Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer

Michael J. Freedman, Emil Sit, Josh Cates, Robert Morris
MIT Laboratory for Computer Science
{mfreed,sit,cates,rtm}@pdos.lcs.mit.edu

Abstract

We introduce Tarzan, a peer-to-peer anonymous network layer that provides generic IP forwarding. Unlike prior anonymizing layers, Tarzan is flexible, transparent, decentralized, and highly scalable.

Tarzan achieves these properties by building anonymous IP tunnels between an open-ended set of peers. Tarzan can provide anonymity to existing applications, such as web browsing and file sharing, without change to those applications. Performance tests show that Tarzan imposes minimal overhead over a corresponding non-anonymous overlay route.

1 Introduction

The ultimate goal of Internet anonymization is to allow a host to communicate with a non-participating server in such a manner that *nobody* can determine his identity. Toward this goal, we envision an Internet-wide pool of nodes, numbered in the millions, that relay each others' traffic to gain anonymity. This paper describes a design aimed at realizing that vision. First, however, we discuss why less ambitious approaches are not adequate.

In the simplest alternative to our vision, a host connects to a server through a proxy, such as a Anonymizer.com [1]. This system fails if the proxy reveals a user's identity or an adversary can observe traffic on the proxy's network. Furthermore, servers can block these centralized proxies and adversaries can prevent usage with denial-of-service attacks.

To overcome this single point of failure, a host can connect to a server through a set of mix relays [2]. The anonymous remailer system [4], Onion Routing [9], and Zero-Knowledge's Freedom [5] offer such a model, relying on a small, fixed core set of relays to provide service. Such reliance increases vulnerability to individual node failures, and still provides obvious targets for attacking or blocking. Furthermore, a corrupt relay can perform network-edge traffic analysis on such a system: if the relay receives traffic from a non-core node, that node must be the ultimate origin of the traffic. A corrupt entry relay can conspire with a corrupt exit to determine both source and destination, using timing analysis. An external adversary capable of observing traffic that enters and exits the set of core relays can make the same analysis. This reduces the anonymizing power of long mix paths.

Tarzan, the design presented in this paper, involves sequences of mix relays chosen from a large pool of volunteer participants. All participants are equal peers; they are all potential originators of traffic, as well as potential relays. This design overcomes the edge-analysis weakness: a relay cannot tell if it is the first hop in a mix path. This design is still vulnerable if an adversary can observe traffic throughout the Internet, but this attack seems unlikely.

Tarzan is composed of an open-ended set of participating nodes, with no centralized component; as in other peer-to-peer systems, this lowers the barriers to participation. Tarzan allows client applications on participating hosts to talk to non-participating servers on the Internet. Tarzan is transparent to both client applications and servers, though it must be installed and configured on the client node.

Tarzan routes packets through tunnels involving a randomly chosen sequence of Tarzan peers using mix-style layered encryption. The two ends of a tunnel are a Tarzan node running a client application and a Tarzan node running a network address translator; the latter forwards the client's traffic to the ultimate destination, an ordinary Internet server. These mechanisms provide anonymity in the face of malicious

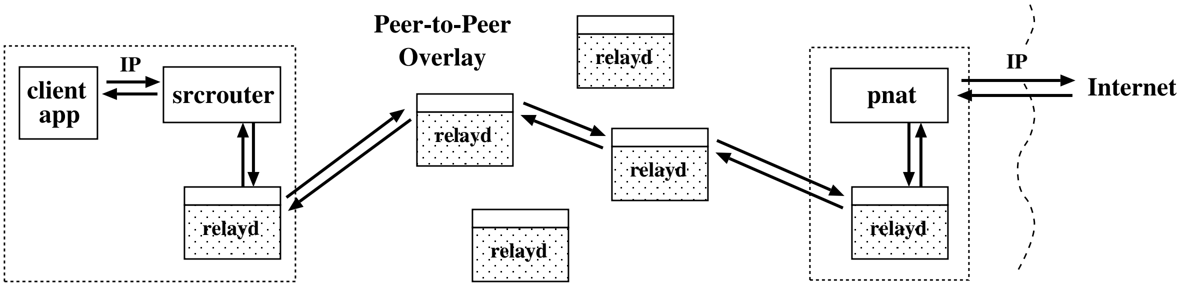

Figure 1: Tarzan Architecture Overview

Tarzan participants, inquisitive Internet servers, and observers who can see traffic on a limited number of network links.

The larger purpose of Tarzan is 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 efficiency. The immediate effect of this approach will be to reduce the effort required for application writers to incorporate anonymity into existing designs, and for users to add anonymity without changing applications. In the long term, the ability for a single anonymizing relay to participate in multiple kinds of traffic may make it easier to achieve a critical mass of anonymizing relays.

2 Architecture and design

This section describes Tarzan's basic tunnel mechanism. Figure 1 shows a simple Tarzan overlay network. All participating nodes run software that 1) discovers other participating nodes, 2) intercepts packets generated by local applications that should be anonymized, 3) manages tunnels through chains of other participants to anonymize these packets, 4) forwards packets to implement other nodes' tunnels, and 5) operates a NAT (network address translator) to forward other participants' packets onto the ordinary Internet.

Typical use proceeds in three stages. First, a node running an application that desires anonymity selects a set of nodes to form a path through the overlay network. Next, this source-routing node establishes a tunnel using these nodes. Finally, it routes data packets through this tunnel. The exit point of the tunnel is a NAT, which forwards the anonymized packets to servers that are not aware of Tarzan.

Tarzan operates at the IP (Internet Protocol) level and offers a best-effort delivery model. The burden of providing functionality like reliability or authentication is left to the communicating end-hosts.

Tarzan uses layered encryption similar to Chaumian mixes [2]: each leg of the tunnel removes or adds a layer of encryption, depending upon the direction of traversal of the packet. IP headers are sanitized at the tunnel entry-point.

The rest of this section first describes how Tarzan nodes relay packets along existing tunnels; this clarifies the necessary per-node tunnel state. Next, it shows how this state is established during the tunnel setup phase, which includes key distribution. Finally, it describes how IP forwarding happens at the tunnel endpoints.

2.1 Packet relay

A Tarzan tunnel passes two distinct types of messages between nodes: data packets, to be relayed through existing tunnels, and control packets, containing commands and responses that set up and maintain tunnels. Tarzan encapsulates both packet types inside UDP.

A flow tag (similar to MPLS) uniquely tags each hop of each tunnel. 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 of each hop.

In the forward path, the tunnel entry-point clears each IP packet's source address field, performs a nested encryption per tunnel hop, and encapsulates the result in a UDP packet. More precisely, if the tunnel consists of a sequence of nodes $T =$

$(h_1, h_2, \ldots, h_l)$ and the forward key for each node is k_{h_i}, the originating node produces 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, re-tags 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.

2.2 Tunnel setup

When forming a tunnel, Tarzan selects a series of nodes uniformly at random from existing peers in the network. 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. Section 2.4 describes how Tarzan selects tunnel nodes and acquires their public keys.

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 encryption keys.

Each hop is set up using the same procedure. 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 cannot distinguish whether the packet originated from node h_{i-1} or from one of that node's predecessors; node h_{i-1} cannot distinguish successive establish requests from ordinary tunneled data. 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 tag packets going in each direction. The initiating node 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 to the origin for 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. This overhead is sufficiently small for realistic choices of l.

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 NATs its own address to a random address assigned by the PNAT from the reserved private address space. The PNAT translates this private address to one of its real addresses. Remote hosts can communicate with PNAT 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 origin port numbers for TCP and UDP packets. For existing application-level protocols that leak information (such as `http`), the client can choose to run an application-level sanitizer.

The pseudonymous NAT can also offer port forwarding to allow ordinary Internet hosts to connect through Tarzan tunnels to servers. This mode of operation provides anonymity to the server. 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.

2.4 Peer selection

Tarzan requires that its peer selection mechanism provide three functions: new peer discovery, scalability, and random selection. Additionally, these mechanisms should be robust against adversaries attempting to bias the selection process.

Tarzan uses the Chord lookup algorithm [8] to obtain this functionality, although and peer-to-peer lookup system that provides these functions would be suitable. Chord is a distributed peer-to-peer hash function mapping flat keys to nodes. Each Chord node has a unique 160-bit node identifier (ID) obtained with a cryptographic hash of its IP address.

All Tarzan relays participate in a single Chord ring. New relays join the ring by contacting an existing relay to discover its proper set of overlay neighbors. We assume that an adversary may only impersonate a limited address space, such as the subnetwork from which he is connected. Therefore, he may only join the network with a relatively small number of Chord nodes, as he cannot respond to RPCs sent to other IP addresses.

Keys are mapped in Chord into the 160-bit space by a universal hash function. The *successor* of a key is the node with the smallest ID greater than or equal to that key (with wrap-around), much as in consistent hashing [6]. The Chord operation, $lookup(K)$, discovers the IP address of the successor of K by iteratively sending RPCs to nodes around the Chord ring until reaching the desired successor.

The system is highly scalable, as the expected number of messages involved in a lookup is $O(\log n)$, for network size n nodes. The iterative lookup allows the peer to validate the existence of every intermediate node.

A peer efficiently picks a random peer by generating a random lookup key and finding that key's successor. The successor responds with its IP address and public key.

This lookup would reveal the tunnel initiator's identity if performed immediately before tunnel establishment. Therefore, every node occasionally performs a random key lookup (say, once per minute) and caches this information for later use. The freshness of the key-to-node mapping is not important; the critical mapping is public key to IP address.

3 Policy issues

Anonymity policy decisions 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, tunnel intra-hop latencies have a noticeable impact on end-to-end performance. However, a user more concerned with anonymity may sacrifice some latency to ensure that packets are routed through certain points – for example, nodes outside his government's jurisdiction – or even desire hops that explicitly delay and batch messages.

While each peer responds to lookup queries with a set of its attributes, intra-hop latency is highly dependent upon the underlay network path. Tarzan provides explicit ping meassages to measure per-hop latency through our overlay network. These messages are relayed as normal data packets in the tunnel until reaching the specified node.

4 Security and anonymity analysis

This section explains how Tarzan provides adequate security and anonymity against a limited active adversary.

A limited active adversary cannot possibly sniff and perform traffic analysis on all system participants. As all Tarzan users run relays, there are at least as many relays as active participants. We imagine several hundred nodes in early stages of deployment and possibly thousands or more in later stages. Because tunnel setup selects its path randomly at runtime (modulo any policies) from the large set of peers, an adversary cannot consequently predict or target nodes for attack.

Additionally, Tarzan confounds adversaries by funneling all communication through each node's relay, regardless of whether the traffic originates locally or remotely. Tarzan achieves *sender anonymity*: both passive sniffers and malicious participants cannot distinguish whether a node initiates a message or merely relays it.

This argument follows the Crowds analysis [7], but differs in several important ways. First, Tarzan rebuilds individual *links* following failures rather than entire *paths*, minimizing the intersection attack on linked flows. Second, adversaries cannot link flows as easily. Tarzan layer-encrypts data in the tunnel, while all nodes in a Crowds path see plaintext. This encryption additionally protects against message coding attacks (and provides data confidentiality in the tunnel as a second-order effect). Third, Tarzan chooses nodes independently. With c colluding adversaries in a n-node network, the probability of choosing a fully-compromised l-length route is roughly $\left(\frac{c}{n}\right)^l$.

A peer-to-peer system also offers new challenges. An adversary can pseudospoof the system and create a multitude of identities: only the number of IP addresses available limits the number of virtual identities usable by him. However, the user can specify policies to avoid routes with similar IP prefixes.

5 Implementation

We have implemented Tarzan in C++ on Unix to validate our approach. The core component of Tarzan is a stand-alone relay server that 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, that is executable on a variety of BSD, Linux, and Unix platforms.

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.

6 Performance

In this section, we present some preliminary performance measurements running on a 1.2 GHz Athlon PC with 128 MB of RAM running FreeBSD 4.3 connected to a 100 Mbps switched ethernet.

Table 1 shows the average latency as the time needed by the Tarzan relay to read a packet off the network, decrypt and route it, and send it out to the next relay. For large-enough packet, latency scales linearly with packet size. Throughput also scales roughly linearly.

Table 2 shows end-to-end latency required to setup a tunnel. To differentiate Tarzan's overhead from the cost of Chord lookups, we provide two measurements. Clients pre-fetch node information in one and perform lookups on-demand in the second. On average, we incur a setup cost of 20 msec per hop. Therefore, underlay network latency still dominates, even during tunnel setup.

Pkt size	**Latency**	**Throughput**	
(bytes)	(μ-sec)	(pkts/s)	(Mbits/s)
64	244	14000	7.2
512	376	8550	35.0
1024	601	7325	60.0

Table 1: Per-hop latency and forwarding rate

Tunnel length	**Pre-fetch latency**	**On-demand latency**
1	30.19	29.51
2	46.54	59.77
3	68.37	106.70
4	91.55	146.37

Table 2: Tunnel setup latency in msec

7 Related work

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

The majority of application-specific anonymous systems focus on email, web browsing, or file sharing. For example, the mix networks proposed by Chaum [2] were designed to achieve untraceable anonymous email. The Cypherpunk and Mixmaster remailers [4] incorporate these techniques. Web systems range from centralized sanitizers [1] to non-mix peer-to-peer systems [7] that lack self-organization. Systems for anonymous publishing include Freenet [3]. In contrast to application-layer solutions, Tarzan provides a single general-purpose anonymizer that can be used transparently by many applications.

Few systems attempt anonymity for low-level, real-time communication. The Onion Routing system [9] 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 [5]. 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, which removes any notion of entry-point into the anonymizing layer. 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 that is free to use and plan on making our source code available under the GNU Public Licence.

8 Conclusion

Tarzan provides a flexible, transparent layer for providing anonymity to generic IP connections. Tarzan's peer-to-peer design makes it decentralized, highly scalable, and easy to manage.

We show that Tarzan imposes minimal overhead over a corresponding non-anonymous overlay route. Latency through Tarzan tunnels is completely dominated by transmission speed through the Internet.

Tarzan's ability as a single anonymizing relay to participate in multiple kinds of traffic furthers its usefulness and, hopefully, adoption.

References

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

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

[3] Ian Clarke, Oscar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer-Verlag, 2001. `http://freenet.sourceforge.net`.

[4] Electronic Frontiers Georgia (EFGA). Anonymous remailer information. `http://anon.efga.org/Remailers/`.

[5] Ian Goldberg and Adam Shostack. Freedom network 1.0 architecture, November 1999.

[6] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Com puting*, pages 654–663, May 1997.

[7] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[8] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[9] Paul Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, May 1997.