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].

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.

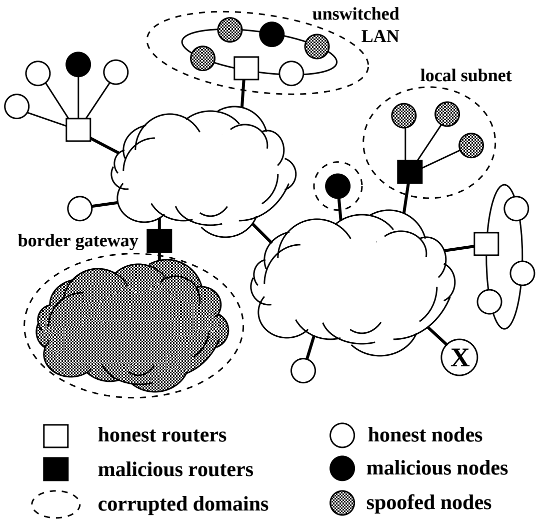

Figure 1: Tarzan network model. In relation to node X, adversarial machines can control address spaces and can spoof virtual nodes within corrupted domains.

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.

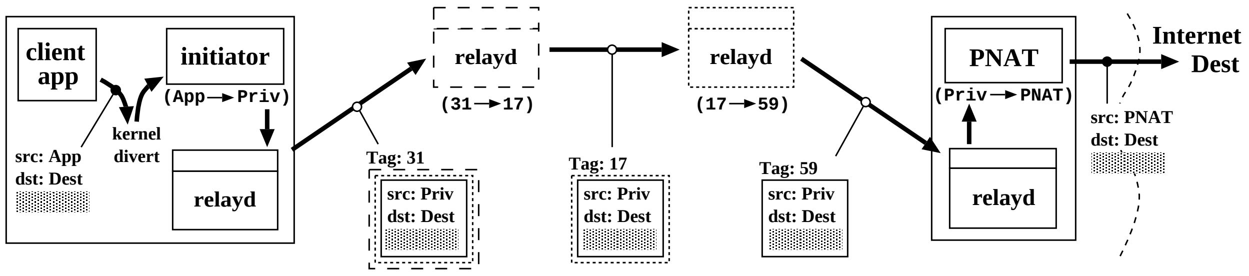

Figure 2: Tarzan Architecture Overview: An IP packet is diverted to the local tunnel initiator, which NATs it to a private address space, wraps it in several layers of encryption, and sends it to the first relay in UDP. Based on the packet's flow tag, the relay decrypts one layer of the encryption and 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
            $a.maintain(b)$;  $b.maintain(a)$;
```

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

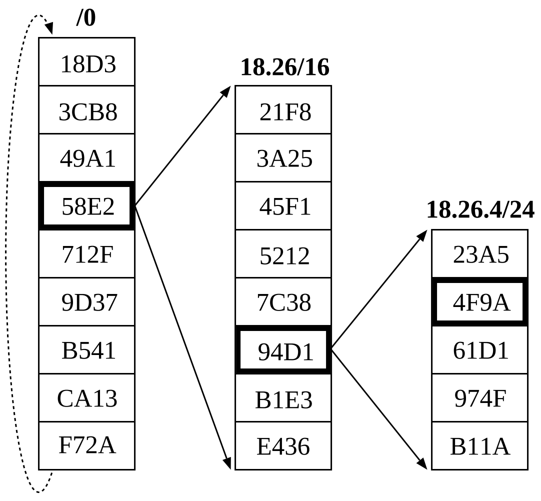

Figure 5: Peer selection on validated nodes. Shown is a *lookup*(key) with id_{16} = 541A, id_{24} = 82F1, and id_{32} = 261B. This ultimately maps to the hash value 4F9A, which yields a node with IP address 18.26.4.9.

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.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.