defending against a denial-of-service attack that floods the system with junk data in order to fill up all available space or to overwrite existing data. Although the eviction mechanism works to eliminate files that are never requested, if it does not act quickly enough important files may be pushed out. On the other hand, reducing the priority of new data may result in files being deleted too soon before they have had a chance to be requested. Various modifications to the caching policy are being explored to balance these considerations, such as caching less aggressively farther down the data source pointer tree.

Performance Analysis

Simulations were used to test the performance of Freenet. Here we summarize the most important results; for full details, see [9].

Scalability

To test Freenet's scalability, we created a simulated network of 20 nodes connected initially in a ring topology. Inserts of randomly-generated files were sent to random nodes in the network, interspersed with random requests for files that had been inserted so far (all with TTL 20). After every five inserts/requests, a new node was created and announced itself to a random existing node. After every hundred inserts/requests, the network's performance was measured by issuing a set of test requests for previously-inserted files and recording the resulting distribution of pathlengths (the number of hops actually taken to find the data). This continued until the network reached 200,000 nodes.

Figure 2 shows the evolution of the first, second, and third quartiles of the request pathlength versus network size, averaged over ten trials. We can see that the median pathlength strongly fits a sublinear power law with exponent k=0.28, in line with recent mathematical modelling of peer-to-peer networks done by Adamic *et al.*[1]. Extrapolating to larger sizes, it appears that Freenet

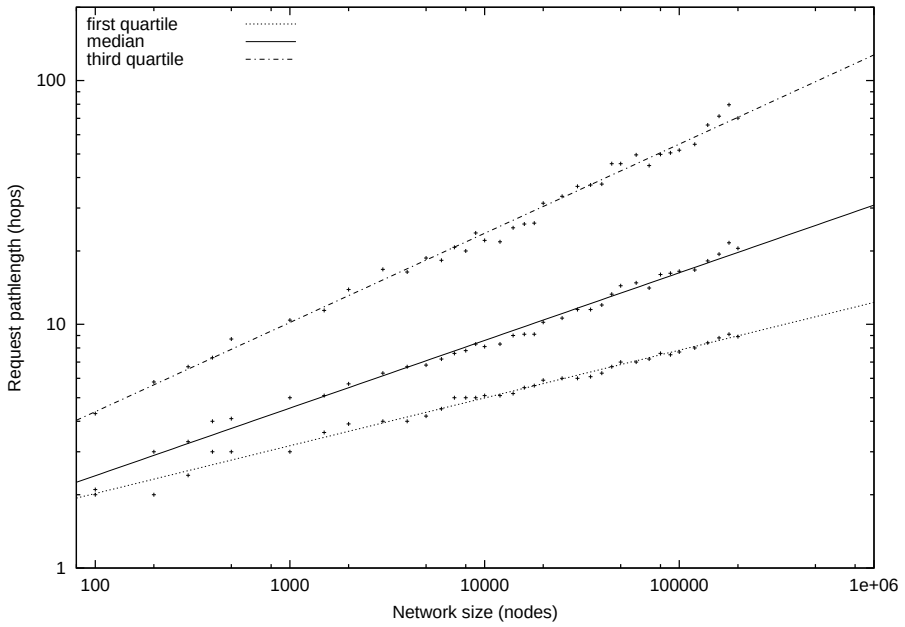

Fig. 2. Request pathlength versus network size.

should be capable of scaling to one million nodes with a median pathlength of just 30.

Fault Tolerance

Next, we examined Freenet's fault tolerance. After repeating the previous training procedure to 10,000 nodes, we progressively removed random nodes from the network to simulate node failures. Figure 3 shows the resulting evolution of the request pathlength, averaged over ten trials. The network is surprisingly robust against quite large failures. The median pathlength remains below 20 even when up to 30% of nodes fail. (Note that requests were capped at 500 hops before giving up.)

The Small-World Model

These characteristics of Freenet can be explained in terms of a *small-world* network model[15]. Small-world networks are characterized by a power-law distribution of graph degree in which the majority of nodes have only relatively few, local connections to other nodes but a significant small number of nodes have large, wide-ranging sets of connections. The small-world topology enables efficient short paths even in very large networks because of the shortcuts provided by the well-connected nodes.

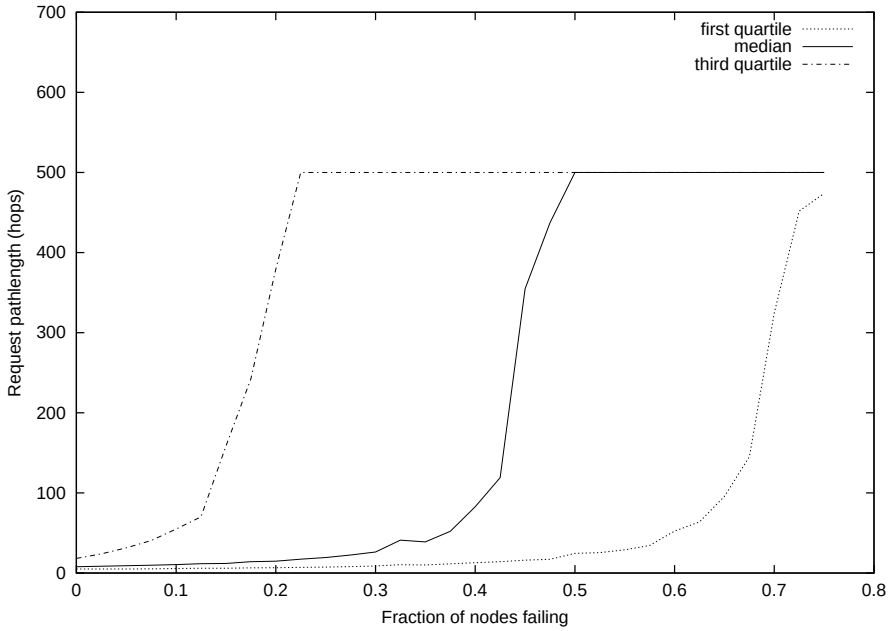

Fig. 3. Change in request pathlength under random failure.

Is Freenet a small world? Figure 4 shows the average distribution of graph degree (i.e. number of routing table entries) in a 10,000-node trained network. We see that the distribution closely approximates a power law with t=1.5, except for an outlier resulting from the maximum routing table size cutoff (250 in this simulation). This is not surprising, as power-law distributions tend to arise naturally through the interaction of network growth with preferential attachment (that is, new nodes prefer to connect to nodes that already have many links)[2]. The new-node announcement protocol initially creates a preferential attachment effect because following random links gives a higher probability of arriving at nodes that have more links. During normal operation, the effect continues because nodes that are well-known tend to see more requests and become even better connected ("the rich get richer").

In addition to creating short paths, the power-law distribution also gives small-world networks a high degree of fault tolerance[2]. Random failures are most likely to knock out nodes from the poorly-connected majority, whose loss will not greatly affect routing. It is only when the number of failures becomes high enough to knock out a significant number of well-connected nodes that performance will be noticeably affected. On the other hand, if the well-connected nodes are specifically targeted first, a small-world network falls apart much more quickly. This can be seen in Figure 5, which shows the size of the largest connected component in a 10,000-node network as nodes are removed, both randomly and in order from most-connected to least-connected. Under random fail-

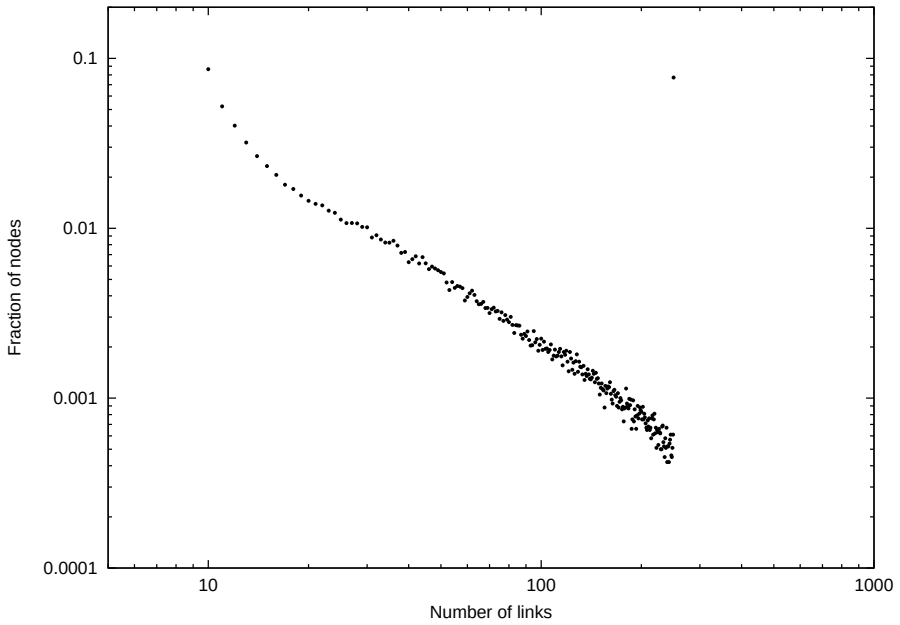

Fig. 4. Degree distribution among Freenet nodes.

ure, the vast majority of the network remains connected until almost the very end. Under targeted attack, however, the network undergoes a percolation transition near 60% removal, when it abruptly falls apart into disconnected fragments.

Related Work

The most well-known similar systems are Napster (`http://www.napster.com/`) and Gnutella (`http://gnutella.wego.com/`), which both implement large-scale pooling of disk space among individual users. The major difference is that they provide a file *sharing* service rather than a file *storage* service—that is, participants make their own files available to others but do not push files to other nodes for storage. Hence data is not persistent in the network; files are only available when their originators (or subsequent requesters) are online. Neither does either system attempt to provide anonymity. Gnutella is also extremely inefficient, broadcasting thousands of messages per request.

The paradigm followed by Freenet more closely resembles the proposal for the Eternity service[3], which set out a broad vision of a highly-survivable network for archiving information permanently and anonymously, though lacking in specifics on how to efficiently implement such a service. Another Eternity-like system is Free Haven[7], an anonymous peer-to-peer publication system that uses trust

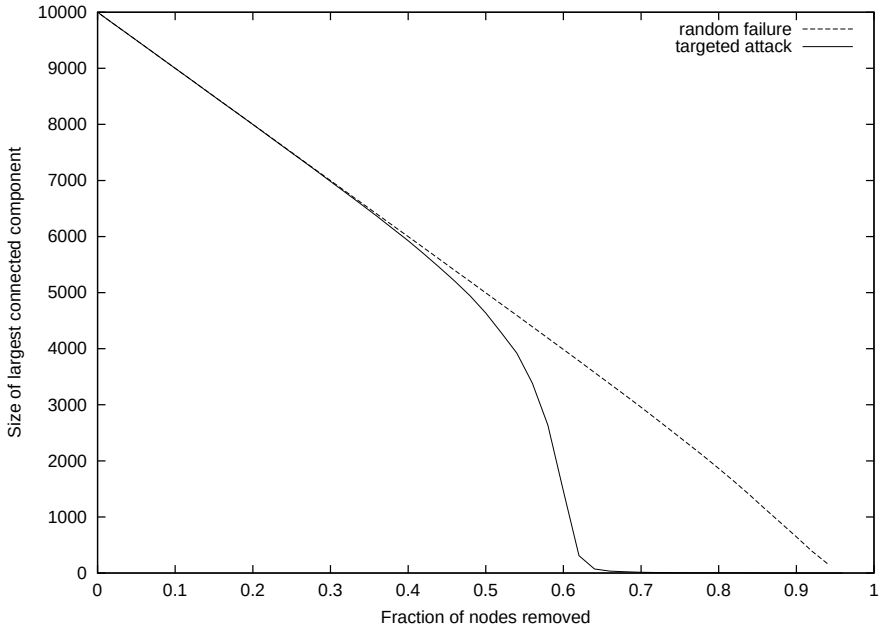

Fig. 5. Network connectivity under random failure and targeted attack.

mechanisms and file trading to enforce server accountability as well as user anonymity; however, it can take days to retrieve files from it.

A number of peer-to-peer file storage systems have been developed recently that focus on efficient location of data rather than issues of privacy and security against malicious participants: these include OceanStore[11], CFS (based on Chord)[6], and PAST[13]. All are based on routing models in which each node is assigned a fixed identity and maintains some knowledge of those nodes whose identities vary in specified ways from their own. (OceanStore additionally uses a probabilistic routing algorithm based on Bloom filters.) Data is deterministically placed on nodes whose identities most closely match the data's GUID. In these systems, data can be located by progressively visiting nodes whose identities match more and more bits of the desired GUID. Their main advantage is that they can provide strong guarantees that data will be located within certain time bounds (generally logarithmic) if it exists. In turn, this permits better handling of issues like storage management.

The main disadvantage of these systems relative to Freenet is that they are more difficult to secure against attack. It is easier for a malicious node to manipulate its identity so as to gain responsibility for a particular piece of data and suppress it. Links and routing are also more deterministically structured and visible to all, making it easier to trace messages, and harder to avoid malicious nodes that sabotage requests by pretending data could not be found. PAST as currently constituted also requires users to trust external smart cards.

Freenet was designed from the ground up under the assumption of hostile attack from both inside and out. Therefore it intentionally makes it difficult for nodes to direct data towards themselves and keeps its routing topology dynamic and concealed. Unfortunately, these considerations have had the side effect of hampering changes that might improve Freenet's routing characteristics. To date, we have not discovered a way to guarantee better data locatability without compromising security.

Systems focusing on privacy for information consumers include browser proxy services such as the Anonymizer (`http://www.anonymizer.com/`) and SafeWeb / Triangle Boy (`http://www.safeweb.com/`). Both provide anonymity by proxying requests for Web content on the user's behalf, although users are vulnerable to logging by the services themselves. Crowds[10] improves anonymity over simple proxying through a request-chaining technique similar to the one we use. None of these systems directly store information themselves; they only provide anonymized access to information available on the Web.

On the producer/holder side, the Rewebber (`http://www.rewebber.de/`) provides some privacy for information holders with an encrypted URL service that is the inverse of a browser proxy, but is similarly vulnerable to logging by the service operator. TAZ[8] extends this idea with chains of nested encrypted URLs that point to successive Rewebber-type servers to be contacted. Neither protects information producers and both rely on a single server as the ultimate source of information. Publius[14] enhances robustness and protects producer anonymity by distributing files as redundant partial shares among many holders; however, since the identity of the holders is not anonymized, an adversary could still destroy information by attacking a sufficient number of shares. None of these systems protect information consumers, although Rewebber also operates a browser proxy service.

Conclusions

Freenet provides an effective means of anonymous information storage and retrieval to help combat the growth of censorship and the erosion of privacy online. By using cooperating nodes spread over many computers in conjunction with an efficient adaptive routing algorithm, it keeps information anonymous and available while remaining highly scalable. Initial beta deployment is underway, and is so far proving successful, with hundreds of thousands of copies downloaded and many interesting files in circulation. Because of the anonymous nature of the system, it is impossible to tell exactly how many users there are or how well inserts and requests are working, but anecdotal evidence is positive. We are working on implementing a simulation and visualization suite that will enable more rigorous tests of the protocol and routing algorithm. More realistic simulation and formal modelling are necessary to explore the effects of nodes joining and leaving, variations in node capacity and bandwidth, and larger network sizes. Finally, more work is needed to develop search mechanisms and to provide more protection against denial-of-service attacks.

Acknowledgements

The second author thanks the Marshall Aid Commemoration Commission for their support. This material is partly based upon work supported under a National Science Foundation Graduate Research Fellowship.

Additional information about Freenet, including author contact information and software downloads, is available at http://www.freenetproject.org/.

References

1. L.A. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman, "Search in power-law networks," *Physical Review E* **64**(4), 046135 (2001).
2. R. Albert, H. Jeong, and A. Barabási, "Error and attack tolerance of complex networks," *Nature* **406**, 378–382 (2000).
3. R.J. Anderson, "The Eternity service," in *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology*, CTU Publishing House, Prague (1996).
4. D.L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM* **24**(2), 84–88 (1981).
5. I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: a distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*, LNCS 2009, ed. by H. Federrath. Springer: New York (2001).
6. F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *18th ACM Symposium on Operating System Principles (SOSP '01)*, ACM Press (2001).
7. R. Dingledine, M.J. Freedman, and D. Molnar, "The Free Haven project: distributed anonymous storage service," in *Designing Privacy Enhancing Technologies*, LNCS 2009, ed. by H. Federrath. Springer: New York (2001).
8. I. Goldberg and D. Wagner, "TAZ servers and the Rewebber network: enabling anonymous publishing on the world wide web," *First Monday* **3**(4) (1998).
9. T. Hong, "Performance," in *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, ed. by A. Oram. O'Reilly: Sebastopol, CA, USA (2001).
10. M.K. Reiter and A.D. Rubin, "Anonymous web transactions with Crowds," *Communications of the ACM* **42**(2), 32–38 (1999).
11. S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing* **5**(5), 40–49 (2001).
12. J. Rosen, *The Unwanted Gaze: The Destruction of Privacy in America*, Vintage Books (2001).
13. A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *18th ACM Symposium on Operating System Principles (SOSP '01)*, ACM Press (2001).
14. M. Waldman, A.D. Rubin, and L.F. Cranor, "Publius: a robust, tamper-evident, censorship-resistant, web publishing system," in *Proceedings of the Ninth USENIX Security Symposium*, Usenix Association (2000).
15. D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature* **393**, 440–442 (1998).