Tarzan: A Peer-to-Peer Anonymizing Network Layer

Michael J. Freedman
NYU Dept of Computer Science
715 Broadway #715
New York, NY 10003 USA
mfreed@cs.nyu.edu

Robert Morris
MIT Lab for Computer Science
200 Technology Sq. #509
Cambridge, MA 02139 USA
rtm@lcs.mit.edu

ABSTRACT

Tarzan is a peer-to-peer anonymous IP network overlay. Because it provides IP service, Tarzan is general-purpose and transparent to applications. Organized as a decentralized peer-to-peer overlay, Tarzan is fault-tolerant, highly scalable, and easy to manage.

Tarzan achieves its anonymity with layered encryption and multi-hop routing, much like a Chaumian mix. A message initiator chooses a path of peers pseudo-randomly through a restricted topology in a way that adversaries cannot easily influence. Cover traffic prevents a global observer from using traffic analysis to identify an initiator. Protocols toward unbiased peer-selection offer new directions for distributing trust among untrusted entities.

Tarzan provides anonymity to either clients or servers, without requiring that both participate. In both cases, Tarzan uses a network address translator (NAT) to bridge between Tarzan hosts and oblivious Internet hosts.

Measurements 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 an arbitrary server in such a manner that *nobody* can determine the host's identity. Toward this goal, we envision a system that uses an Internet-wide pool of nodes, numbered in the thousands, to relay each others' traffic to gain anonymity.

Different entities may be interested in exposing the host's identity, each with varying capabilities to do so: curious individuals or groups may run their own participating machines to snoop on traffic; parties skirting legality may break into a limited number of others' machines; and large, powerful organizations may tap and monitor Internet backbones.

Clearly, each type of adversary suggests different design criteria for an anonymizing system. Prior systems have either underestimated the ease of cracking or crashing individual machines, or discounted the prevalence of wide-spread eavesdropping capabilities, exemplified by the "Great Firewall of China" [30], the FBI's Carnivore system [11], or subpoenas of Tier-1 ISP traffic for copyright-protection compliance [22].

CCS'02, November 18–22, 2002, Washington, DC, USA.

This paper describes Tarzan, a practical system aimed at realizing anonymity against all three flavors of adversary. First, however, we discuss why less ambitious approaches are not adequate.

In the simplest alternative, a host sends messages to a server through a proxy, such as Anonymizer.com [1]. This system fails if the proxy reveals a user's identity [18] or if an adversary can observe the proxy's traffic. Furthermore, servers can easily 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 [3]. The anonymous remailer system [10], Onion Routing [26], and Zero-Knowledge's Freedom [13] offer such a model, providing anonymity through a small, fixed core set of relays. However, if a corrupt relay receives traffic from a non-core node, the relay can identify that node as the ultimate origin of the traffic. Colluding entry and exit relays can use timing analysis to determine both source and destination. Even an external adversary can mount the same attack. Therefore, the connecting host remains vulnerable to individual relay failures, and these relays provide obvious targets for attacking or blocking.

Few of these systems attempt to provide anonymity against an adversary that can passively observe all network traffic. Such protection requires fixing traffic patterns or using cover traffic to make such traffic analysis more difficult. Proposals that do exist have several shortcomings, however. Some protect only the core of the static mix network and thus allow traffic analysis on its edges [2, 26]. Some simulate full synchrony and thus trivial DoS attacks halt their operation in entirety [7]. And some require central control and knowledge of the entire network [15].

Tarzan, on the other hand, does not suffer from these same weaknesses. Its main contributions are two-fold.

First, Tarzan extends known mix-net designs to a peer-to-peer environment. Tarzan nodes communicate over sequences of mix relays chosen from an open-ended pool of volunteer nodes, without any centralized component. We present techniques to securely discover and select other nodes as communication relays: All peers are potential originators of traffic; all peers are potential relays. Such a scalable design lessens the significance of targeted attacks and inhibits network-edge analysis, as a relay cannot tell if it is the first hop in a mix path. Furthermore, we leverage our new concept of a *domain* to remove potential adversarial bias: An adversary may run hundreds of virtual machines, yet is unlikely to control hundreds of different IP subnets.

Second, Tarzan introduces a scalable and practical technique for cover traffic that uses a restricted topology for packet routing: Packets can be routed only between *mimics*, or pairs of nodes assigned by the system in a secure and universally-verifiable manner. This technique is practical in that it does not require network synchrony and consumes only a small factor more bandwidth than the data traffic to be hidden, and it is powerful as it shields all network participants, not only core routers.

Tarzan allows client applications on participating hosts to talk to non-participating Internet servers through special IP tunnels. 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 its ultimate Internet destination. Tarzan is transparent to both client applications and servers, though it must be installed and configured on participating nodes.

Tarzan supports a systems-engineering position: anonymity can be built-in at the transport layer, transparent to most systems, trivial to incorporate, and with a tolerable loss of efficiency compared to its non-anonymous counterpart. This approach immediately reduces the effort required for application writers to incorporate anonymity into existing designs, and for users to add anonymity without changing existing non-anonymous applications. In the long term, the ability for individual anonymizing relays to easily participate in multiple kinds of traffic may make it easier to achieve a critical mass of anonymizing relays.

The rest of this paper is structured as follows. Section 2 explains Tarzan's design goals and threat models. Section 3 describes the design of Tarzan: its tunneling architecture, peer discovery and selection protocols, and restricted topology and cover traffic mechanism. Section 4 presents an analysis of Tarzan's anonymity properties. Section 5 describes Tarzan's implementation, and Section 6 evaluates its performance. Section 7 discusses integration transparency, Section 8 describes related work, and Section 9 concludes.

2. DESIGN GOALS AND NETWORK MODEL

This paper uses the following terminology. A *node* is an Internet host's virtual identity in the system, created by running an instantiation of the Tarzan software on a single IP address. A *tunnel* is a virtual circuit for communication spread across an ordered sequence of nodes. A *relay* is a node acting as a packet forwarder as part of a tunnel.

We designed Tarzan to meet a number of goals. Ordered by priority, these goals are the following:

1. **Application independence:** Tarzan should be transparent to existing applications and allow users to interact with existing services. To achieve this, Tarzan should provide the abstraction of an IP tunnel.

2. **Anonymity against malicious nodes:** Tarzan should provide *sender* or *recipient anonymity* against colluding nodes. That is, a particular host should not be uniquely linkable as the sender (recipient) of any message, or that a message should not be linkable to any sender (recipient) [20]. We consider these properties in terms of an *anonymity set*: the set of possible senders of a message. The larger this set, the "more" anonymous an initiator remains.

   These properties implies the weaker *relationship anonymity*: an adversary should not be able to identify a pair of hosts as communicating with each other, irrespective of which host is running Tarzan.

3. **Fault-tolerance and availability:** Tarzan should resist an adversary's attempts to overload the entire system or to block system entry or exit points. Tarzan should minimize the damage any one adversary can cause by running a few compromised machines.

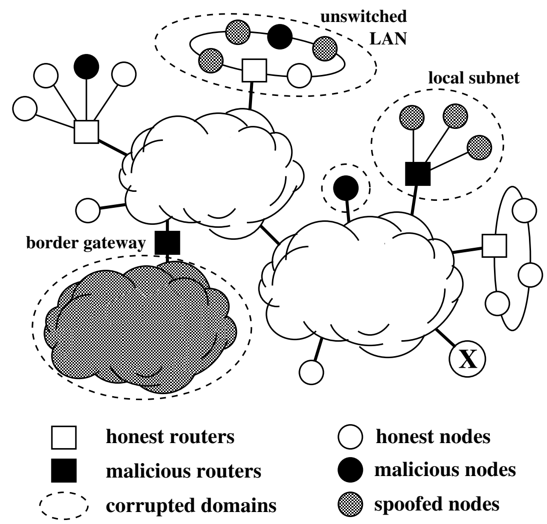

Figure 1: Tarzan network model. In relation to node X, adversarial machines can control address spaces and can spoof virtual nodes within corrupted domains.

4. **Performance:** Tarzan should maximize the performance of tunnel transmission, subject to our anonymity requirements, to make Tarzan a viable IP-level communication channel.

5. **Anonymity against a global eavesdropper:** An adversary observing the *entire* network should be unable to determine which Tarzan relay initiates a particular message. Therefore, a node's traffic patterns should be statistically independent of it originating data traffic.

Because anyone can join Tarzan, the system will likely be targeted by misbehaving users. While a correct host runs only one *honest* node—which forwards packets properly, does not log addressing or timing information, and so on—an adversary can run potentially many malicious nodes or spoof many fake addresses. A node is *malicious* if it modifies, drops, or records packets, analyzes traffic patterns, returns incorrect network information, or otherwise does not properly follow the protocols.

From a naive viewpoint, the fraction of Tarzan nodes that are malicious determines the probability that a tunnel relay is malicious. Yet, a single compromised computer may operate on multiple IP addresses and thus present multiple Tarzan identities.

To defend against such a situation, we make the observation that a single machine likely controls only a *contiguous* range of IP addresses, typically by promiscuously receiving packets addressed to any IP address on a particular LAN or by acting as a gateway router.

This observation is useful in bounding the damage each malicious node can cause. We will call this subnet controllable by a single malicious machine a *domain*.[1]

A node belongs to a $/d$ domain if the node's d-bit IP prefix matches that of the domain. Figure 1 shows the dependence of intra-domain node failure: a malicious machine "owns" all of the address space behind it.

Domains capture some notion of fault-independence: While an adversary can certainly subvert nodes within the same domain in

[1] Our domain notion is completely unrelated to DNS and applies to both IPv4 and IPv6.

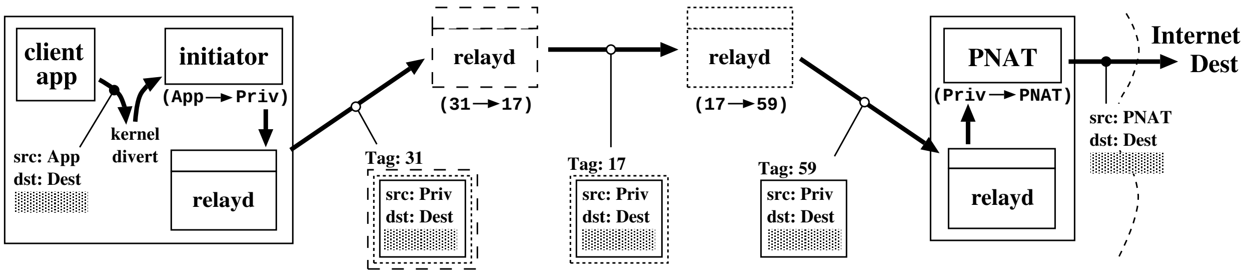

Figure 2: Tarzan Architecture Overview: An IP packet is diverted to the local tunnel initiator, which NATs it to a private address space, wraps it in several layers of encryption, and sends it to the first relay in UDP. Based on the packet's flow tag, the relay decrypts one layer of the encryption and sends the result to the next relay. The PNAT decrypts the last layer, extracts the original IP packet, NATs the packet to its own public address, and writes the raw packet to the Internet.

a dependent fashion, nodes in different domains may fail independently. Therefore, when selecting relays, Tarzan should consider the notion of distinct domains, not that of distinct nodes.

Ideally, we would know the actual size of each domain in address space and count all nodes within that address space as a single entity. However, this internetwork topology is non-uniform and difficult to measure. Therefore, Tarzan chooses some fixed IP prefix size as its granularity for counting domains: first among $/16$ subnet masks, then among $/24$ masks. We believe that this provides a reasonable notion of distinct physical and administrative control.[2]

3. ARCHITECTURE AND DESIGN

This section describes Tarzan's design: its basic tunnel mechanism, its peer-discovery protocol, and its cover-traffic technique.

Figure 2 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 nodes 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, which includes the distribution of session keys. Finally, it routes data packets through this tunnel. The exit point of the tunnel is a NAT. This NAT forwards the anonymized packets to servers that are not aware of Tarzan, and it receives the response packets from these servers and reroutes the packets over this tunnel.

Tarzan restricts route selection to pairs of nodes that use cover traffic to maintain traffic levels independent of data rates. The system enforces this topology by assigning neighbors in a decentralized yet verifiable manner.

Tarzan operates at the IP (Internet Protocol) level and offers a best-effort delivery model. End hosts must provide functionality like reliability or authentication.

Tarzan uses layered encryption similar to Chaumian mixes [3]: each leg of the tunnel removes or adds a layer of encryption, depending upon the packet's direction of traversal. The tunnel initiator sanitizes IP headers, as well as TCP headers if applicable.

[2]Even years since the introduction of CIDR, active Internet addresses still disproportionally belong to network prefixes that reflect classful addressing [17].

3.1 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 establish and maintain these virtual circuits. Tarzan encapsulates both packet types inside UDP.

A flow tag (similar to MPLS [23]) uniquely identifies each link of each tunnel. A relay rapidly determines how to route a packet tag. Symmetric encryption hides data, and a MAC protects its integrity, on a per-relay basis. Separate keys are used in each direction of each relay.

In the forward path, the tunnel initiator clears each IP packet's source address field, performs a nested encoding for each tunnel relay, and encapsulates the result in a UDP packet. More precisely, consider a tunnel that consists of a sequence of nodes $T = (h_1, h_2, \ldots, h_l, h_{pnat})$.[3] Let the forward encryption and integrity keys for each node be ek_{h_i} and ik_{h_i}, respectively, and let seq be the packet sequence number, initiated to zero at the time of tunnel establishment. Then, an initiator produces the following block B_i for each relay h_i in the tunnel, starting with h_{pnat}:

$$\begin{aligned} c_i &= \mathit{ENC}(ek_{h_i}, \{B_{i+1}\}) \\ a_i &= \mathit{MAC}(ik_{h_i}, \{seq, c_i\}) \\ B_i &= \{seq, c_i, a_i\} \end{aligned}$$

The origin tags block B_1 with the first relay's flow identifier and forwards the result to h_1. The first relay extracts the packet's payload, determines the relevant keys by its flow identifier, checks the block's integrity, decrypts the block (*i.e.*, strips off one layer of encryption), retags the resulting block B_2, encapsulates it in a new UDP packet, and forwards the packet on to the next relay. The node drops any packet that fails its integrity check. This process continues until the packet reaches the last relay, which strips off the innermost layer of encryption, revealing the initiator's IP packet.

On the reverse path, each successive relay performs a single encryption with its appropriate key for the reverse direction, 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 $l+1$ decryptions. This design places the bulk of the encryption workload on the node seeking anonymity. Nodes that are merely relaying perform only a single symmetric key operation per packet that is processed.[4]

[3]Section 3.7 explains our strange bookkeeping with the last relay.

[4]Section 3.7 describes an additional encryption-decryption used between immediate nodes.

$h_0 =$ *initiator*;
$h_1 \in_R \{h_0.neighbors\}$;
for $i = 1$ **to** l
    $h_{i+1} \in_R \{h_i.neighbors\}$
    send *establish_request*(h_{i-1}, h_{i+1}) **to** h_i **via tunnel**;
    $rc =$**wait for** *establish_response*;
    if $rc \in \{!ok, timeout\}$
        $i = i - 1$;
        while $rc \in \{!ok, timeout\}$
            if *max retries exceeded*
                decrement i and break;
            $h_{i+1} \in_R \{h_i.neighbors\}$;
            send *reset_forward_request*(h_{i+1}) **to** h_i;
            $rc =$**wait for** *reset_forward_response*;
send *establish_response*(h_l) **to** h_{pnat} **via tunnel**;

Figure 3: Pseudocode for tunnel establishment protocol

3.2 Tunnel setup

When forming a tunnel, a Tarzan node pseudo-randomly selects a series of nodes from the network based on its local topology (see Section 3.7). The initiator is responsible for iteratively setting up the entire tunnel, one relay at a time. This process consists mainly of generating and distributing the symmetric keys, encrypted under the relays' public keys. Section 3.5 describes how an initiator discovers nodes and their corresponding public keys. Each node generates its public key locally the first time it enters the network.

The high-level establishment algorithm is shown in Figure 3. 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 initiating node creates an establish request by using the public key of node h_i to encrypt the initial forward session key, thereafter used to decrypt packets received from h_{i-1}. This session key encrypts the forward integrity key, the subsequent reverse keys for packets from h_{i+1}, the addresses of h_{i-1} and h_{i+1}, and the flow identifiers that will be used to tag packets going in each direction. 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-relay messages to complete. This overhead is sufficiently small for realistic choices of l.

3.3 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 that matches user-specified IP firewall rules and ships them over a Tarzan tunnel. The client forwarder replaces its real address in the packets with 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 the Internet Protocol address and special header fields, such as origin port numbers, for TCP and UDP packets. Section 7 discusses ways of coping with applications that require more work than this to anonymize.

The pseudonymous NAT also offers port forwarding to allow ordinary Internet hosts to connect through Tarzan tunnels to anonymous servers. In fact, to achieve both sender *and* recipient anonymity, any two users can communicate by each creating a tunnel to a different PNAT; each user's application connects to the other's PNAT to form a *double-blinded* channel.

3.4 Tunnel failure and reconstruction

A tunnel fails if one of its relays stops forwarding packets. To detect failure, the initiator regularly sends ping messages to the PNAT through the tunnel and waits for acknowledgments. Upon multiple unsuccessful retries, the initiator attempts to determine the point-of-failure by sending pings through the tunnel to each relay.

If the PNAT is the point-of-failure, *i.e.*, h_l still responds to pings, the initiator selects a new h_{pnat} for the tunnel. Otherwise, it attempts to rebuild the tunnel to the original PNAT, so that higher-level connections, such as TCP, do not die upon tunnel failure.

If the furthest node to reply to the ping is h_i, for $i < l$, intermediate relay h_{i+1} is unreachable. So, the initiator attempts to rebuild the tunnel from h_i forward, $T' = (h_1,\ldots, h_i, h'_{i+1},\ldots, h'_l, h_{pnat})$. Upon multiple unsuccessful attempts, the initiator decrements i by one and reattempts reconstruction.

3.5 Peer discovery

A Tarzan node requires some means to learn about all other nodes in the network, knowing initially only a few other nodes. Anything less than near-complete network information allows an adversary to bias the distribution of a node's neighbor set towards malicious peers, leaks information through combinatorial profiling attacks, and results in inconsistencies during relay selection. Section 4.1 discusses these attacks in more depth.

Tarzan uses a simple gossip-based protocol for peer discovery. Tarzan's goal—to learn about all network resources—differs from recent peer-to-peer lookup protocols [25], which spend great effort to achieve immediate information propagation and load balancing in a flat namespace, often at the cost of security.

Gossiping offers a simple mechanism for nodes to learn about new neighbors.[5] A node can prune inactive neighbors lazily when they do not respond to cover traffic establishment requests, which we explain further in Section 3.7.

This problem can be modeled as a directed graph: vertices represent Tarzan nodes; edges correspond to the relation that node a knows about, and thus can communicate with, node b. Edges are added to the graph as nodes discover other peers. We assume that the graph is initially *weakly connected*; otherwise, nodes in separate network partitions could never learn of one another. Tarzan's peer discovery goal is to make this graph *fully connected*.

Our technique to grow this network graph is similar to the Name-Dropper resource discovery protocol [16]. In each round of Name-Dropper, node a simply contacts one neighbor at random and transfers its entire neighbor set.

The Tarzan discovery protocol supports three related operations: *initialization*, *redirection*, and *maintenance*. Initialization provides the bulk-transfer functionality of Name-Dropper, which allows fast information propagation. Redirection allows nodes to shed load by redirecting new nodes to random neighbors. As this protocol progresses, nodes sending entire neighbor sets will transmit many elements already known to their recipients, wasting bandwidth.

In response, maintenance messages provide an incremental update P of a node's peer database with only new information ($P \cap db = \emptyset$). Tarzan calculates these set differences efficiently by performing k-ary searches on prefix-aggregated hashes of the set elements. This mechanism is briefly described in Section 4.1.

[5]"Gossiping" is a slight misnomer. Traditional gossiping protocols assume a fully-connected or fixed network and seek to optimize the broadcast of extra information, such as link state.

// Let U_a be the set of a's unvalidated known peers
// Let V_a be the set of a's validated known peers
$a.$**gossip**$()$
  while $true$
    if $(U_a = \emptyset), U_a = V_a;$
    $b \in_R U_a;$
    if $(|V_a| < \frac{1}{c}|V_b|)$
      $b.busy \; ? \; a.redirect(b) \; : \; a.initialize(b);$
    else if $(|V_b| < \frac{1}{c}|V_a|)$
      $a.busy \; ? \; b.redirect(a) \; : \; b.initialize(a);$
    else
      $a.maintain(b); \quad b.maintain(a);$

Figure 4: Pseudocode for the peer discovery protocol

Tarzan differentiates between *unvalidated addresses* (U_a) and *validated addresses* (V_a) in a node's peer database. A node learns $\{ipaddr, port, hash(pubkey)\}$ tuples through gossiping: these unvalidated values can easily be forged.

A node validates a tuple once its corresponding peer correctly responds to a discovery request sent directly to its gossiped address. The request includes a random nonce. This two-way network handshake is a weak yet practical authentication mechanism to show a node "speaks for" its address. This validation distinction stops an adversary from injecting arbitrary tuples into a peer database and later impersonating a streak of invalid addresses following it in a tunnel (see Section 4.1).

Figure 4 shows the main gossip protocol. To join the system, a new node a contacts some existing node b to discover a new set of unvalidated addresses. Node a validates b once a receives a response. Node a successively contacts the new neighbors in U_a before retrying neighbors in V_a.

Running the discovery protocol, a node learns about and validates all other nodes in the network in $O(n)$ connections.

3.6 Peer selection

This section describes Tarzan's method for selecting nodes from this peer database.

One may be tempted to simply choose nodes completely at random from V_a. This approach is problematic: if an adversary runs as many Tarzan nodes as IP addresses to which it has access, a user is very likely to select malicious nodes. However, these addresses are rarely scattered uniformly through the IP address space. Instead, they are often located in the same IP prefix space. Thus, we choose among *distinct* IP prefixes, not among all known IP addresses.

We select nodes by choosing randomly among the populated *domains* at each level of the table in Figure 5. Tarzan uses a three-level hierarchy: first among all known /16 subnets, then among /24 subnets belonging to this 16-bit address space, then among the relevant IP addresses.

A node generates this table by inserting all peers in V_a into their corresponding *identifier rings*. The leading d-bits of a node's IP address are transformed to an *identifier* via *hash*$(ipaddr/d,\, date)$, where *hash* is a cryptographic hash function and *date* is the day-of-the-month according to GMT. Identifiers are ordered on their corresponding rings modulo $2^{|id|}$.

Therefore, Tarzan's *lookup*(key) method selects peers as follows: Node a first generates identifier id_{16} via *hash*$(key/16,\, date)$ and finds the smallest identifier $\geq id_{16}$ (with wrap-around) on the /0 identifier ring; it repeats this process recursively with id_{24} and id_{32} on their corresponding rings of increased specificity.

Note that a node executes *lookup* completely locally, based on information already accumulated in its peer database. Therefore, two

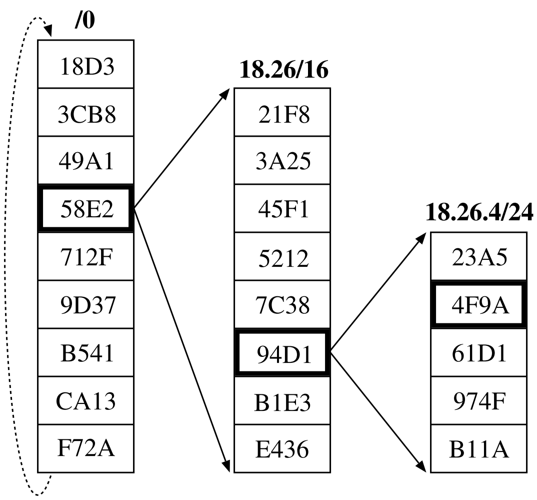

Figure 5: Peer selection on validated nodes. Shown is a *lookup*(key) with id$_{16}$ = 541A, id$_{24}$ = 82F1, and id$_{32}$ = 261B. This ultimately maps to the hash value 4F9A, which yields a node with IP address 18.26.4.9.

nodes may have slightly different lookup structure replicas, which can yield temporarily inconsistent results. We return to the impact of inconsistencies in the next section.

Tarzan includes the date in identifier hashes to daily reorder of ring elements. This randomization stops any particular domain or address from owning a larger space in the ring for any duration. Furthermore, this rebalancing reorders the validated set daily, randomizing how nodes propagate their neighbors during *maintain*.

3.7 Cover traffic and link encoding

If the pattern of inter-node Tarzan traffic varied with usage, a wide-spread eavesdropper could analyze the patterns to link messages to their initiators. Prior work has suggested the use of cover traffic to provide more time-invariant traffic patterns independent of bandwidth demands [3]. Such traffic provides a node with stronger *plausible deniability* that it is the actual message initiator.

Our key contributions include introducing the concept of a traffic *mimic*. We propose traffic invariants between a node and its mimics that protect against information leakage. These invariants require some use of cover traffic and yield an anonymity set exponential in path length.

3.7.1 Selecting mimics

Upon joining the network, node a asks k other nodes to exchange *mimic* traffic with it. Similarly, an expected k nodes select a as they look for their own mimics. Thus, each node has κ mimics, where $E(\kappa) = 2k$ for some global parameter k. Mimics are assigned verifiably at random from the set of nodes in the network.

A node establishes a bidirectional, time-invariant packet stream with a *mimic* node, into which real data can be inserted, indistinguishable from the cover traffic.

This mimic relationship must be symmetric for three reasons. First, a otherwise would send data only on its outgoing links, not trusting its incoming mimic connections. This practice halves a's anonymity set on average. Second, and related, variations in host density behind different IP prefixes may account for some nodes receiving few incoming connections. Second, a otherwise would not be incentivized to provide cover traffic on its incoming links.

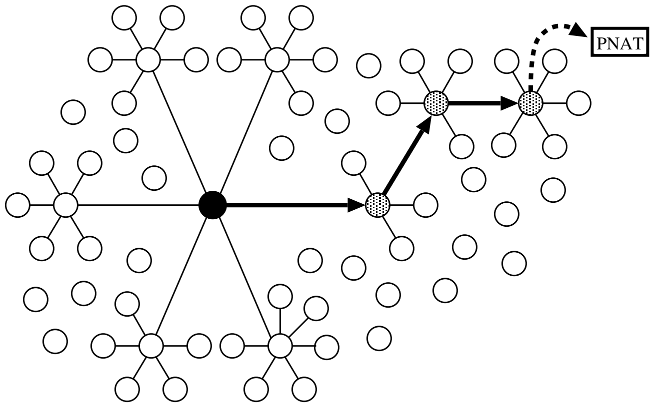

Figure 6: Mimic topology and traffic flows for $k=3$: Each node has $\kappa \approx 6$ mimics (shown connected by solid lines). A node establishes its tunnel over mimic links (arrows in bold) and completes it with a random PNAT (dotted line). Nodes inject cover traffic with data traffic to yield a uniform total traffic $\mathcal{T}$ that flows bidirectionally over the solid lines. All packets on these links are pair-wise encrypted and padded to the same size.

This mimic relationship must be universally verifiable to stop an adversary from selecting more than k mimics. As mimics will be used for tunnel establishment, an adversary could otherwise bias a node's choice of tunnel relays.

Node a chooses its ith mimic, or $\mathcal{M}_i^a$, as the peer returned by *lookup*$^i(a.ipaddr)$. This function is similar to that in Section 3.6, except that the identifier id_d^i is generated by recursively applying the cryptographic hash function i times to $\{a.ipaddr/d, date\}$, $i \leq (k+1)$. Therefore, we map domains to domains. We explain this design choice in Section 4.1.

Node a sends a mimic request, including the tuple $\{a.ipaddr, i\}$, to $\mathcal{M}_i^a$, which we refer to as b. Node b accepts a mimic establishment request from node a if and only if the following hold:

- $1 < i \leq (k+1)$
- $b.$*lookup*$^i(a.ipaddr) = b$

If the *lookup*^{i} check fails, we must consider two cases. First, node b can have a different network view than a: its execution $b.$*lookup*$^i(a.ipaddr)$ maps to a different node. This inconsistency occurs when b knows a ring identifier between $a.ipaddr/d$ and b/d, unknown to a. Node b declines the mimic request but returns the identifier's corresponding node, to which a sends a new request.

Second, a may initially contact node c and receive no response, signifying c's failure. Node a removes c from V_a, and $a.$*lookup*^{i} now maps to b. However, b is not aware of this failure, thus a's new request includes the message that c has failed. To verify, b pings c and waits for the acknowledgment to timeout, at which time b removes c from V_b and thus accepts a's mimic request.

If a looses connectivity to its i^{th} mimic, a removes it from V_a and replaces it with the new node mapped from *lookup*^{i}.

3.7.2 Tunneling through mimics

We constrain a tunnel initiator's choice of relays at each hop to those mimics of the previous hop, instead of allowing it to choose any random node in the network. Therefore, nodes only construct tunnels over links protected by cover traffic. Figure 6 shows this manner of tunnel establishment over mimics.

To initiate a tunnel, node a chooses a mimic $\mathcal{M}_i^a$ as its first tunnel relay. This node $\mathcal{M}_i^a$ has its own mimics $\{\mathcal{M}_1^{\mathcal{M}_i^a} \ldots \mathcal{M}_\kappa^{\mathcal{M}_i^a}\}$, and it returns this list to a. Node a already knows about most of these returned nodes (from gossiping) and can verify them: a's execution of *lookup* with $\mathcal{M}_i^a$'s key should result in the same set. Node a randomly selects node $\mathcal{M}_j^{\mathcal{M}_i^a}$ from the resulting set and continues by tunneling an establish request to this node via $\mathcal{M}_i^a$. It repeats this process for l relays. Finally, a selects a random node for h_{pnat} by *lookup*$(random)$.

If a chose its PNAT in the mimic overlay as well, it could not reconstruct a broken tunnel to this same publicly-addressed node, as no alternative l-length path to the PNAT likely exists. Changing PNATs breaks any application-layer connections over the tunnel.

3.7.3 Unifying traffic patterns

The packet headers, sizes, and rates of a node's incoming traffic from its mimics must be identical to its outgoing traffic, so that an eavesdropper cannot conclude that the node originated a message.

All packets along these mimics links are symmetrically encrypted. This encryption—an additional layer on top of the tunnel encoding—makes cover traffic indistinguishable from data flows. Encrypted packets along these links are padded to be all the same size.[6] A node generates and distributes symmetric keys when it connects with a new mimic.

Our model considers several types of traffic flows in order to properly regulate throughput and cover traffic levels. Data requires a fixed path, *i.e.*, an unsplittable flow. Thus, it places a demand on some *specific* outgoing link. Incoming cover traffic can be dropped on demand or rebalanced on *any* outgoing links. We consider the following traffic flows between a node a and its mimic b:

- $\mathcal{T}_I(b)$: Total incoming rate (data + cover) from b to a
- $\mathcal{T}_O(b)$: Total outgoing rate to b from a
- $\mathcal{D}_O(b, t_i)$: Outgoing data rate to b on a single tunnel t_i

For its behavior to appear innocuous, an honest Tarzan node uses the following relations when generating traffic levels. As before, let $\mathcal{M}^a$ be the set of node a's mimics. Let $\mathcal{T}_I^{\mathcal{M}^a} = \{\mathcal{T}_I(b) : \forall b \in \mathcal{M}^a\}$ be the multiset of total incoming traffic rates from $\mathcal{M}^a$. Thus, $\forall b \in \mathcal{M}^a, \forall t_i \in tunnels(a, b)$:

$$\mathcal{D}_O(b, t_i) \quad \leq \quad f(\mathcal{T}_I^{\mathcal{M}^a}) \tag{1}$$

We choose $f(\cdot)$ as the 33^{rd} percentile of send rates.

Intuitively, Equation 1 limits the outgoing data rate on any link to a function of the incoming traffic rate, such that a malicious mimic could not identify the node as being a clear source of data. However, this function still allows the node to send data at reasonable speeds if up to $\frac{1}{3}$ of its mimics are slow, either maliciously or not.

This data rate is independent (although strictly $\leq$) of the total outgoing traffic level, which itself is constrained, $\forall b \in \mathcal{M}^a$:

$$f(\mathcal{T}_I^{\mathcal{M}^a}) \leq \mathcal{T}_O(b) \leq \max(\mathcal{T}_I^{\mathcal{M}^a}) + \epsilon \tag{2}$$

Ideally, each node provides "enough" traffic to cover its mimics' actions. Per the lower-bound in Equation 2, an honest node maintains a reasonably high level of outgoing traffic to its mimics, in order to cooperatively provide them with anonymity. Additionally, this rate stops a node from being a clear sink of traffic, which would otherwise impact its recipient anonymity.

[6] Our implementation pads to two different sizes as a performance optimization and thus manages two separate queues.

information exposed?	Bad first relay			Bad intermediate relay			Bad last relay			Bad first and last relays		
	OR	Crowds	Tarzan	OR	Crowds	Tarzan	OR	Crowds	Tarzan	OR	Crowds	Tarzan
sender activity	Yes	Maybe	Maybe	No	No	Maybe	No	No	No	Yes	Maybe	Maybe
recipient activity	No	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
sender content	No	Maybe	No	No	No	No	No	No	No	Yes	Maybe	Maybe
recipient content	No	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 1: A comparison of anonymity properties between Onion Routing (OR), Crowds, and Tarzan. Positive table entries mean that the adversary succeeds in its attack and a participant loses some property of its anonymity. Note again that "recipient" refers to a non-participating Internet host; transparent port-forwarding, such as in Tarzan, reverses this notion of sender and recipient.

The upper-bound specifies a maximal rate to prevent the following situation: Node a receives its $f(\cdot)^{th}$ percentile level of total traffic from b and uses this to generate some level of data traffic to mimic c (per Equation 1, a chooses $\mathcal{D}_O(c, t_i) = \mathcal{T}_I(b)$). If node b's own incoming traffic rates are less than its outgoing rate to a, the data received by c could have been initiated by a or b, but not one of b's other mimics. To prevent this, the maximum rate of b's outgoing traffic should be approximate to its own maximal incoming rate; the ϵ term allows a set of honest nodes to cooperatively raise their maximum traffic levels.

Every 20-msec time period, a relay checks its incoming tunnel queues for available data. It reorders packets received out-of-sequence and drops any unexpected or replayed packets, *i.e.,* those ones with sequence numbers smaller than the last packet forwarded. The relay also starts dropping packets once these queues begin to fill. Any congestion control or retransmission is pushed back to the higher-lever protocol between communicating end-hosts.

Once a relay establishes which data packets from the previous period may be forwarded (per Equation 1), it batches some number of these data packets by randomly selecting packets in-order from the available incoming queues (thus mixing between tunnels), creates cover packets to round up to its rate requirement (per Equation 2), inserts the cover randomly into the batched set, and writes the packets to the link-layer.

4. SECURITY ANALYSIS

Per Section 2, Tarzan's anonymity goal is to ensure that a participant has either *sender* or *recipient anonymity* against both colluding nodes and a global eavesdropper. While Tarzan nodes may act as anonymous servers (via port forwarding), we assume here that Tarzan nodes act as clients connecting to oblivious Internet hosts.

Generally, an adversary attempts to break sender anonymity by back-tracing observed messages to their source, usually by carefully watching traffic patterns or message encodings. Similarly, an adversary must trace a message forward to its egress from a PNAT (or control the PNAT itself) to compromise the recipient anonymity of non-participating servers.

This section presents analysis that demonstrate the paper's claims. First, Tarzan's peer-to-peer design exposes less identifying topological information to malicious relays than core-based architectures, yet its decentralized design is still secure. Second, Tarzan resists powerful traffic-analysis attacks without requiring high-latency or bandwidth-inefficient communication.

Therefore, our security analysis first considers the consequences of Tarzan's peer-to-peer model, then discusses general attacks on its cover traffic and anonymous tunneling design.

4.1 On Tarzan's peer-to-peer model

This section shows how Tarzan's peer-to-peer mix-net architecture provides more protection against malicious relays than alternative designs. Second, we consider how Tarzan's handles the security concerns inherent to an open-admission, self-organizing, peer-to-peer model.

Tarzan is the first anonymizing communication system that is both self-organizing and fully-decentralized. Other systems such as the web-specific Crowds system [21] and the "local COR" configuration of Onion Routing [26] propose a peer-to-peer model, yet the former uses a centralized registration and peer discovery server, while the latter does not offer any self-organizing peer-discovery or relay-selection mechanisms.[7] Tarzan's model scales to much larger networks and removes any single or few points-of-failure.

An analysis of Onion Routing's "local COR" configuration in [27] provides a good description of the anonymity benefits of combining a peer-to-peer model with mix-net tunneling. While we have generally described sender and recipient anonymity in terms of hiding the *identity* of a message's sender or recipient, we should differentiate between an adversary's ability to detect the mere presence of activity and the ability to understand the content transmitted: A user surfing the web might only want to prevent an adversary from determining *which* web pages he requests, not the fact that he is making encrypted requests. That is, an adversary detects *sender activity* when it discovers the mere fact that a sender is sending something; we treat messages as black-boxes. The adversary determines *sender content* when it matches sender activity to a particular, readable (glass-box) message. That is, while a Tarzan PNAT can certainly read the clear-text of a message, it reveals sender content only when knows the sender's identity *as well.* We likewise define such properties for recipients.

4.1.1 *Comparing anonymity properties*

Table 1 shows a comparison of Tarzan's model (peer-to-peer, layered encryption) against Onion Routing (network core, layered encryption) and Crowds (peer-to-peer, link encryption only). This direct comparison emphasizes how Tarzan provides better protections against a *static adversary* than other system designs.

A *static adversary* can corrupt some number of independent physical machines *prior* to observing any system behavior. It can read packets addressed to machines under its control; it can analyze the contents, sizes, rates, and volumes of these packets. The adversary can use timing analysis to determine whether packets seen at different relays belong to the same tunnel, but not to estimate the tunnel distance between relays (see Section 4.2.3).

The *relay homogeneity* of the peer-to-peer network model prevents an adversary from deterministically concluding the identity of a sender: All nodes both originate and forward traffic; thus, a compromised relay cannot determine whether it is the first hop in the tunnel. So, it can only guess that its immediate predecessor is the initiator with some confidence, hence the "maybe's" in Table 1 for sender activity and content. On the other hand, a malicious

[7]Therefore, we do not hereafter consider Onion Routing a peer-to-peer system, even while we ignore Crowds' centralized component for demonstrative purposes. Please see Section 8 for related work.

router in a network core system can conclude that a sender is active, with *full* confidence, whenever the client communicates through it.

Crowds [21] provides a good quantitative analysis of the confidence a malicious relay has that its predecessor is the initiator. We extend this analysis to Tarzan in Appendix A; the main difference arises from Tarzan restricting relay choice to source-routing mimics, which additionally explains Tarzan's "maybe" loss of sender activity anonymity with malicious intermediate relays.

The layered (end-to-end) encryption of mix-net systems stops a non-terminal malicious relay from determining the *content* of a message. Thus, it also protects a message's recipient from exposure, which otherwise would be identified through addressing information in the message clear-text. On the other hand, a system that only encrypts messages link-by-link exposes such information to all tunnel relays.

4.1.2 Considering adaptive adversaries

While the previous section discussed the anonymity properties of systems against a *static adversary*, we now concern ourselves with the impact of a *time-bounded adaptive adversary*. In addition to the capabilities of a static adversary, a time-bounded adaptive adversary can pick-and-choose which machines to compromise *after* it joins the system. It may compromise these nodes via buffer overflows and other standard exploits, or even obtain court orders to recover available information through proper legal channels [22]. However, it requires some period of time to compromise any one machine, and it can only control each machine for some period of time until the compromise is detected and subsequently stopped.[8]

If the period to compromise a tunnel's relays is sufficiently small, one cannot achieve anonymity against an adaptive adversary: All entries in Table 1 would be true. An adversary initially controlling any relay can just back- or forward-trace a tunnel to the sender or recipient by successively compromising nodes. The adversary may additionally compromise the nodes with which the sender shares cover traffic, if applicable,[9] to verify that it indeed originates the traffic of interest.

To protect against an adaptive adversary, 1) the period to compromise all tunnel relays must be longer than the tunnel's duration, and 2) tunnels should not be repeatedly constructed through the same small set of largely-compromised relays.

Tarzan's design makes a real effort to ensure that this period to compromise is non-trivial. Its node-selection mechanism of choosing randomly from known domains constructs tunnels that emphasize *host diversity*, which spreads relays across jurisdictional and operational boundaries. Diverse relays will likely run a variety of operating systems and could potentially use different software implementations. Furthermore, an adversary gains little additional information by compromising Tarzan relays *after* their participation in a tunnel: Honest nodes store tunnel keys and routing tables only in memory, and they disable core dumps and process tracing.

Tarzan's scalable architecture offers a large choice of nodes through which to construct tunnels, and Tarzan's mimic reassignment ensures that an initiator's set of possible relays changes daily. Taken together, a new tunnel is unlikely to traverse the same small set of nodes already compromised by the adaptive adversary. Thus, long-term node observation or time-intensive attacks, such as issuing court orders, are less effective. This property does not hold for systems with a central core or a small network, in which a user repeatedly constructs tunnels over the same nodes, and thus the nodes offer clear targets for both long- or short-term compromise.

For the remainder of our discussion of Tarzan's peer-to-peer design, we assume that a tunnel's duration is shorter that the period needed by an adaptive adversary to sequentially compromise the tunnel's relays. In other words, we only consider a static adversary that compromises machines *prior* to tunnel construction, although it can actively attempt to bias an initiator during relay selection.

4.1.3 Defining probability of failure

The attacks shown in Table 1 succeed with the probability that a malicious node occupies the respective tunnel position. How should we analyze this probability? Tarzan's insight into this question led to a domain-driven tunnel construction mechanism that offers additional protections over other systems.

The analyses of most systems, including those of Onion Routing and Crowds, make an obvious independence assumption of adversarial control: if m nodes are malicious in a network (crowd) of size n, any given node is malicious with probability $\frac{m}{n}$. Thus, the first and last hop are together malicious with probability $(\frac{m}{n})^2$.

This assumption is understandable, as we cannot *know* how nodes will experience correlated failures. However, Tarzan's design accounts for how we *predict* that nodes will fail. Thus, our analysis makes the weaker assumption that domains fail independently, and our design reflects this accordingly.

The *node-independence* assumption presents a problem. There are no known techniques to prevent individual entities from presenting multiple identities to the system, without some out-of-band authentication procedure and trusted centralized authority [8, 9]. Thus, without some limit on multiple identities, an adversary with only one host at its disposal can arbitrarily raise $\frac{m}{n}$ toward 1.

Systems may attempt to combat this by exposing relay selection to users [26, 13], in order to encourage host diversity. However, much of the information available to users cannot be authenticated, and it may in fact be used to bias their selection.

Tarzan leverages the following observation. If an adversary compromises M gateway routers or LAN machines that belong to previously-uncorrupted domains, it can then run many malicious nodes (m) within each of these M corrupted domains. As every honest participant runs only one node and assuming these honest nodes are well-distributed, an N-domain system will have a much higher preponderance of nodes within its corrupted domains: $\frac{m}{M} \gg \frac{n}{N}$, thus $\frac{m}{n} \gg \frac{M}{N}$. Although Tarzan's use of domains removes the incentive for such an attack, this fact implies that a random *node* has a higher probability of compromise than a random *domain* in systems that do not consider domains.

For our *domain-independence* claim to be true—an initiator selects a malicious node as a tunnel relay only with probability $\frac{M}{N}$, independent of correlated intra-domain failures—we must first show the following.

CLAIM 1. *A node selects a malicious mimic with prob.* $\frac{M}{N}$.

CLAIM 2. *Nobody can bias an initiator's choice of relays.*

To achieve these strong claims, a node must select its mimics uniformly over the entire set of domains. It also must build tunnels only through other nodes' mimics that are similarly selected in a random and unbiased fashion.

However, malicious nodes may attempt to do the following, in order to bias a's mimic selection and increase its frequency of using malicious relays in its tunnel.

- **Corrupt gossiping:** An adversary gossips addresses that do not exist or only returns malicious nodes.

[8] For example, through the application of periodic security checks and patches by system administrators.

[9] These additional attacks are only necessary against peer-to-peer systems that protect against global eavesdropping through cover traffic. Tarzan is the only such system of which we are aware.

- **Leverage open admission:** An adversary tries to control "important" IP addresses or run multiple nodes.
- **Ignore neighbor-selection algorithm:** A malicious node attempts to select malicious nodes as its mimics, so that honest nodes will build tunnels through sequences of malicious nodes.

The remainder of this section describes how Tarzan protects against these types of attacks.

4.1.4 Securing resource discovery

To bias the distribution of known nodes, an adversary may gossip invalid or only malicious neighbors.

To protect against fake entries, Tarzan differentiates between unvalidated and validated addresses in the peer-discovery and selection process. Honest nodes only propagate validated addresses and only select mimics from their set of validated addresses.

Fake addresses could otherwise be used as placeholders in a tunnel. Consider a malicious node that creates fake addresses and the corresponding key pairs. If an initiator selects this node as tunnel relay h_i, the node can respond successfully to tunnel establishment requests to fake addresses *following* it on the tunnel (*i.e.*, spoof $h_{i+1} \dots h_j$), even though it may not control their relevant domains. If this is the case, the probability that r malicious "relays" are on the tunnel is much greater than $(\frac{M}{N})^r$ and, in fact, approaches $\frac{M}{N}$.

A node must learn and validate all n nodes in the network to prevent a bias towards malicious nodes in its peer database. This bias arises as honest nodes return both honest and malicious nodes, while adversaries return only the latter.

Furthermore, a global eavesdropper that watches nodes gossip can record the α nodes we assume each learns from the total set of size n. Therefore, only α possible nodes could have initiated a tunnel establishment request to a colluding malicious node. In general, for any $\alpha < n$, this discovery-profiling attack leaks some information[10] and thus should be avoided.

Lastly, a node attempts to check the validity of other nodes' mimics during tunnel establishment.[11] If the check fails, the initiator does not consider the mimic for its tunnel, reducing the anonymity-set fan-out to $< (\kappa-1)$ for the hop. For the check to succeed often, the node must also know nearly all nodes in the network.

To handle these security considerations, Tarzan offers an effective and efficient peer-discovery mechanism: A node will discover all n nodes in the network with high probability in connection complexity $O(n)$ and pointer complexity $O(n \log_k n)$.

Tarzan's discovery protocol propagates knowledge the same as Name-Dropper [16]. However, for nodes with neighbor lists of size within a constant factor, node a will only transmit to node b those neighbors not previously known to b.

In Name-Dropper, each node contacts an expected $O(log^2 n)$ other nodes. As a Tarzan node validates its neighbor list by directly contacting every neighbor, we increase the per node connection complexity to $O(n)$.

We measure pointer complexity by both the number of machine addresses and hash values passed during communication. Nodes can determine one set difference briefly as follows. The ith element of a node's sorted set V_a has the hash value $H_i = h(\dots h(h(V_a[1])+V_a[2])\dots+V_a[i])$, generated by recursively aggregating prefixes. If two nodes have the same last hash value $H_{|V_a|}$, their sets are identical and the comparison terminates. Otherwise, the nodes compare their value for $H_{|V_a|/2}$ and proceeds with a binary search. Therefore, nodes can find one set difference in $O(\log n)$ by transferring one hash value per round. We can reduce round complexity to $O(\log_k n)$ on average by sending $(k-1)$ values per round to perform a k-ary search. Thus, *maintain* passes $O(\log_k n)$ pointers to learn one new node.

At least one of a node's initial entry points into the network must be correct; otherwise, all tuples the node learns thereafter could correspond to malicious nodes.

4.1.5 Hardening the open admissions policy

An entity can present multiple identities in Tarzan by using different public keys or running multiple instances of Tarzan on different IP addresses. As stated in Section 4.1.3, Tarzan provides practical measures to lessen the impact of such attacks. First, nodes distribute their keys indirectly through a gossiping protocol. Second, tunnel initiators choose mimics (and thus relays) by selecting uniformly at random from among available *domains*.

First, consider a poor alternative to Tarzan's key-distribution via gossiping: node a learns b's public key by directly contacting it. If b is malicious, it could create a new public key per request and store a mapping from this key to the requester. Later, upon receiving a tunnel establishment message, b could use this mapping to resolve the identity of the tunnel's initiator (based on which key the message is encrypted under). A Tarzan node distributes its public key through gossip channels beyond its direct control. Such indirection protects a tunnel initiator against this key-mapping attack.

Second, consider alternative approaches to peer selection: There is a trade-off between security and network performance. One approach is a local variant of Chord's distributed hash table [25], which maps IP addresses onto a single /32 identifier ring via a hash function. $lookup(random)$ selects some random element from this ring. This approach has obvious load-balancing benefits, but suffers from the admission control problem [9]: An adversary's multiple addresses are distributed randomly around the ring, each having an equal probability of being selected as honest nodes.

Alternatively, a node can simply map IP addresses onto a ring, rather than their hash values. Or, we can still use Tarzan's hierarchical identifier rings, but change $lookup^i$ to hash the node's IP address for each $/d$ ring, as opposed to hashing its d-bit IP prefix. These approaches protect a node in the outgoing direction, as it selects mimics in the adversary's contiguous IP addresses (domains) with low probability. However, an adversary running many nodes—in only a constant number of domains—can create many outgoing links to honest nodes. Thus, honest nodes will have more mimics than the expected $2k$, of which many are malicious.

Emphasizing security, Tarzan maps domains to domains, such that an adversary can influence neither a node's outgoing links nor its incoming ones. Furthermore, as domain prefixes map to random locations on the identifier ring, we assign equivalent importance to all individual /16 prefixes. Therefore, an adversary does not benefit by running multiple intra-domain nodes.

We recognize that a highly-skewed distribution of host densities within $/16$ prefixes will result in poor network performance for mismatched domains. Still, for the smaller network sizes that Tarzan considers, we believe that this distribution will be closer toward uniformity. For instance, real deployment may show Tarzan nodes run on NAT boxes for trusted local area networks.

[10] The expected number of nodes that know about a particular node sequence of length r is $n\binom{\alpha}{r}/\binom{n}{r}$. As Tarzan constrains tunnel selection to mimics, this tighter bound is $(\kappa-1)^{l-r}\binom{\alpha}{r}/\binom{n}{r} \approx (\kappa-1)^{l-r}\alpha^r/n^r$. To uniquely identify the initiator, an eavesdropper needs this value to be ≤ 1, or $\alpha \leq n/(\kappa-1)^{\frac{l-r}{r}}$.

[11] The initiator performs a mimic *lookup* on its own peer database as if it is the node in question, in order to check the correctness of the node's mimic relationships.

4.1.6 *Enforcing proper mimic selection*

An initiator should construct its tunnel only through nodes selected in an unbiased and random fashion. Merely asking the potentially-malicious relay for its mimics is certainly not sufficient: As an initiator extends its tunnel recursively, it must be able to verify a relay's proper mimics.

Let us consider when an initiator accepts some node w as one of malicious node u's mimics. This result occurs when the initiator's *lookup* on u's key K_i yields w, which happens in one of two cases. First, the initiator might not know of some node v which lies between K_i and w on the identifier ring. However, this contradicts that the initiator knows all n nodes in the network with high probability (per Section 4.1.4). Second, node w is actually a properly-selected mimic.

A similar proof by contradiction shows that mimic selection is secure. A node trusts the k mimics it chooses itself. It can also trust the expected k mimics it accepts from other nodes' requests. Otherwise, it would know about some other currently-available node v between the lookup keys and itself.

4.2 Traffic analysis attacks

The previous section's analysis assumes that an adversary can gain no additional information through traffic analysis that would otherwise allow it to compromise Tarzan's anonymity properties. This section discusses how Tarzan 1) resists powerful traffic analysis on the Tarzan overlay itself and 2) prevents information leakage at its exit points, yet 3) may be susceptible to some attacks that use additional application-layer information.

4.2.1 *Information leakage in tunnels*

Existing provably-secure anonymous communication algorithms assume synchrony [3, 4], but a practical system must operate asynchronously. Thus, practical systems must hide discernible, asynchronous events in background statistical noise. Approaches include mixing the order of received packets, batching packets to ensure a sufficient pool size for mixing, delaying packets at relays, and sending cover traffic to hide discernible traffic patterns.

If an eavesdropper can observe all network traffic, relay homogeneity alone does not provide sender or relationship anonymity. The eavesdropper can back-trace traffic flows from an observed PNAT to yield a set of potential senders. If most of these nodes have little activity at that exact time, the adversary may uniquely identify the sender with high probability.

Cover traffic (dummy traffic) that is indistinguishable from data can prevent such analysis. First, an eavesdropper cannot determine whether a relay initiates new data or just sends cover packets. Second, an observer must back-trace the multiple sources of a node's incoming traffic, creating a fan-out of possible senders.

Different mix-network topologies provide various benefits and challenges for achieving anonymity against a global eavesdropper. If an initiator has complete freedom to randomly build tunnels, its anonymity set is naively $(n-m)$. However, such systems cannot guarantee incoming traffic flows to any particular node. Explicitly providing cover traffic across all n^2 links is prohibitively expensive.

In contrast, Tarzan's mimic mechanism explicitly assigns node pairs to exchange cover traffic. This fixed allocation makes the generation of cover more practical, at the cost of a loss of freedom for tunnel initiators.

How may misbehaving mimics affect the protections offered by cover traffic? A misbehaving mimic b may not send traffic at the appropriate incoming packet rate $\mathcal{T}_I(b)$, either through malice or bandwidth limitations. So, Equation 1 balances the DoS threat from slow mimics with the risk of reducing fan-out from malicious mimics: It sets the rate of any individual outgoing data flow $\mathcal{D}_O(h, t_i)$ to at most the 33^{rd} percentile of incoming traffic rates.

If node a rate-limits $\mathcal{D}_O(h, t_i)$ by the minimum incoming rate, node b could trivially DoS a by not sending any traffic (as in [7]). On the other hand, if a limits $\mathcal{D}_O(b, t_i)$ only by the maximum incoming rate, the node's fan-out reduces to the one maximal mimic. This mimic is likely b itself, and thus a may originate data faster than could be explained by its incoming traffic from honest mimics. (This attack succeeds against the current design when malicious nodes comprise two-thirds $(1-f(\cdot))$ of a's mimics and a originates data faster than any incoming traffic from its honest mimics.)

In short, Tarzan selects its rate equations to maintain a "sufficient" outgoing data rate, even if one-third of mimics are slow or malicious. Tarzan's cover traffic mechanism offers protection against traffic analysis of message volume or content, against message flooding, and against DoS attacks of slowing incoming rates.

4.2.2 *Information leakage at network exit points*

While cover traffic and layered encryption protects data traffic within tunnels, an adversary may attempt to leverage the fact that data exits the Tarzan network in clear-text. These network-edge attacks include packet replay, tagging, reordering, and flooding. They generally require an adversary to control some node or link on a tunnel and to observe a PNAT or the non-participating Internet host of interest.

While Tarzan is less susceptible to such attacks on its *sender anonymity* due to its lack of any entrance points, an adversary may attempt such attacks to reduce its *recipient anonymity* or to partially back-trace a packet to reduce a sender's anonymity set. This section briefly describes Tarzan's protections against such attacks.

A Tarzan node uses packet sequence numbers to drop replayed packets. Otherwise, an adversary could replay packets and try to find corresponding activity at non-Tarzan recipients. Tarzan includes an integrity check (*i.e.*, the MAC) to protect against a node or active router from changing the sequence number and defeating our replay protection. Similarly, the MAC prevents an adversary from tagging the packet in such a way that the packet could be identified once it exits the Tarzan network, for example, by flipping some bits in the payload such that TCP checksums are invalidated, given that packet transmission generally introduces few errors.

Tarzan also protects against reordering attacks. As the IP layer usually transmits packets in order, an adversary can reorder TCP packets in a way identifiable once their TCP sequence numbers are in clear-text. While mixing packets alone may initially seem sufficient, an adversary can always reorder a sequence longer than that batched and mixed at nodes. Therefore, a Tarzan node buffers incoming packets and only batches packets for mixing and forwarding in-order. It drops any later packets received out-of-order.

Lastly, an adversary may attempt to flood a node with packets. It hopes to reduce the number of other senders that can simultaneously use the relay, and then try to identify its own outgoing packets from those of others. Tarzan greatly reduces the effectiveness of such flooding. First, mimics encrypt messages between them, making it difficult for an adversary to identify its own packets. Second, cover traffic cannot be distinguished from the legitimate traffic of other nodes. Third, the rigid structure of the mimic overlay limits the set of nodes that an adversary attack: A malicious node can only flood mimics through a well-formed tunnel in the overlay.

4.2.3 *Information leakage from application-layer*

Designed at the IP-layer, Tarzan promises no real protection against information that leaks through application-layer interaction. Even so, its design helps limit such information.

First, an adversary may time application-layer events, such as the period between queries and responses, to estimate the tunnel length between itself and the initiator. Good results would change our characterization of Tarzan's anonymity in Table 1. However, we expect it difficult to estimate tunnel length beyond a margin of error $\pm$ several hops, as 1) wire propagation delays dominate transmission time (see Section 6), and 2) intermediate relays delay packets for short random periods as a second-order effect of ensuring proper traffic rates each transmission round. An initiator can always add additional random delays to its own packets, although this practice obviously impacts performance. Furthermore, application-layer anonymizing support at the PNAT or initiator can defend against such attacks, *e.g.*, Crowds suggests a PNAT web proxy to defeat automatic HTTP re-requests [21].

Second, an adversary may use users' behavior to trace them by observation over a long period. An analysis of anonymity degradation by this *intersection* attack is presented in [31]. If tunnel relays collude, the adversaries require $2\left(\frac{N}{M}\right)^r \ln n$ *rounds* to uncover the initiator's identity with high probability, where $r = 2$ if relays can perform timing analysis to determine they belong to the same tunnel, *i.e.*, only the first relay and PNAT need be malicious.

A new *round* begins when an initiator constructs a completely new tunnel that an adversary can deterministically link to the initiator's previous tunnels, through content, timing, or behavioral information leaked by the application-layer. As the network grows in size or usage, linking tunnels may become increasingly difficult.

We note that Tarzan's tunnel reconstruction protocol offers some protection against intersection attacks: Intermediate nodes will keep reappearing on reconstructed tunnels, yet an adversary cannot differentiate between reconstructed tunnels and new tunnels. This reappearance is viewed as a weakness in [31], which argues that an initiator can continually reuse its first hop to implicate it in intersection attacks. We view this capability as a strength, as an adversary has less confidence in its conclusion.

5. IMPLEMENTATION

We have implemented Tarzan's basic tunneling mechanism in C++ on Unix to validate our approach. SFS libraries [19] provide callback-style functionality for fast asynchronous I/O and automate marshalling data into the standard XDR wire representation.

Tarzan's core component is a stand-alone relay server which performs the per-hop packet relaying. This relay daemon can be run by unprivileged users, as can be the peer discovery daemon.

Tarzan's source-routing IP forwarder and its pseudonymous NAT require root permissions to make raw socket calls. For IP forwarding, we take advantage of FreeBSD divert sockets.

These IP forwarder and PNAT components are built on top of the Tarzan library, which communicates with the relay server to establish tunnels, to listen for incoming connections, and to send and receive data. The Tarzan library presents an API modeled after standard Unix sockets, albeit asynchronous, and is executable on a variety of BSD, Linux, and Unix platforms.

Other Tarzan-aware applications can easily be written using this library as well. For example, the implementation of an anonymous echo client-server is about 150 lines of code. Or, one may wish to modify the PNAT to require access control via SSL certificates and use Tarzan tunnels as an anonymous VPN.

All built-in cryptographic operations are implemented by the SFS `crypt` library. The payloads of all Tarzan packets, which include encapsulated IP headers in data packets, are encrypted in CBC mode with a 20-byte Blowfish [24] key. Each packet includes an 8-byte random initialization vector. A pseudo-random generator based on DSS generates symmetric keys, IVs, and flow identifier tags. See [19] for details of the seeding mechanism. Tarzan uses SHA-1 for all cryptographic hash operations and the Rabin public-key cryptosystem to encrypt the forward-path session key for tunnel establishment. The Rabin implementation is secure against adaptive chosen-ciphertext attacks and is plaintext-aware.

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

Table 2: Per-hop latency and forwarding rate. Packet sizes shown are the actual number of bytes transmitted: packets are not padded to Tarzan's standardized lengths.

Tunnel length	**Setup latency**	**Variance (1 StD)**
1	30.19	1.38
2	46.54	0.53
3	68.37	0.73
4	91.55	1.20

Table 3: Setup latency (msec) and its variance for the construction of multi-hop tunnels through pre-defined relays.

6. PERFORMANCE

This section examines the overhead added by Tarzan's packet handling and crypto processing, as well as its packet forwarding rates. Tunnel latency is lower-bounded by the propagation delay of Internet routes between Tarzan relays.

We performed controlled-environment tests on a 100 Mbps switched network with the tested relay daemon running on a 1.2 GHz Athlon box with 128 MB of RAM running FreeBSD 4.3. Tables 2 and 3 summarize our results.

Table 2 shows the average time that a Tarzan relay needs to read a packet off the network, decrypt and route it, and send it out to the next relay. These preliminary measurements only examine basic data packet handling and do not reflect the overhead of cover traffic and packet batching. We calculated latency using `tcpdump` measurements. For large enough packets, the latency scales linearly with packet size. Throughput also scales roughly linearly. As shown, a Tarzan relay has reasonable performance for user-level packet forwarding and can easily saturate a T1 line, if not limited by its cover traffic algorithms.

The authors have personal experience using multi-hop Tarzan tunnels within a LAN. We experienced no noticeable delays while web-surfing and only slightly perceivable delays, although acceptable, while running the ssh protocol over Tarzan.

Setup latency is measured end-to-end from when a client initiates the tunnel, connects iteratively through one to four relay processes run on the relay machine, registers itself with the server-side tunnel end-point application (such as a PNAT), and receives a final acknowledgment. On average, we incur a setup cost of 20 msec per hop. This data suggests that the cost of latency incurred by 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. As each relay adds a packet-handling overhead of less than 1 msec, propagation delays through the underlying Internet route will completely dominate tunnel latency. Therefore, route efficiency will pervade any performance measurements.

These measurements support our goal that anonymity can be incorporated at the IP level within a tolerable loss of efficiency compared to its non-anonymous counterpart.

7. INTEGRATION

Tarzan's primary goal is to factor out anonymity in the design of larger systems. That is, one should be able to take an existing non-anonymous application that uses IP (or UDP/IP or TCP/IP), and make it anonymous by substituting the Tarzan layer for IP. For this to work, Tarzan should be as transparent as possible, and should integrate painlessly with applications and higher-level protocols.

7.1 Transport-layer protocols

To a first order, Tarzan acts just like an IP layer from the point of view of higher-level protocols such as TCP or UDP. The main difference is that Tarzan uses a network address translator at the exit point. Thus, it inherits the non-transparent aspects of NATs. For example, if a NAT fails, connections already set up through it will also fail.

7.2 Application-layer protocols

One can transparently anonymize an existing protocol with Tarzan only if the protocol reveals the client's identity solely in the packet headers. Protocols that put identifying information in packet payloads need additional work to anonymize.

For example, HTTP headers include identifying information such as page referrals and cookies. However, web browsers and plug-ins offer cookie-blocking tools, as well as the ability to turn Javascript off. Users may use a simple web proxy to sanitize HTTP headers. Tarzan offers the piece still lacking with these tools: a method to strongly anonymize IP address.

Similarly, email SMTP headers include a list of servers through which a message has been routed. While a sanitizing proxy could also scrub this information, pseudonymous email would be more convenient, since it handles replies. A user could simply connect to any pseudonymous email service through Tarzan, such as the free web-based Hotmail system.

Tarzan can transparently anonymize other important protocols. DNS requests do not identify the senders. Login authentication mechanisms, such as telnet, ssh, and SSL, do not leak additional information; Tarzan could be used for pseudonymous identification. Instant messaging systems such as ICQ and AIM implement their own namespaces at the application-layer without leaking personally-identifying information. Tarzan already handles some protocols that explicitly encode IP address and port information in packet payloads, by using existing NAT code which takes care of this for common protocols such as FTP and IRC.

7.3 Software systems

Beyond anonymizing existing protocols, we designed Tarzan as a building-block for anonymous systems. Section 8 mentions several such publishing and file-sharing systems. Systems like Freenet [5] make architectural design decisions for anonymity that impact functionality and performance. For example, the current Freenet implementation (version 0.4.4) always rewrites the source addresses of replies, likely for anonymity reasons. This decreases the ability of Freenet nodes to build efficient routing tables.

Good design practices suggest that such systems should use a layered approach to peer-to-peer file-sharing. For instance, CFS [6] separates its file-system block store from its Chord lookup mechanism. Similarly, Tarzan could serve as its underlying communications layer. Free Haven [8] and Tangler [28] specifically cite the need for a system like Tarzan for their anonymity requirements.

8. RELATED WORK

Prior work in anonymizing asynchronous systems falls into two categories: systems which 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, Chaum [3] proposed mix networks to achieve untraceable anonymous email. The Cypherpunk and Mixmaster remailers [10], which use Chaumian mixes, are also only useful for email. The latter additionally provides *reply blocks* for recipients of email to respond to the anonymous senders; Tarzan offers similar functionality with its PNATs. Web systems range from centralized sanitizers [1] to the Crowds peer-to-peer system [21] to those using mixes [2]. Systems for anonymous publishing and storage include the URL-chaining Rewebber system [14], Publius [29], and Freenet [5]. Each of these systems provides its own particular anonymity design, interwoven with the design of the application-specific protocol. In contrast, 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 [26] creates a mix-net over sequential TCP connections, and Zero-Knowledge Systems developed the first commercial mix-net system, known as the Freedom network [13] (which stopped service in mid-2001). Tarzan differs from these systems in a few key ways.

Application integration is achieved in Onion Routing using separate application proxies; they propose at least sixteen such proxies. However, a TCP-based anonymizer is less-suited for UDP-based protocols such as DNS that provide retransmission at the application-layer. Freedom supports only a few protocols. Operating at the network level by diverting packets using standard firewall rules, Tarzan can anonymize client traffic with fewer application-specific components.

Onion Routing uses a fixed set of onion routers with static public keys. The Freedom network consists of commercially-run nodes deployed at various ISPs to route traffic between them. Freedom centrally manages and provisions the network, building a PKI for router authentication. Tarzan's peer-to-peer architecture requires no central management or control. Nodes can join and leave the network dynamically; we place no requirements on authentication to provide anonymity.

Freedom ties the network into a centrally-managed pseudonym authentication system. Tarzan does not place such barriers at the network level. Instead, the Tarzan library would allow one to build custom server-side applications for access-control or other higher-level considerations.

Tarzan's peer-to-peer architecture removes any notion of entry-point into the anonymizing layer. This topology allows us to provide cover traffic mechanisms that are both practical and powerful: Tarzan offers *sender* and *recipient anonymity* in addition to the *relationship anonymity* provided by Onion Routing and Freedom, without their susceptibility to network-edge attacks.

Both Onion Routing and Freedom expose router selection to the end user. Tarzan automates randomized relay selection and tunnel construction. Furthermore, Tarzan can route around failures for ongoing connections.

Onion Routing's proposed *reply onions* are static, slow, and more vulnerable to node failure, brute force decryption, and even subpoena attacks. Tarzan's IP forwarding architecture allows anonymous servers to interact transparently with clients by performing dynamic pseudonymous network address translation and port forwarding. This design is fast, flexible, and forward anonymous.

9. CONCLUSION

Tarzan provides a flexible layer for sender, recipient, and relationship anonymity, even when operating in real-time. It sustains such anonymity in hostile environments, against both malicious participants and global eavesdroppers.

Tarzan operates transparently at the IP level, so it works with any Internet application. Its peer-to-peer design makes it decentralized, highly scalable, and easy to manage. Additionally, this lack of any network core increases its fault-tolerance to individual relay failure, benefiting both anonymity and availability.

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

Tarzan's ability to participate in multiple kinds of traffic furthers its usefulness and, hopefully, adoption. We believe a common and open-source infrastructure like this will help to make Internet anonymization easier and more prevalent.

By providing privacy-enabling tools for anonymous communication, we hope to reinforce the rights of freedom of speech and freedom of information as integral parts of everyday life.

Acknowledgments

We thank Emil Sit and Josh Cates for significant contribution to the software design and implementation of Tarzan tunnels; David Mazières for helpful discussions regarding security design and analyses; David Karger and Charles Leiserson for useful pointers concerning the peer discovery protocol; and David Andersen, Richard Clayton, George Danezis, Roger Dingledine, Wei Dai, and David Molnar for comments.

10. REFERENCES

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

[2] BERTHOLD, O., FEDERRATH, H., AND KÖSPELL, S. Web mixes: A system for anonymous and unobservable internet access. In Federrath [12], pp. 115–129.

[3] CHAUM, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (February 1981).

[4] CHAUM, D. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology 1* (1988), 65–75.

[5] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In Federrath [12], pp. 46–66. `http://freenet.sourceforge.net`.

[6] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (Banff, Canada, 2001).

[7] DAI, W. Pipenet. `http://www.eskimo.com/~weidai/pipenet.txt`.

[8] DINGLEDINE, R., FREEDMAN, M. J., AND MOLNAR, D. The Free Haven Project: Distributed anonymous storage service. In Federrath [12], pp. 67–95. `http://freehaven.net`.

[9] DOUCEUR, J. R. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)* (Cambridge, MA, March 2002).

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

[11] FEDERAL BUREAU OF INVESTIGATIONS. Carnivore diagnostic tool. `http://www.fbi.gov/hq/lab/carnivore/carnivore.htm`.

[12] FEDERRATH, H., Ed. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability* (2001), vol. 2009 of *Lecture Notes in Computer Science*, Springer-Verlag.

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

[14] GOLDBERG, I., AND WAGNER, D. TAZ servers and the Rewebber network: Enabling anonymous publishing on the World Wide Web. *First Monday 3*, 4 (1998).

[15] GUAN, Y., LI, C., XUAN, D., BETTATI, R., AND ZHAO, W. Preventing traffic analysis for real-time communication networks. In *Proceedings of Milcom '99* (November 1999).

[16] HARCHOL-BALTER, M., LEIGHTON, T., AND LEWIN, D. Resource discovery in distributed networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '99)* (Atlanta, Georgia, 1999).

[17] KOHLER, E., LI, J., PAXSON, V., AND SHENKER, S. Observed structure of addresses in IP traffic. In *Proceedings of the SIGCOMM Internet Measurement Workshop 2002* (Marseille, France, November 2002).

[18] MARTIN, D., AND SCHULMAN, A. Deanonymizing users of the safeweb anonymizing service. In *Proceedings of the 11th USENIX Security Symposium* (San Francisco, California, August 2002).

[19] 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* (Kiawah Island, South Carolina, Dec. 1999), pp. 124–139.

[20] PFITZMANN, A., AND KÖHNTOPP, M. Anonymity, unobservability and pseudonymity — a proposal for terminology. In Federrath [12], pp. 1–9.

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

[22] RIAA V. VERIZON. Motion to enforce July 24, 2002 subpoena. U.S. Distinct Court, District of Columbia, August 20, 2002. Case No. 1:02MS00323.

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

[24] SCHNEIER, B. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Lecture Notes in Computer Science 809* (1994), 191–204.

[25] 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).

[26] 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.

[27] SYVERSON, P., TSUDIK, G., REED, M., AND LANDWEHR, C. Towards an analysis of onion routing security. In Federrath [12], pp. 96–114.

[28] WALDMAN, M., AND MAZIÈRES, D. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security* (Philadelphia, Pennsylvania, November 2001).

[29] 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* (Denver, Colorado, August 2000), pp. 59–72.

[30] WALTON, G. China's Golden Shield: Corporations and the development of surveillance technology in the People's Republic of China, 2001. `http://go.openflows.org/`.

[31] WRIGHT, M., ADLER, M., LEVINE, B. N., AND SHIELDS, C. An analysis of the degradation of anonymous protocols. In *Network and Distributed System Security Symposium* (San Diego, California, February 2002).

APPENDIX

A. MEASURING ANONYMITY

This appendix presents some quantitative analysis supporting our negative goal—that an adversary cannot compromise a participant's anonymity—by considering Tarzan's properties against the theoretical static adversary. Given this model, as described in Section 4.1, we calculate the confidence with which a malicious node can identify any given node as the tunnel initiator a. These results allow us to quantify the "maybe's" in Table 1. Crowds [21] asks a similar question and provides the model for our analysis.

In a core-based architecture, a malicious relay can certainly determine when it receives a message from a non-core node and can thus detect sender activity, even though it may not know the message's contents or destination. Tarzan's *relay homogeneity*, on the other hand, does not expose this information. While a malicious Tarzan relay *can* distinguish between cover and data traffic from its predecessor, it cannot *deterministically* conclude whether this node is initiating the traffic (and is thus a) or merely relaying the data. Given that initiator a chooses a tunnel length randomly through several potentially malicious nodes, what is the adversary's *confidence* that its predecessor is the initiator? Or that one of the possible predecessors of its predecessor is the initiator? And so on.

To derive this probability, we first assume that an initiator selects malicious relays with independent probability $\frac{M}{N}$, as discussed in Section 4.1.3. The malicious relay could conclude deterministically that its predecessor is a if it knew both the tunnel's length l and that $(l-1)$ relays succeed it on the tunnel. However, as Tarzan allows variable path lengths, l takes some probability distribution and our analysis becomes one of conditional probabilities.

Let H_i $(i \geq 1)$ denote the event that the first malicious relay on the tunnel appears at the ith position on the tunnel. Define $H_{i+} = H_i \wedge H_{i+1} \wedge \ldots \wedge H_l$ to be the event that the first malicious relay h_{i+} appears at or after position i.

If the initiator chooses whether to extend the tunnel length by flipping a (biased) coin as in Crowds, the probability that the tunnel is extended by one more hop is independent of its existing tunnel length. Thus, the malicious relay h_{i+} can gain no additional information if it learns how many relays *follow* it.

However, as Tarzan allows users to select tunnels whose lengths likely take other distributions, $Pr(l \geq k+r | l \geq r) < Pr(l \geq k)$ unlike before. Thus, h_{i+} gains additional information by determining that some number of relays follow it, call this number r. We note that colluding relays do not need to sequentially follow h_{i+} to determine whether they are on the same tunnel, assuming they can use timing analysis to reach this conclusion. In fact, h_{i+} can determine r even when only one in three of these relays is malicious, as each malicious relay knows its two possibly-honest neighbors. Let $E(r)$ be the expected number of such relays as known to h_{i+}.

The probability that the tunnel first wanders $(i-1)$ honest nodes is $(\frac{N-M}{N})^{i-1}$ multiplied by the probability the tunnel is at least $(i-1+E(r))$ hops long, given that it is at least $E(r)$ hops.

$$Pr\Big(H_i\Big) \quad = \quad \left(\frac{Pr(l \geq i-1+E(r))}{Pr(l \geq E(r))}\right)\left(\frac{N-M}{N}\right)^{i-1}\left(\frac{M}{N}\right)$$

$$Pr\Big(H_i+\Big) \quad = \quad \sum_{k=i-1}^{\infty}\left(\frac{Pr(l \geq k+E(r))}{Pr(l \geq E(r))}\right)\left(\frac{N-M}{N}\right)^{k}\left(\frac{M}{N}\right)$$

Let I_j $(i \geq j \geq 1)$ be the event that a node j hops preceding the malicious relay h_{i+} on the tunnel is actually the initiator. For example, I_1 is true if and only if the malicious relay's predecessor is the initiator, and I_2 is true only when one of this predecessor's other mimics is the initiator. Generally, I_j is true only when the node at position $(i-j)$ on the tunnel is the initiator.

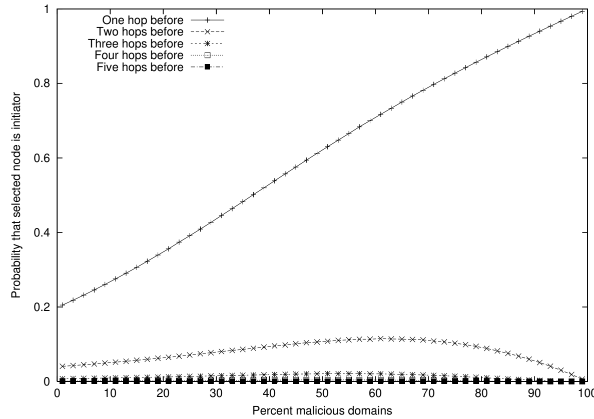

Figure 7: The malicious relay's confidence that a specific relay i hops before it is the tunnel's initiator, given $\kappa = 6$ and tunnel length takes a Lognormal distribution with median 4, σ 0.5.

As Tarzan initiators will never choose themselves as intermediate relays in their own tunnels, unlike Crowds, the initiator is always at position $i = 0$. Thus, $\forall j > i, Pr(I_i | H_j) = 0$; in other words, $Pr(I_i | H_{(i+1)+}) = 0$. Therefore, I_j is true if and only if $j = i$, and $H_i \Rightarrow I_i$. The probability that an adversary observes an initiator is

$$Pr(I_i) \quad = \quad Pr(H_i)Pr(I_i|H_i) = Pr(H_i)$$

Thus, given that a malicious relay h_{i+} appears *somewhere* in a tunnel at or after position i, the adversary's confidence that *some* node preceding it by i hops is the tunnel initiator is $Pr(I_i | H_{i+})$.

$$Pr(I_i|H_{i+}) = \frac{Pr(I_i)Pr(H_{i+}|I_i)}{Pr(H_{i+})} = \frac{Pr(I_i)}{Pr(H_{i+})} = \frac{Pr(H_i)}{Pr(H_{i+})}$$

Lastly, consider that I_i merely identifies the *position* of the initiating node in a tunnel. For $i = 1$, h_{i+} uniquely identifies its immediate predecessor. This is the problem that Crowds considers; I_2 is not interesting as all other nodes in the crowd are equi-likely to precede h_{2+}'s predecessor, provided that we assume an adversary cannot perform traffic analysis. However, Tarzan constrains an initiator's selection to mimics to make cover traffic feasible. Thus, for I_2, the number of possible initiators two hops prior to h_{2+} in the mimic overlay is exactly the set of its predecessor's *honest* mimics. Generally, I_i applies to the set of nodes that precede h_{i+} by i hops, which is precisely the initiator's *anonymity set* AS_i.

$$E(|AS_i|) \quad = \quad \left(\left(\frac{N-M}{N}\right)\Big(\kappa - 1\Big)\right)^{i-1}$$

Assuming that h_{i+} assigns all nodes in AS_i to have an equal probability of being the initiator, let C_i be h_{i+}'s confidence that a *specific* node from this set is the initiator:

$$C_i \quad = \quad \frac{Pr(I_i|H_{i+})}{E(|AS_i|)}$$

Figure 7 shows the malicious relay's confidence C_i, for $i = 1 \ldots 5$, plotted against increasing adversarial compromise of the network. We model $L(\cdot)$ as a Lognormal (heavy-tailed) distribution and assume that the network is large enough such that all nodes in AS_i are unique. Note that as $\frac{M}{N} \rightarrow 1, C_1 = Pr(I_1|H_{1+}) \rightarrow 1$.