Tarzan: A Peer-to-Peer Anonymizing Network

Anonymous

December 6, 2001

Abstract

Tarzan is a peer-to-peer anonymous network layer, providing generic IP forwarding. Unlike prior anonymizing layers, Tarzan is flexible, transparent, decentralized, and highly scalable. We achieve this by building anonymous IP tunnels between peers. Tarzan can provide anonymity to existing applications such as web browsing, file systems, and similar applications. We show that Tarzan imposes minimal overhead over a corresponding non-anonymous overlay route.

1 Introduction

Anonymity on the Internet has been difficult to achieve in any generalized sense. Systems that provide anonymous communication and transactions on the Internet are highly application-specific and often rely on a small set of core systems to perform the anonymizing service. For example, there are rather few anonymous remailers [6] and this system is only applicable to electronic mail. More general systems such as Onion Routing [14] and Zero-Knowledge's Freedom [7] still rely on a small core set of routers to provide service. Such reliance increases vulnerability to individual node failures and also provides an obvious target for attacks. Additionally, such systems do not necessarily scale well and require special support for many session-layer protocols. A robust free infrastructure which could anonymize any connection would benefit a wide array of distributed systems, since the anonymous connection could seamlessly replace the current connection.

With increased bandwidth and processing power generally available, we believe that a dynamic anonymizing infrastructure can be built out of volunteer peers. By distributing the burden of forwarding traffic among many peers, the effect of any single node failure or compromise is minimized.

This paper describes the Tarzan peer-to-peer anonymizing network layer. Tarzan provides an architecture that enables nodes in the network to use their peers to communicate anonymously with other peers. That is, it allows a sender to communicate with a receiver, without the receiver being aware of the sender's identity. This core allows us to build a transparent and generic IP forwarder, which allows a participant in Tarzan to communicate bidirectionally with an arbitrary Internet host in a way such that traffic analysis can not reveal that the two parties are communicating. Further, with application-level sanitizing proxies, participants can also communicate anonymously with arbitrary hosts when using targetted applications.

By providing anonymous channels, Tarzan can serve as an underlying component for building privacy-protecting systems and for anonymizing existing protocols. We use this network to support a systems-engineering position: anonymity can be built-in as an underlying transport layer, transparent to most systems, trivial to incorporate, and with a tolerable loss of perceived efficiency compared to their non-anonymous counterparts.

1.1 Threat and usage model

The Tarzan design is motivated by what we consider a realistic average-case threat model for

Internet communications. A perfect anonymizing system must defend against many different attacks. Notably, they must protect against a global passive observer that can observe traffic on every link in the network. Naturally, most systems are not perfect [2]. Providing such protection requires cover traffic, mixing, batching, and timing delays, all which greatly reduce the efficiency and practicality of the system. Whereas mixes are viewed as centralized shared resources, we seek a system built of peers which all provide relay services. If peers are widely distributed across different jurisdictional and operational boundaries, we believe that performing wide-spread passive observation is extremely difficult.

Thus, our system chooses to focus instead on the a non-global adversary. Our adversary is able to create a variety of new misbehaving nodes, and he has a limited ability to be a passive observer, by (for example) listening at some ISPs or on transoceanic communication links. However, the adversary will not be able to influence a majority of nodes, or observe all intra-node traffic.

We envision that nodes in the system will consist of several types. Some nodes will be more-or-less constant participants, providing a set of well-known coordination points. This will allow new users to enter the system easily and guarantee that at least some nodes are always operational. In addition, there will be many more transient nodes that attach to the network to use the service and then leave. As with most popular services, there will always be some relatively large number of nodes attached. Tarzan's ability to quickly and correctly detect and use these nodes will give this system its anonymity.

1.2 Outline of paper

The rest of this paper is structured as follows. Our architecture is discussed in detail in Section 2. In addition to high-level design, there are also many relevant policy issues; these are considered in Section 3. We analyze Tarzan's security and anonymity in Section 4. Section 5 describes our implementation and Section 6 evaluates our performance. We present related work in Section 7. Finally, Section 8 conludes.

2 Architecture and design

Tarzan is an anonymizing overlay network layer through which participating nodes can communicate anonymously with non-participants. All participating nodes run relay software that handles node discovery and packet forwarding. Typical use proceeds in three stages. First, the initiating node selects a path in the overlay network, as each relay can ask any other relay to forward its packets. Next, it establishes a tunnel using these nodes, and finally, it routes data packets through this tunnel. An overview of our system is shown in Figure 1.

Tarzan operates purely at the network level and continues to offer the best-effort delivery model. The burden of providing functionality like reliability or authentication is left on the communicating end-hosts.

Tarzan provides unlinkability by using layered encryption: each leg of the tunnel adds or removes a layer of decryption. IP headers are sanitized at the tunnel entry-point.

In addition to the technical core of Tarzan, there are many policy related issues and choices that users of an anonymizing system must consider. The Tarzan design provides mechanisms for users to be aware of such policy issues and allows implementors and clients to make decisions that can best fulfill the desired policies. We defer discussion of these details until Section 3. In the rest of this section, we first describe the process of packet relay; this clarifies the necessary state at each tunnel layer. Next, we show how this state is established during the tunnel setup phase. Finally, we describe how IP forwarding happens at the tunnel endpoints.

2.1 Packet relay

Packet routing is the most common operation for Tarzan relays. Tarzan passes two distinct types of messages between nodes: data pack-

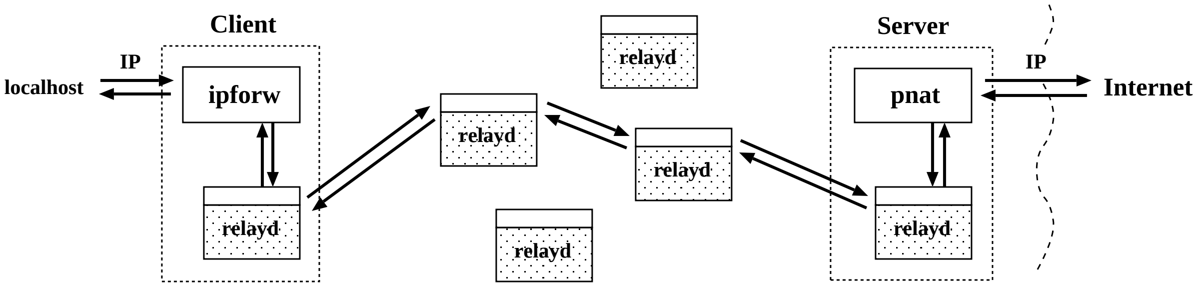

Figure 1: Tarzan Architecture Overview

ets, to be relayed through existing tunnels, and control packets, containing commands and responses which set up and maintain tunnels. Both packet types flow over UDP.

A flow identifier (similar to MPLS [12]) uniquely tags each packet. A relay rapidly determines how to route a packet based on its tag. Symmetric encryption protects data on a per-hop basis, with separate keys being used in each direction each hop.

In the forward path, the tunnel entry-point encapsulates each IP packet into a single encrypted block to be transmitted through a nested encryption under each hop's forward key. More precisely, if the tunnel consists of a sequence of hops $T = (h_1, h_2, \ldots, h_l)$ and the forward keys for each hop are k_{h_i}, the origin produce the encrypted block $\{\{\cdots\{\{p\}_{k_{h_l}}\}_{k_{h_{l-1}}}\cdots\}_{k_{h_2}}\}_{k_{h_1}}$ from the input packet p. The origin tags this block with the first hop's flow identifier and forwards it to h_1. That node's relay will decrypt the data, *i.e.* strip off one layer of encryption, retag the packet, and forward it on to the next hop. This process continues until the packet reaches the last hop, which strips off the innermost layer of encryption, revealing the original IP packet.

On the reverse path, each successive relay performs a single encryption with its appropriate reverse key, retags and forwards the packet back towards the origin. This process wraps the packet in layers of encryption, which the origin of the tunnel must unwrap by performing l decryptions. Note that this design places the bulk of the encryption workload on the node seeking anonymity. To send a packet, that node performs l symmetric encryptions and then performs a corresponding l decryptions when receiving a packet. Nodes that are merely relaying will perform only a single symmetric key operation per packet that is processed.

2.2 Tunnel setup

When forming a tunnel, Tarzan selects a set of relays uniformly at random from all existing relays in the network. We describe how our implementation approximates this process in Section 5. Each relay publishes a public key that is generated locally the first time it enters the network. We rely on this relay being the only one that knows the corresponding private key.

Tunnels are established on an iterative hop-by-hop basis. The tunnel entry-point is responsible for setting up the entire tunnel, which consists mainly of generating and distributing the symmetric keys. The benefit is that the entry-point, which is seeking anonymity, has control over the generation of secure keys.

The high-level establishment algorithm is shown in Figure 2.

An establish request sent to node h_i is relayed as a normal data packet from h_1 through h_{i-1}. Node h_i can not distinguish whether the packet originated from node h_{i-1} or from one of that node's predecessors. The establish request contains the forward decryption key that h_{i-1} will use when sending packets to h_i and the encryption key that should be used for sending packets received from h_{i+1}. Additionally, it establishes the flow identifiers that will be used to

for i *in* $1, \ldots, l-1$*:*
send ***ESTABLISH_REQ***(h_{i-1}, h_{i+1}) *message to* h_i
r = *wait for* ***ESTABLISH_RESP***
while $r \neq$ *ok or timeout*
if max retries exceeded, decrement i *and break*
send ***RESET_FORWARD_REQ***(h_i) *message to* h_{i-1}
r = *wait for* ***RESET_FORWARD_RESP***
send ***ESTABLISH_REQ***(h_l) *message to* h_l

Figure 2: Tunnel Establishment Protocol

tag packets going in each direction. This message uses the public key of node h_i to encrypt the initial forward session key and then this session key to encrypt the subsequent reverse key, node addresses, and flow identifiers. When h_i has successfully stored the state for this request, it responds back to the origin with an establish response to provide an end-to-end check of correctness.

For path length l, this algorithm takes $O(l)$ public-key operations and $O(l^2)$ inter-hop messages to complete. However, for small l, this should not be a major problem.

2.3 IP Packet Forwarding

Tarzan provides a client IP forwarder and a server-side pseudonymous network address translator (PNAT) to create a generic anonymizing IP tunnel. The IP forwarder diverts certain packets from the client's network stack and ships them over a Tarzan tunnel. The client replaces its real address from the packets with a random address from the reserved private address space. The server side NATs this private address to one of its real addresses. Remote hosts can communicate with it normally, as if it originated the traffic. Correspondingly, response packets are deNAT'ed twice, once at each end of the tunnel.

The IP forwarder only hides Internet Protocol address and sequence number information. This is sufficient for building future anonymous systems and anonymizing most existing protocols. For existing application-level protocols which leak information (such as `http`), the client can choose to run a pseudonym manager or application-level sanitizer to protect their identity.

The pseudonymous NAT can also choose to offer port-forwarding to allow incoming connections to the remote endpoint over the anonymous tunnel. This allows hosts on the public Internet to connect to servers that wish to remain anonymous. For example, a user can join a file-sharing network such as Napster as an anonymous server by simply registering her PNAT's address. To Napster, the PNAT would appear to be the client. In fact, any two parties can communicate anonymously by each creating a tunnel to a different PNAT; a normal connection between these two PNATs will form a *double-blinded* channel.

3 Policy issues

3.1 Peer selection

Anonymity policy decisions can affect tunnel selection and maintenance. While Tarzan provides easy and efficient peer discovery, user concerns may direct the actual choice of peers for a tunnel. For example, inter-hop latencies within the tunnel can have a noticeable impact on end-to-end performance. Thus, it may be desirable to select successive tunnel hops such that the latency between each hop is low. On the other hand, a user more concerned with anonymity and privacy may wish to sacrifice some amount of latency in order to ensure that packets are routed through certain points — for example, points outside the jurisdiction of his own government — or even desire hops which explicitly delay and batch messages to increase anonymity.

Latency is a special case since inter-hop latency will be highly dependent upon the underlying network path between any particular pair of hosts. We provide explicit `PING` messages in order to measure per-hop latency through our overlay network. These messages are relayed as normal `DATA` packets until reaching the specified node in the tunnel.

3.2 Route maintenance and lifetime

As the Tarzan network is comprised of unreliable peers, the system must handle the unexpected or malicious failure of nodes. However, using a combination of ping and tunnel establishment/reset packet, we can explicitly identify and route around failures in order to reestablish communication with the tunnel endpoint. This functionality is especially useful when long-standing endpoints are desirable.

As discussed in [11], we might want to change routes only at a low frequency. We return to this security analysis in Section 4.

3.3 Resource control

Although each relay is unaware of the contents of the traffic that passes through it, the relay may wish to impose certain restrictions upon the traffic. For example, a relay may wish to limit the total number of tunnels that it participates in, or the amount of total bandwidth it is willing to allocate to tunnels. We provide some mechanisms for dealing with these issues as well.

We normally require, for example, that all nodes relay data for all other nodes. However, a node may determine that flows from a particular node (or even, one flow from a particular node) are being too aggressive. Such flows can be blacklisted and required to backoff before the relay will continue to forward data on its behalf.

4 Security and anonymity analysis

This section explains how the Tarzan system provides adequate security and anonymity with respect to our goals. As described in Section 1.1, we have designed our system to be anonymous in face of a limited active adversary.

In our design, all participants run a relay, be they client or server. Thus, there are at least as many relays as there are active participants. In the early stages of use, one might imagine that several hundred nodes are participating in the system and later possibly thousands. These nodes will likely be distributed as regular Internet users are distributed.

A limited active adversary can not possibly sniff all the participants in a network. Our security then, is based on selecting nodes at random so that the adversary will not be able to predict and target certain nodes for attack. This is an important part of the design that is handled in an implementation specific manner. Our implementation of Tarzan arranges these nodes into a single Chord ring which we make use of to randomly select hosts to include in tunnels (modulo any policies). Naturally, we rely on the ability of Chord to operate correctly in the face of malicious nodes attempting to divert lookups and routing. So long as Chord is secure, we will in fact be choosing nodes randomly from a large set. On the other hand, a centralized system like Mixmaster provides a small number of nodes to target in attack. Limited targetted attacks are likely easier than wide-spread network sniffing. This divide becomes greater as the number of users increases.

The task of any adversary is also made more difficult because all communication is funnelled through the relay running on each node, regardless of whether it originates locally or remotely. Thus adversaries (be they sniffers or malicious participants) can not distinguish between a node which is merely functioning as a relay and one that is also a client or server.

One large downside of a peer-to-peer system is that an adversary can pseudospoof the system and create a multitude of identities: the number of virtual identities available to one physical adversary is limited only by the number of IP addresses available to him. However, the user can specify policies to avoid routes that have similar IP prefixes.

Tarzan also provides sender anonymity and relationship anonymity. The argument for sender anonymity follows Crowds analysis. However, we differ in two ways. First, Tarzan rebuilds paths following any failure rather then starting over. This minimizes the intersection attack of linked flows that Crowds mentions. Second, linking flows is much harder as data is layered encrypted; in Crowds all nodes in path saw plaintext.

With this in mind, let n be size of network, c the size of the largest set of colluding nodes, and l is path length. Then the probability that a node selected at random is compromised is just $\frac{c}{n}$. Fortunately, our design ensures that each node is chosen independently, so the probability of choosing a completely compromised route is roughly $\left(\frac{c}{n}\right)^l$. Note that there may be more than c adversarial nodes in the system; however, so long as they do not share information, anonymity can not be completely compromised.

Finally, we also provide confidentiality for data as it transits the Tarzan network. The IP forwarder does not provide any confidentiality between the PNAT and the desintation host. If confidentiality at that link is desired, end-to-end encryption must be used. However, this is not a particular goal of the system — internal confidentiality is only used to protect against message coding attacks. Note that the lack of end-to-end encryption means that we do not provide *recipient anonymity* at the IP forwarding level. We do, however, provide recipient anonymity if two peers are communicating using a custom Tarzan application. Sender anonymity requires at least encryption through one hop, which we provide.

5 Implementation

We have implemented Tarzan in `C++` on Unix to validate our approach. SFS libaries [10] provide callback-style functionality for fast asynchronous I/O and automate marshalling data into a standard XDR wire representation.

The core component of Tarzan is a stand-alone relay server which performs the per-hop packet relaying. This server can be run by unprivileged users. Another component, the Tarzan library, communicates with the relay server to establish tunnels, to listen for connections, and to send and receive data. The Tarzan library presents an API modelled after standard Unix sockets, albeit asynchronous.

Applications such as the IP forwarder and the pseudonymous NAT are built on top of this library. For IP forwarding, we take advantage of FreeBSD's divert sockets. Other Tarzan-specific applications can easily be written using this library as well. For instance, the implementation of an anonymous echo client-server was about about 150 lines of code.

Scalable peer discovery and self-organization is accomplished by using the Chord lookup algorithm [13]. All Tarzan relays participate in a single Chord ring. Peers can efficiently locate other peers by generating a random number and finding that number's successor node. The successor will respond with its address, public key, and further information helpful for policy-driven tunnel selection. Each node's relay maintains a Chord peer list and provides its clients with this method for discovering random peers. We rely on Chord's ability to operate correctly while malicious nodes attempt to divert lookups.

6 Performance

The perceived latency of Tarzan tunnels will be highly dependent upon the actual route taking between relays through the underlay network. In this section, however, we examine the actual overhead added by Tarzan's packet handling and crypto processing, as well as packet forwarding rates. Tests were performed on a 100 Mbps switched network with the tested `relayd` running on a 1.2 GHz Athlon box with 128 MB of RAM running FreeBSD 4.3. Our results are summarized in Table 1.

Pkt size	**Latency**	**Throughput**	
(bytes)	(μ-sec)	(pps)	Mbps
64	244	14000	7.168
512	376	8550	35.020
1024	601	7325	60.006

Table 1: Per-hop packet forwarding rate

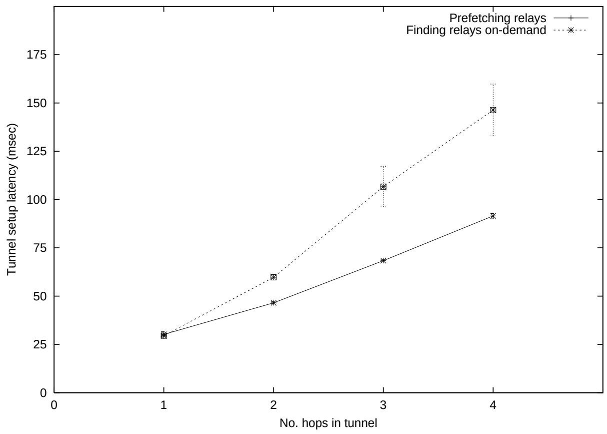

Figure 3: Multi-hop tunnel setup latency

The average latency shown is the time needed for a Tarzan relay to read a packet off the network, decrypt and route it, and send it out to the next relay. For large enough packets, the latency scales linearly with packet size. Throughput also scales roughly linearly.

Tarzan's design criteria for a separate tunnel setup stage, via in-tunnel control packets, versus a packet forwarding stage for end-to-end data is motivated by efficiency concerns for per-packet public-key encryption. While Table 1 shows that a Tarzan relay has reasonable performance for user-level packet forwarding and can easily saturate a T1 line, Figure 3 shows the overhead incurred by setting up multi-hop tunnels using public-key operations.

Latency is measured end-to-end from when the tunnel is initiated on the client, connects iteratively through one to four `relayd` processes run on the relay machine, registers itself with the server-side tunnel end-point application (such as `pnatd`), and received a final acknowledgement. To differentiate the cost of peer lookups through Chord, we show two modes: node information is pre-fetched before the tunnel is initiated in one, while lookups are performed on-demand and factor into the latency in the second. On average, we incur a setup cost of 20 msec per hop.[1] This data suggests that the cost of latency incurred on the underlying network will dominate latency even during tunnel setup.

In summary, a Tarzan relay enjoys a fast packet forwarding rate, high throughput, and reasonable tunnel-setup latency. This measurements support our goal that anonymity can be incorporated at the IP level within a tolerable loss of perceived efficiency compared to their non-anonymous counterparts.

7 Related work

Prior work in this area falls into two categories: systems which provide application-specific anonymity, and systems that offer a more generic transport framework.

Many of the systems we describe are based on mix networks, first proposed by David Chaum [4]. A theoretical mix network is a collection of routers, called mixes, that use layered encryption to encode data and routing information as messages pass through the network. Mixes have public keys that are used by clients seeking anonymity to encrypt messages to be processed by the mix. Clients include a header to indicate the next recipient of the message. A single mix will batch incoming messages, decrypt them, strip them of identifying information, and send them to their indicated recipient. For more security, mixes can be chained (or cascaded) by simply layering this process of encryption and forwarding information. Most systems, including Tarzan, make use of layered encryption and forwarding to ensure that messages look different at each step and thus can not be correlated based on simply comparing contents.

The majority of application-specific anonymous systems focus on email, web browsing, or

[1] The greater variance we see for multi-hop tunnel fetching nodes on-demand is merely a artifact of the small network size, related to the *coupon collector problem.*

file sharing. For example, the mix networks proposed by Chaum [4] were designed to achieve untraceable anonymous email. The Cypherpunk and Mixmaster remailers [6] incorporate these techniques. The latter also provides *reply blocks* for recipients of email to respond to the anonymous senders. Pseudonym servers like `nym.alias.net` [9] automate reply block management. Tarzan's PNAT is conceptually similar to the pseudonym mail servers. However, our PNATs are widely distributed and serve their function for any given node only transiently.

Other application specific systems include those for web browsing, publishing and storage. Web systems range from centralized sanitizers [1] to peer-to-peer systems [11] to those using mixes [3]. Systems for anonymous publishing and storage include the URL-chaining Rewebber system [8], the Publius Censorship Resistant Publishing System [15] and Freenet [5].

Few systems attempt anonymity for low-level, real-time communication. The Onion Routing system [14] creates a mix-net over TCP connections. Application integration is achieved using application proxies that create paths in the system by successively encrypting a control packet, or *onion*. Zero-Knowledge Systems developed the first commercial mix-net system, known as the Freedom network [7]. The Freedom network consists of nodes deployed at various ISPs to route traffic between them, using a model similar to Onion Routing. Client-side integration is closely tied to the operating system and Freedom's pseudonym authentication system. Unfortunately, Freedom was shut down in mid-2001 for financial reasons. Both of these systems only provide plausible unlinkability of sender and recipient.

In comparison, Tarzan uses the same basic idea to mix traffic, but achieves IP-level anonymity by generic and transparent packet forwarding. Tarzan offers *sender anonymity* in addition to the *unlinkability* of sender and recipient provided by Onion Routing and Freedom. This is derived from its peer-to-peer architecture. Tunnel construction is client-driven, allowing users to find more efficient paths by incrementally building tunnels. Tarzan's IP forwarding architecture also allows servers to interact transparently with anonymous clients by performing dynamic pseudonymous network address translation; Onion Routing's proposed *reply onions* are static and thus more vulnerable to node failure, brute force decryption, and even subpoena attacks. Finally, we hope to provide an anonymization tool which is free to use and plan on making our source code available.

8 Conclusion

Tarzan provides a flexible, transparent layer for providing sender anonymity to generic IP connections. We show that Tarzan imposes minimal overhead over a corresponding non-anonymous overlay route. By providing privacy-enabling tools to enable safer and more reliable communication, we hope to reinforce the rights of freedom of speech and freedom of information as integral parts of everyday life.

Acknowledgements

We would like to thank Robert Morris and Frank Dabek for their comments.

References

[1] The Anonymizer. `http://anonymizer.com`.

[2] Berthold, O., Federrath, H., and Köhntopp, M. Project "anonymity and unobservability in the internet.". In *Workshop on Freedom and Privacy by Design / CFP2000* (April 2000). `http://www.inf.tu-dresden.de/~hf2/publ/2000/BeFK2000cfp2000/index.html`.

[3] Berthold, O., Federrath, H., and Köspell, S. Web mixes: A system for anonymous and unobservable internet access. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability* (2001), H. Federrath, Ed., vol. 2009 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 115–129.

[4] CHAUM, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 4*, 2 (February 1982).

[5] CLARKE, I. The Free Network Project. `http://freenet.sourceforge.net/`.

[6] ELECTRONIC FRONTIERS GEORGIA (EFGA). Anonymous remailer information. `http://anon.efga.org/Remailers/`.

[7] GOLDBERG, I., AND SHOSTACK, A. Freedom network 1.0 architecture, November 1999.

[8] GOLDBERG, I., AND WAGNER, D. TAZ servers and the Rewebber network: Enabling anonymous publishing on the World Wide Web. *First Monday 3*, 4 (1998). `http://www.firstmonday.dk/issues/issue3_4/goldberg/`.

[9] MAZIÈRES, D., AND KAASHOEK, M. F. The design, implementation and operation of an email pseudonym server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-5)* (San Francisco, California, November 1998), pp. 27–36.

[10] MAZIÈRES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles* (Dec. 1999), pp. 124–139.

[11] REITER, M. K., AND RUBIN, A. D. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security 1*, 1 (1998), 66–92.

[12] ROSEN, E., VISWANATHAN, A., AND CALLON, R. *Multiprotocol Label Switching Architecture*, Jan. 2001. RFC 3031.

[13] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference* (San Diego, California, August 2001).

[14] SYVERSON, P., GOLDSCHLAG, D. M., AND REED, M. G. Anonymous connections and onion routing. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, California, May 1997), pp. 44–54.

[15] WALDMAN, M., RUBIN, A. D., AND CRANOR, L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium* (August 2000), pp. 59–72.