Protecting Free Expression Online with Freenet

Freenet uses a decentralized P2P architecture to create an uncensorable and secure global information storage system.

Ian Clarke and Scott G. Miller
Uprizer

Theodore W. Hong
Imperial College of Science, Technology, and Medicine

Oskar Sandberg and Brandon Wiley
Freenet Project Inc.

The growth of censorship and erosion of privacy on the Internet increasingly threatens freedom of expression in the digital age. Personal information flows are becoming subject to pervasive monitoring and surveillance, and various state and corporate actors are trying to block access to controversial information and even destroy certain materials altogether. Recent incidents such as the publication of Monica Lewinsky's deleted personal e-mails in a U.S. congressional report further point to an unprecedented level of intrusion into private life.[1] These trends cause concern not only to whistleblowers and political dissidents, but to anyone disturbed by the thought of others reading their e-mail or following their Web activities.

Fortunately, concurrent advances in the power of personal computers have made it possible to develop peer-to-peer technologies to respond to these challenges. Our project, Freenet, is a distributed information storage system designed to address information privacy and survivability concerns.[2] A beta version of the software is currently available under open source at http://www.freenetproject.org/.

In simulations of up to 200,000 nodes, Freenet has proved scalable and fault tolerant. It operates as a self-organizing P2P network that pools unused disk space across potentially hundreds of thousands of desktop computers to create a collaborative virtual file system. To increase network robustness and eliminate single points of failure, Freenet employs a completely decentralized architecture. Given that the P2P environment is inherently untrustworthy and unreliable, we must assume that participants could 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.

Design Motivation

As documented by Human Rights Watch (http://www.hrw.org/advocacy/internet/) and the Global Internet Liberty Campaign (http://www.gilc.org/), governments around the world have undertaken efforts to force Internet service providers to block access to content deemed unsuitable or subversive, or to make them liable for such material hosted on their servers. The Electronic Privacy Information Center (http://www.epic.org/) has also raised privacy and civil liberties questions about developments like the Federal Bureau of Investigation's Carnivore electronic monitoring system and the European Union's new "Convention on Cybercrime," which gives authorities broad powers to intercept and record digital communications.

Though seemingly separate, the prevention of censorship and the maintenance of privacy are both fundamental to free expression in a potentially hostile world. Preserving the availability of controversial information is only half the problem; individuals can often be subject to adverse personal consequences for writing or reading such information and might need to conceal their activity in order to protect themselves. Indeed, the U.S. Supreme Court, among others, has long recognized the important role of anonymous speech in political dissent.

A common objection to mechanisms for secure communication is that criminals might use them to evade law enforcement. Freenet is not particularly attractive for such purposes, as it is designed to broadcast content to the world — not so useful for secret criminal plots. In any case, however, anonymous electronic communication is simply a tool, like payphones or postal mail, to be used for good or bad. A terrorist might use it to plan an attack, or an informant could use it to turn the terrorist in to the authorities. Most importantly, the freedom to communicate is a fundamental value in a democratic society. There is no way to deny it to the "bad guys" without also denying freedom to the "good guys" — civil rights activists, minority religious groups, or ordinary citizens who simply wish to keep their affairs private.

In designing Freenet, we focused on

- privacy for information producers, consumers, and holders;
- resistance to information censorship;
- high availability and reliability through decentralization; and
- efficient, scalable, and adaptive storage and routing.

Maintaining privacy for creating and retrieving files means little without also protecting the files themselves — in particular, keeping their holders hidden from attack. We have thus made it hard to discover exactly which computers store which files. Together with redundant replication of data, holder privacy makes it extremely difficult for censors to block or destroy files on the network.

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. Many systems solve this problem by requiring payment (in disk space or money, for example), but we would rather encourage publishing than keep out authors who can't run peer nodes themselves or are too poor to pay for storage. To keep junk documents from filling all available space or overwrit ing existing data, we implement a probabilistic storage policy. We hope, however, that Freenet will attract sufficient resources from participants to preserve most files indefinitely.

We must assume that participants could operate maliciously or fail without warning.

Freenet 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 file's 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 request's originator.

GUID Keys

Freenet GUID keys are calculated using SHA-1 secure hashes. The network employs two main types of keys: *content-hash keys*, used for primary data storage, and *signed-subspace keys*, intended for higher-level human use. The two are analogous to inodes and filenames in a conventional file system.

Content-hash keys. The content-hash key (CHK) is the low-level data-storage key and is generated by hashing the contents of the file to be stored. This

Related Work in P2P

The best-known systems similar to Freenet 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 whereas Freenet provides a file-storage service, these systems provide a file-sharing service. That is, participants make files available to others but do not push files to other nodes for storage. This architecture means that data is not persistent in the network; rather, files are available only when their originators (or subsequent requesters) are online. Another difference is that neither system attempts to provide anonymity. Gnutella is also extremely inefficient, broadcasting thousands of messages per request.

Freenet more closely resembles the Eternity service, which was described in a proposal for a highly survivable network for permanently and anonymously archiving information.[1] However, the proposal lacked specifics on how to efficiently implement such a service. Free Haven is an Eternity-like anonymous P2P publication system that uses trust mechanisms and file trading to enforce server accountability and user anonymity.[2] Unfortunately, it can take a very long time — even days — to retrieve files from it.

Security Issues

Several recently developed P2P file-storage systems focus on efficient data location rather than privacy and security against malicious participants. Systems such as OceanStore,[3] Cooperative File System (CFS),[4] and PAST[5] are all based on routing models in which each node is assigned a fixed identity and maintains some knowledge of nodes whose identities vary in specified ways from its own. These systems deterministically place data on nodes that most closely match the data's globally unique identifier (GUID). A user can thus locate data by progressively visiting nodes whose identities match more and more bits of the desired GUID. The main advantage to these systems is that they can provide strong guarantees that data will be located within certain time bounds (generally logarithmic) if it exists. Thus, they can provide 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 to gain responsibility for a particular piece of data and suppress it. Links and routing are also more visible and deterministically structured, making it easier to trace messages and harder to route around malicious nodes that sabotage requests (for example, by pretending data could not be found). PAST, as currently constituted, also requires users to trust external smart cards.

Privacy Issues

Systems focusing on privacy for information consumers include browser proxy ser-

continued on p. 43

process gives every file a unique absolute identifier (SHA-1 collisions are considered nearly impossible) that can be verified quickly. Unlike with URLs, you can be certain that a CHK reference will point to the exact file intended. CHKs also permit identical copies of a file inserted by different people to be automatically coalesced because every user will calculate the same key for the file.

Signed-subspace keys. The signed-subspace key (SSK) sets up a personal namespace that anyone can read but only its owner can write to. You could create a subspace for an archive on the Vietnam War, for example, by first generating a random public-private key pair to identify it. To add a file you first choose a short text description, such as `politics/us/pentagon-papers`. You would then calculate the file's 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 subspace's public key (perhaps stored on your "keyring") 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 real-world identity. For example, you can use SSKs to send out a newsletter, to publish a Web site, or (operated in reverse) to receive e-mail.

Typically, SSKs are used to store indirect files containing pointers to CHKs rather than to store data files directly. Indirect files combine the human readability and publisher authentication of SSKs with the fast verification of CHKs. They also allow data to be updated while preserving referential integrity. To perform an update, the data's owner first inserts a new version of the data, which will get a new CHK because the file contents are different. The owner then updates the SSK to point to the new version. The new version will be available by the original SSK, and the old version will remain accessible by the old CHK. Indirect files can also be used to split large files into multiple pieces by inserting each part under a separate CHK and creating an indirect file that points to all the parts.

Related Work in P2P (cont.)

continued from p. 42

vices 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[6] improves anonymity over simple proxying through a request-chaining technique similar to the one we use. None of these systems directly stores information; 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 (temporary anonymous zone) servers[7] extend this idea with chains of nested encrypted URLs that point to successive Rewebber-like servers to be contacted. Neither system protects information producers or provides redundant information storage. Publius[8] enhances robustness and protects producer anonymity by distributing files as redundant partial shares among many holders; however, because 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 protects information consumers, although Rewebber also operates a separate browser proxy service.

References

1. R.J. Anderson, "The Eternity Service," *Proc. 1st Int'l Conf. Theory and Applications of Cryptology*, CTU Publishing House, Prague, Czech Republic, 1996, pp. 242-252.
2. R. Dingledine, M.J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Designing Privacy Enhancing Technologies*, Lecture Notes in Computer Science 2009, Springer-Verlag, Berlin, H. Federrath, ed., 2001, pp. 67-95.
3. S. Rhea et al., "Maintenance-Free Global Data Storage," *IEEE Internet Computing*, vol. 5, no. 5, Sep./Oct. 2001, pp. 40-49.
4. F. Dabek et al., "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating System Principles* (SOSP 2001), ACM Press, New York, 2001.
5. A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. 18th ACM Symp. Operating System Principles* (SOSP 2001), ACM Press, New York, 2001.
6. M.K. Reiter and A.D. Rubin, "Anonymous Web Transactions with Crowds," *Comm. ACM*, vol. 42, no. 2, 1999, pp. 32-38.
7. I. Goldberg and D. Wagner, "TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web," *First Monday*, vol. 3, no. 4, 1998; available at http://www.firstmonday.dk/issues/issue3_4/goldberg/.
8. M. Waldman, A.D. Rubin, and L.F. Cranor, "Publius: A Robust, Tamper-Evident, Censorship-Resistant, Web Publishing System," *Proc. 9th Usenix Security Symp.*, Usenix Assoc., Berkeley, Calif., 2000, pp. 59-72.

Finally, you can use indirect files to create hierarchical namespaces from directory files that point to other files and directories.

SSKs can also be used to implement an alternative domain name system for nodes that change address frequently. Each such node would have its own subspace, and you could contact it by looking up its public key — its *address-resolution key* — to retrieve the current address.

Messaging and Privacy

Freenet was designed from the beginning under the assumption of hostile attack from both inside and out. Therefore, it intentionally makes it difficult for nodes to direct data toward 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.

Privacy in Freenet is maintained using a variation of Chaum's mix-net scheme for anonymous communication.[3] 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.

Because each node in the chain knows only about its immediate neighbors, the end points could be anywhere among the network's hundreds of thousands of nodes, which are continually exchanging indecipherable messages. Not even the node immediately after the sender can tell whether its predecessor was the message's originator or was merely forwarding a message from another node. Similarly, the node immediately before the receiver can't tell whether its successor is the true recipient or will continue to forward it. This arrangement is intended to protect not only information producers and consumers (at the beginning of chains), but also information holders (at the end of chains). By protecting the latter, we can prevent an adversary from destroying a file by attacking all of its holders. Of course, ensuring privacy is not enough; queries must be able to locate data as well.

Routing

Routing queries to data is the most important element of the Freenet system. The simplest routing method, used by services like Napster, is to

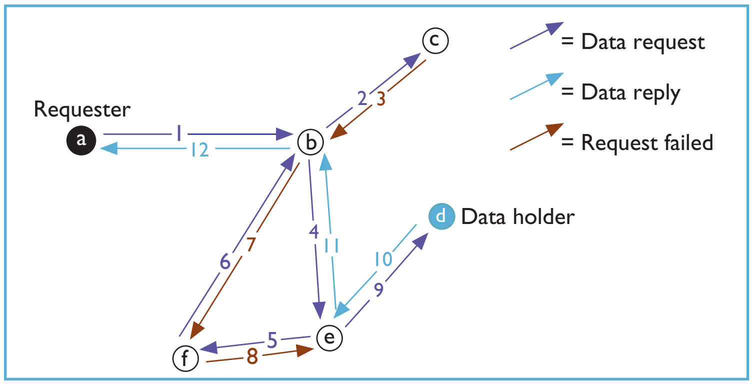

Figure 1. Typical request sequence. The request moves through the network from node to node, backing out of a dead-end (step 3) and a loop (step 7) before locating the desired file.

maintain a central index of files, so that users can send requests directly to information holders. Unfortunately, centralization creates a single point of failure that is easy to attack. For example, if you were trying to phone Michael Jordan, the simplest way to get his number would ordinarily be to call directory assistance. However, because directory assistance is centralized, your access can be easily blocked if Jordan or someone else decides to remove his directory entry, or if the service goes down.

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 Jordan's number, get them to ask their friends, and so on. Within a few steps, thousands of people could be looking for his number. Although this process would eventually find your answer, it 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. You might start searching for Jordan by asking a friend who once played college basketball, for example, who might pass your request on to a former coach, who could pass it to a talent scout, who might pass it to Jordan's agent, who could put you in touch with the man himself.

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 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. That node then checks its store, and so on. 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. Depending on its distance from the holder, each node might also cache a copy locally.

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 holder's 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.

Figure 1 depicts a typical request sequence. The user initiates a request at node *A* and 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 bounces the message back. Unable to contact any additional nodes, node *F* backtracks one step to *E*, which forwards the request to its second choice, *D*, and locates the file. *D* returns the file via *E* and *B* back to *A*, which sends it to the user. Along the way, *E*, *B*, and *A* might also cache the file.

With this approach, the request homes in closer with each hop until the key is found. A subsequent query for this key will tend to approach the first request's path, and a locally cached copy can satisfy the query after the two paths converge. Subsequent queries for similar keys will also jump over intermediate nodes to one that has previously supplied similar data. Nodes that reliably answer queries will be added to more routing tables, and hence, will be contacted more often than nodes that do not.

Inserting files. 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.

To insert a file, a user assigns it a GUID key and sends an insert message to the user's own node containing the new key with a TTL value that represents the number of copies to store. Upon receiving an insert, a node checks its data store to see if the key already exists. If so, the insert fails – 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). In the latter case, the user should choose a different description or perform an update rather than an insert. (Note that we have not yet implemented updates because we are still working on a mechanism to ensure that all old copies get replaced.)

If the key does not already exist in the node's data store, the node looks up the closest key and forwards the message to the corresponding node as it would for a query. If the TTL expires without collision, the final node returns an "all clear" message. The user then sends the data down the path established by the initial insert message. 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. As with requests, if the insert encounters a loop or a dead end, it backtracks to the second-nearest key, then the third-nearest, and so on, until it succeeds.

Data Encryption

For political or legal reasons, node operators might wish to remain ignorant of the contents of their data stores. To this end, we encourage publishers to encrypt all data before insertion. The network proper knows nothing about this level of encryption because it just ships already encrypted bits.

Data encryption keys are not used in routing or included in network messages. Inserters distribute them directly to end users at the same time as the corresponding GUIDs. Thus, node operators cannot read their own files, but users can decrypt them after retrieval. Node operators cannot gain any information by looking at GUIDs, either, because the hashes used to generate them scramble any identifying characteristics. From a node operator's point of view, the data store consists only of random GUIDs attached to opaque data.

Network Evolution

The network evolves over time as new nodes join and existing nodes create new connections after handling queries. As more requests are handled, local knowledge about other nodes in the network improves, and routes adapt to become more accurate without requiring global directories.

Adding Nodes

To join the network, a new node first generates a public-private key pair for itself. This pair serves to logically identify the node and is used to sign a physical address reference. Note that public keys are not certified. We don't need to link them to real-world identities because the node's public key is its identity, even if it changes physical addresses. Certification might be useful in the future for deciding whether to trust a new node, but for now Freenet uses no trust mechanism.

Nodes' routing tables should specialize in handling clusters of similar keys.

Next, the node sends an announcement message including the public key and physical address to an existing node, located through some out-of-band means such as personal communication or lists of nodes posted on the Web, with a user-specified TTL. The receiving node 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 protocol for shared random number generation that prevents any participant from biasing the result. This procedure assigns the new node responsibility for a region of keyspace that all participants agree on while guaranteeing that a malicious node cannot influence the assignment for a specific key that it might want to attack.

Training Routes

As more requests are processed, the network's routing should become better trained. Nodes' routing tables should specialize in handling clusters of similar keys because each node will mostly receive requests for keys that are similar to the keys it is associated with in other nodes' routing tables. When those requests succeed, the node learns about previously unknown nodes that can supply such keys and creates new routing entries for them. As the node gains more experience in handling queries for those keys, it will successfully answer them more often and, in a positive feedback loop, get asked about them 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. Nodes should similarly cluster files cached after requests because most requests will be for similar keys.

Taken together, the twin effects of clustering in routing tables and data stores should improve the effectiveness of future queries in a self-reinforcing cycle. While we do not yet have a good mathematical model to analyze the training and convergence of the Freenet algorithm, the simulations described later show that the network can, in practice, locate files quickly — with a median path length of just 8 hops in a 10,000-node network.

Well-known nodes tend to see more requests and become even better connected.

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.

Suppose, for example, a descriptive string such as `politics/us/pentagon-papers` yields the key AF5EC2. Requests for this file could be satisfied by creating clusters containing the keys AF5EC1, AF5EC2, and AF5EC3, rather than clusters containing works about U.S. politics. In fact, hashes are useful because they ensure that similar works will be scattered throughout the network, lessening the chances that a single node's failure will make an entire category of files unavailable. Similarly, the contents of any given subspace will be scattered across different nodes, which increases robustness.

Searching

One open issue is how users can search the network for relevant keys. This is similar to the problem of searching the Web, and similar solutions are possible: Freenet can be spidered, or individuals can publish lists of bookmarks. However, these approaches are not entirely satisfactory in terms of Freenet's design goals.

One simple approach for a true Freenet search would be to create a special public subspace for indirect keyword files. When authors insert files, they could also insert several indirect files corresponding to search keywords for the original file. The "Pentagon Papers" file might have indirect files named `keyword:politics` and `keyword:united-states` pointing to it, for example.

The system would allow multiple keyword files with the same key to coexist (unlike with normal files), and requests for such keys could return multiple matches. Thus, a search for "politics" might return a pointer to the Tiananmen Papers as well as one to the Pentagon Papers. Managing a large number of indirect files for common keywords would be difficult, however, because all the files with the same name would be attracted to the same nodes. A more sophisticated approach might use some type of distributed search over detailed metadata descriptors inserted along with the original files, but we have not yet devised a way to route such a search efficiently.

Managing Storage

To encourage participation, Freenet does not require payment for inserts or impose restrictions on the amount of data that publishers can insert. Given finite disk space, however, the system must sometimes decide which files to keep. It currently prioritizes space allocation by popularity, as measured by the frequency of requests per file. Each node orders the files in its data store by time of last request, and when a new file arrives that cannot fit in the space available, the node deletes the least recently requested files until there is room.

Because routing table entries are smaller, they can be kept around longer than files. Evicted files don't necessarily disappear right away because the node can respond to a later request for the file using its routing table to contact the original data holder, which might be able to supply another copy. Why would the original holder be more likely to have the file? Freenet's data holder pointers have a treelike structure. Nodes at the leaves might see only a few local requests for a file, but those higher up the tree receive requests from a larger part of the network, which makes their copies more popular.

File distribution is therefore determined by two competing forces: *tree growth* and *pruning*. The query-routing mechanism automatically creates more copies in an area of the network where a file is requested, and the tree grows in that direction. This improves response time and prevents overloading when the popularity of a file increases suddenly. Files that go unrequested in another part of the network are subject to deletion. As that part of the tree shrinks, space is freed up for other files. The net effect is that the number and location of copies adjust to the demand for each file.

Performance Analysis

We have tested Freenet's performance using simulations. We have described more extended results elsewhere,[4] but we will summarize the most important results here. Freenet demonstrates good scalability and fault-tolerance characteristics that can be explained in terms of a *small-world* network model.[5] Small-world networks are characterized by a power-law distribution of graph degree (here, the number of routing table entries) of the general form $p(x) \sim x^{-t}$, where t is a constant, x is the graph degree, and $p(x)$ is the probability that a node has degree x. In such a distribution, the majority of nodes has relatively few local connections to other nodes, but a significant small number of nodes have large wide-ranging sets of connections. Even in very large networks, the small-world topology enables efficient short paths because these well-connected nodes provide shortcuts.

Figure 2 shows the graph degree distribution in a simulation of a 10,000-node trained network. The distribution closely approximates a power law with $t = 1.5$, except for an outlier resulting from the maximum routing table size (250 in this simulation). This is not surprising, as power-law distributions tend to arise naturally when networks grow by preferential attachment (that is, new nodes prefer to connect to nodes that already have many links).[6] 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 well-known nodes tend to see more requests and become even better connected ("the rich get richer").

Scalability

To test Freenet's scalability, we created a simulated network of 20 nodes initially connected in a ring topology. We sent inserts of randomly generated files to random nodes in the network, interspersed with random requests for files that had already been inserted (all with TTL = 20). After every five inserts and requests, we created a new node, which announced itself to a random existing node with TTL = 10. We measured the network's performance after every hundred inserts and requests by issuing a set of test requests for previously inserted files and recording the resulting path length distribution (the number of hops actually required to find the data). This continued until the network reached 200,000 nodes.

Figure 3 shows the evolution of the first, second, and third quartiles of the request path length versus network size, averaged over 10 trials. We

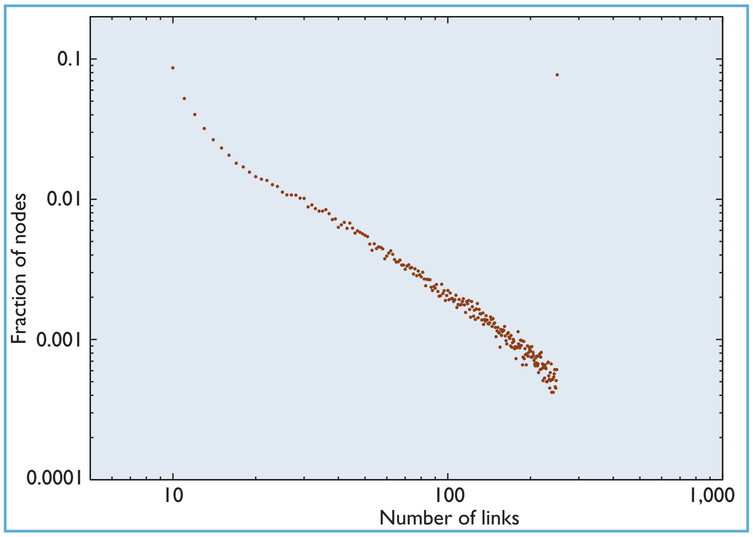

Figure 2. Degree distribution among Freenet nodes. The network shows a close fit to a power-law distribution.

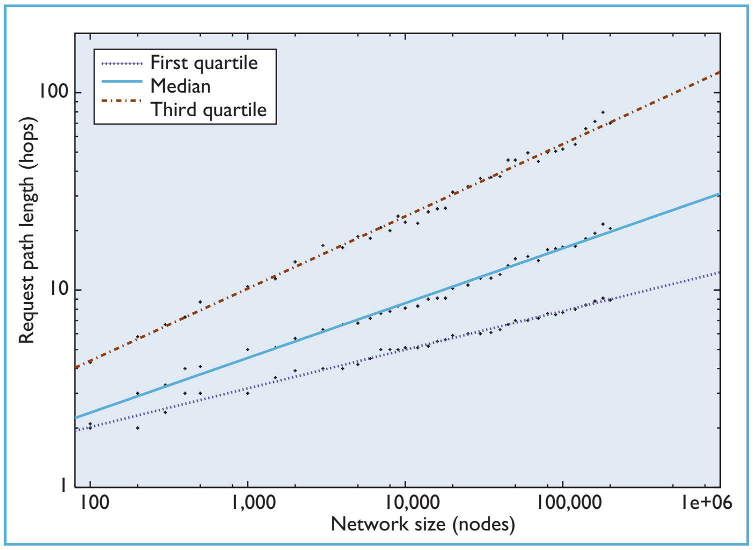

Figure 3. Request path length versus network size. The median path length in the network scales as $N^{0.28}$.

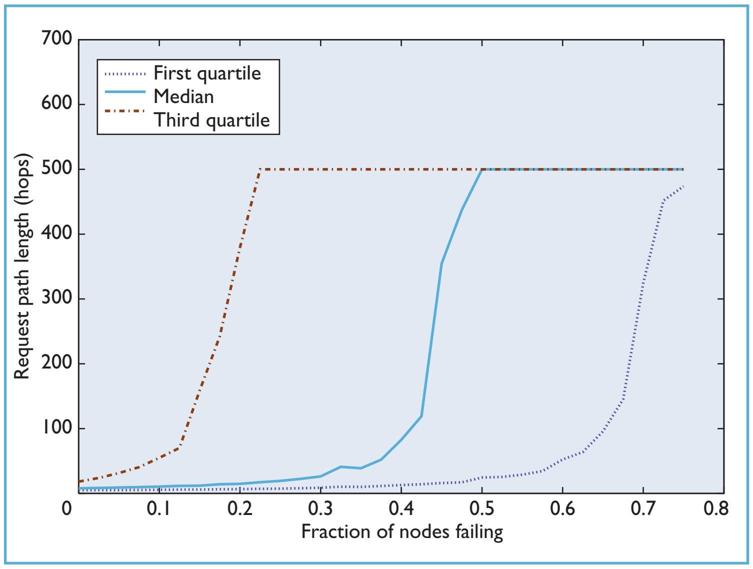

Figure 4. Request path length under random failure. Performance remained reasonable even up to a 30 percent failure rate in our simulation.

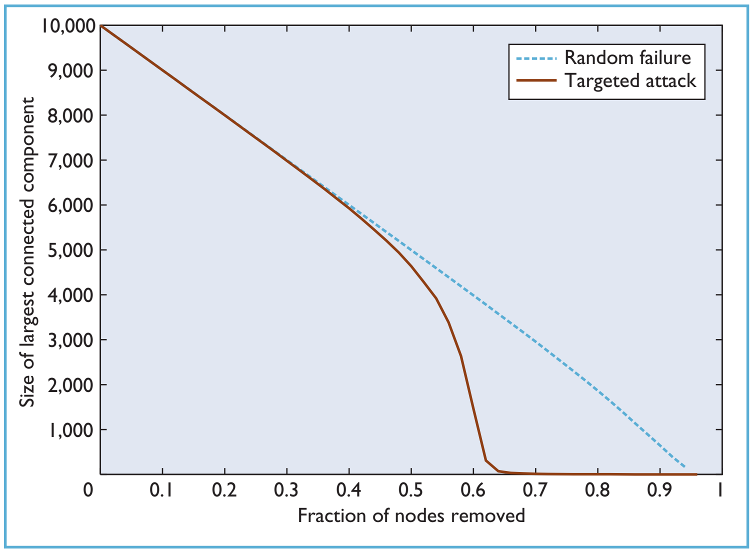

Figure 5. Connectivity under random failure and targeted attack. The network falls apart quickly when the well-connected nodes are targeted first.

can see that the median path length scales sublinearly with network size as $N^{0.28}$, which agrees with recent results in mathematical modeling of peer-to-peer networks.[7] By extrapolation, it appears that Freenet should be capable of scaling to one million nodes with a median path length of just 30.

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 4 shows the resulting evolution of the request path length, averaged over 10 trials, which shows that the network is surprisingly robust against quite large failures. The median path length remained below 20 even when up to 30 percent of nodes failed. (Note that requests were capped at 500 hops before giving up.)

The power-law distribution gives small-world networks a high degree of fault tolerance[6] because random failures are most likely to eliminate nodes from the poorly connected majority. Routing performance is noticeably affected only after there are enough failures to knock out a significant number of well-connected nodes. A small-world network falls apart much more quickly, however, if the well-connected nodes are targeted first. This is evident in Figure 5, which shows the size of the largest connected component in a 10,000-node network as nodes were removed, both randomly and in order from most connected to least connected. Under random failure, the vast majority of the network remained connected until almost the very end. Under targeted attack, the network underwent a "percolation transition" near 60 percent removal, at which point it abruptly broke into disconnected fragments.

Future Work

Initial beta deployment of Freenet is under way, and users have downloaded hundreds of thousands of copies of the software so far. The system's anonymous nature makes it 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 a simulation and visualization suite to enable more rigorous tests of the protocol and routing algorithm. More realistic simulation and formal modeling are needed to explore the effects of nodes joining and leaving, variations in node capacity and bandwidth, and larger network sizes.

We still need to develop search mechanisms and provide more protection against denial-of-service attacks that flood the system with junk data. Although the eviction mechanism works to eliminate files that are never requested, important files could be pushed out if it did not act quickly enough under attack. On the other hand, reducing the priority of new data could result in files being deleted before they have had a chance to be requested. We are exploring various modifications to the caching policy, such as caching less aggressively farther down the data holder pointer tree, to balance these considerations.

Acknowledgments

The third author thanks the Marshall Aid Commemoration Commission for their support. This material is partly based on work supported under a U.S. National Science Foundation graduate research fellowship.

References

1. J. Rosen, *The Unwanted Gaze: The Destruction of Privacy in America*, Vintage Books, New York, 2001.
2. I. Clarke, et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Designing Privacy Enhancing Technologies*, Lecture Notes in Computer Science 2009, H. Federrath, ed., Springer-Verlag, Berlin, 2001, pp. 46-66.
3. D.L. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Comm. ACM*, vol. 24, no. 2, 1981, pp. 84-90.
4. T. Hong, "Performance," in *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, ed., O'Reilly and Assoc., Sebastopol, Calif., 2001, pp. 203-241.
5. D. Watts and S. Strogatz, "Collective Dynamics of 'Small-World' Networks," *Nature*, no. 393, June 1998, pp. 440-442.
6. R. Albert, H. Jeong, and A. Barabási, "Error and Attack Tolerance of Complex Networks," *Nature*, no. 406, July 2000, pp. 378-381.
7. L.A. Adamic et al., "Search in Power-Law Networks," *Physical Rev. E*, vol. 64, no. 4, 2001, article no. 046135.

Ian Clarke is vice president and chief technology officer of Uprizer Inc. He received a BS with honors in computer science and artificial intelligence from the University of Edinburgh, Scotland. He is the original architect and coordinator of the Freenet Project.

Theodore W. Hong is a PhD student at Imperial College, London. His research interests include the dynamics of complex networks, multiagent systems and game theory, and grammatical inference for information extraction. He has appeared as a BBC commentator on digital rights issues. He received an AB in chemistry and physics and mathematics from Harvard, an MSc in microwaves and optoelectronics from University College, London, and was a 1995 Marshall scholar.

Scott G. Miller is a senior software engineer for Uprizer. He received a BS with honors in computer science from Indiana University. His research interests include cryptography and self-organizing systems, which converged in the Freenet Project.

Oskar Sandberg is a core programmer who has been with Freenet since 1999. He is on the board of Freenet Project, Inc. Sandberg is in the final year of his master's degree in mathematics at the University of Stockholm, Sweden.

Readers can contact Clarke at ian@freenetproject.org.