A Peer-to-Peer Anonymizing Network Layer

by

Michael J. Freedman

S.B., Computer Science and Engineering,
Massachusetts Institute of Technology (2001)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Author . . . .
Department of Electrical Engineering and Computer Science
June 17, 2002

Certified by . . . .
Robert T. Morris
Assistant Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Peer-to-Peer Anonymizing Network Layer

by

Michael J. Freedman

Submitted to the Department of Electrical Engineering and Computer Science
on June 17, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Existing Internet systems implement anonymity at the application layer or through centralized components. A robust, decentralized infrastructure that anonymizes any Internet traffic could benefit a wide array of existing protocols and systems. This anonymous network layer could seamlessly replace the current communications channel, and it could continue to offer anonymity and availability even while components fail maliciously.

This thesis proposes Tarzan, 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 in a way that adversaries cannot easily influence. Cover traffic prevents a global observer from drawing conclusions based on traffic analysis as to an initiator's identity.

Tarzan provides anonymity to either clients or servers, without requiring that both participate, presenting the abstraction of a one-way anonymous tunnel. In both cases, Tarzan uses a network address translator (NAT) to bridge between Tarzan hosts and oblivious Internet hosts.

We quantify Tarzan's anonymity properties and show that Tarzan imposes minimal performance overhead over a corresponding non-anonymous overlay route.

Thesis Supervisor: Robert T. Morris
Title: Assistant Professor of Computer Science and Engineering

Acknowledgments

First, I would like to thank my advisor, Robert Morris, for his guidance and advice for both my research and my professional life. Robert is always able to focus on the real problems, ask the right questions, and steer me away from purely tangential issues. Although this topic falls outside his normal interests, Robert welcomed and fully supported me nonetheless.

Second, I acknowledge Emil Sit and Josh Cates for significant contributions to the software design and implementation of the basic Tarzan tunnelling architecture.

I first became interested in Internet anonymity through work with Roger Dingledine and David Molnar. I thank them for fun paper collaborations and continued interesting discussions.

My research has benefited from many other people. I thank David Karger and Charles Leiserson for theoretical discussions on peer discovery, and David Andersen and Chuck Blake for help on the systems-side. Furthermore, David Mazières, Frans Kaashoek, Richard Clayton, George Danezis, and the rest of PDOS have provided useful thoughts and comments.

MIT and LCS have been amazing places for learning and discovery for these past five years. I doubt if they can be replicated elsewhere. Wonderful summer experiences at ZKS Labs and InterTrust STAR Lab also motivated my desire to pursue academics.

My MITOC friends have kept me sane with numerous adventures, over cliff and yonder mountain, through summer's breeze and winter's roar. Nick Feamster and Ajay Kulkarni have provided excellent late-night discussions, both in computer science and otherwise.

Lastly, I wish to dedicate this thesis to my family. My parents have been a constant source of support and love: My father has taught me the value of dedication, perserverance, and constancy; my mother has nurtured and reinforced in me a concern for others. My brother continues to look out for his younger sibling in his own way and has always been a receptive audience, even though his talents lie in physics. My four grandparents—one now of blessed memory—have been just as instrumental in raising me. I hope I have made them proud.

Contents

Chapter 1

Introduction

The right to be let alone is indeed the beginning of all freedom.

— William O. Douglas, U.S. Supreme Count Justice, 1939–1975

Every man should know that his conversations, his correspondence, and his personal life are private.

— Lyndon B. Johnson, President of the United States, 1963–69

Over the past decade, the Internet has seen explosive growth in both the number of users and the sheer quantity of available materials. This expansion has changed its role from the realm of academic research to that of mainstream use. As users perform more conventional activities on the Internet everyday, the amount and value of information related to these activities becomes ever greater.

The New Yorker magazine explained in a famous cartoon, “On the Internet, nobody knows you’re a dog” [18]. Unfortunately, the opposite has become increasingly true. While people do not communicate face-to-face in this medium, identities can be assigned to Internet users nonetheless. Some party can monitor and log virtually every email a user sends, every post to a newsgroup, every purchase made online, and every World Wide Web page accessed. Furthermore, the growth of online databases, the trend towards cheaper and larger hard disks, and the increase in bandwidth capacity all provide for the easy storage of and access to personal information.

In such ways, the Internet enables an improved means of forming personal dossiers on individuals by aggregating information from possibly diverse sources. A multitude of entities—governments, corporations, organizations, and other individuals—may create and use these dossiers.

The lack of privacy in online activities is fairly obvious. Email headers include the routing paths of email messages, including DNS names and IP addresses. Web browsers display user IP addresses; web servers log this information by default. Commonly-used online chat applications such as ICQ and Instant Messenger divulge IP addresses as well.

But the Internet also offers the potential for greater personal privacy. Enabling technologies can bring privacy to areas and activities where such protection was previously impossible. Communications anonymity provides a means for reaching this goal. This type of anonymity "blinds" any information that may be divulged on a communications channel between any two or more parties.

1.1 The problem

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 an Internet-wide pool of nodes, numbered in the millions, that relay each others' traffic to gain anonymity. This thesis describes a design aimed at realizing that vision. First, however, we discuss why less ambitious approaches are not adequate.

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

To overcome this single point of failure, a host can connect to a server through a set of mix relays [3]. The anonymous remailer system [10], Onion Routing [25], and Zero-Knowledge's Freedom [12] offer such a model, providing anonymity through a small, fixed core set of relays. However, a corrupt relay can perform network-edge traffic analysis on such a system: if the relay receives traffic from a non-core node, that node must be the ultimate origin of the traffic. A corrupt entry and exit relay can use timing analysis to determine both source and destination. An external adversary capable only of observing traffic that enters and exits the set of core relays can make the same analysis. This attack reduces the anonymizing power of long mix paths. Therefore, the connecting host still remains vulnerable to individual relay failures, and these relays provide obvious targets for attacking or blocking.

1.2 Our solution

Tarzan, the design presented in this thesis, involves sequences of mix relays chosen from a large pool of volunteer participants. All participants are equal peers; they are all potential originators of traffic, as well as potential relays. This design overcomes the edge-analysis weakness: a relay cannot tell if it is the first hop in a mix path.

An open-ended set of participating nodes composes Tarzan, lacking in any centralized component. As in other peer-to-peer systems, this lowers the barriers to participation. Tarzan allows client applications on participating hosts to talk to non-participating servers on the Internet. Tarzan is transparent to both client applications and servers, though it must be installed and configured on each participating node. In the long term, the ability for individual anonymizing relays to participate in multiple kinds of traffic may make it easier to achieve a critical mass of anonymizing relays.

Tarzan routes packets through tunnels involving a randomly chosen sequence of Tarzan peers using mix-style layered encryption [3]. The two ends of a tunnel are a Tarzan node running a client application and a Tarzan node running a network address translator; the latter forwards the client's traffic to the ultimate destination, an ordinary Internet server. These mechanisms provide anonymity in the face of malicious Tarzan participants, inquisitive Internet servers, and observers who can see traffic on a very limited number of network links.

Tarzan also uses cover traffic to ensure anonymity, even if an adversary can passively observe most or all network traffic. Tarzan generates the cover traffic using a novel *mimic* technique, practical in the sense that it consumes only a small factor more bandwidth than the data traffic to be hidden, and powerful in that it shields all network participants, not only some core routers.

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.

1.3 Thesis overview

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

Chapter 2

Design goals and network model

We use the following terminology. A *node* is an Internet host running an instantiation of the Tarzan software on a single IP address. A node, named by a unique IP address and public key, constitutes a unique *identity* in the system. A *tunnel* is an ordered sequence of nodes that form a virtual circuit for communication. 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, the goals are the following:

1. **Application independence:** Tarzan should provide the abstraction of an IP tunnel and perform transparently to user applications.

2. **Anonymity against malicious nodes:** Tarzan should ensure that a participant has *sender and recipient anonymity* against colluding nodes. That is, a particular host should not be uniquely linkable as the sender (recipient) of any message, and that a message should not be linkable to any sender (recipient). 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.

   This property 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 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. We note that centralized anonymizing systems do not offer these properties.

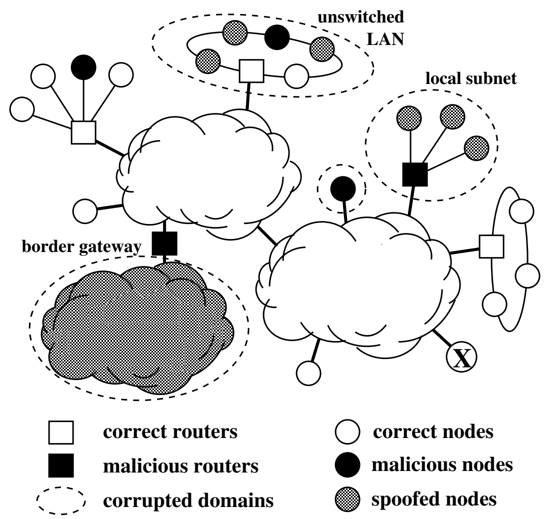

Figure 2-1: Tarzan network model. In relation to node X, adversarial machines can control address spaces and can spoof virtual nodes within corrupted domains.

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 originated traffic should be statistically indistinguishable from the traffic it receives-and-forwards.

Because anybody can join Tarzan, the system will likely include misbehaving entities. A *correct entity* will run one correct node, which forwards packets properly, does not log addressing or timing information, and so on. A *faulty entity*, also referred to as an active adversary, will run potentially many malicious nodes. 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.

To a first approximation, the fraction of Tarzan nodes that are malicious determines the probability that the malicious nodes can break anonymity. Unfortunately, a single malicious computer might be able to pretend to be multiple nodes by controlling multiple IP addresses and thus multiple Tarzan identities.

We assume that one malicious computer controls only a *contiguous* range of IP addresses, typically by promiscuously receiving packets addressed to any IP address on a particular LAN. This assumption will turn out to be useful in bounding the damage each malicious node can cause. We will call the contiguous range of IP addresses controllable by a single malicious computer a *domain*.[1]

A node belongs to a $/d$ domain if the node's d-bit IP prefix matches that of the domain. Figure 2-1 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: an adversary can subvert nodes within the same domain in a dependent fashion. We assume that nodes in different domains fail independently (modulo software exploits and the like). Therefore, when choosing relays, Tarzan should consider the notion of distinct domains, not that of distinct nodes. This differentiation is unique to Tarzan among anonymous systems.

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.

[1] Our domain notion is completely unrelated to DNS.

Chapter 3

Architecture and design

This section describes the design of Tarzan: the basic tunnel mechanism, the peer discovery protocol, and the cover traffic technique.

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

Typical use proceeds in three stages. First, a node running an application that desires anonymity selects a set of nodes to form a path through the overlay network. Next, this source-routing node establishes a tunnel using these nodes, which includes the distribution of ephemeral encryption 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 responses from these oblivious servers and reroutes the packets over this tunnel.

Tarzan operates at the IP (Internet Protocol) level and offers a best-effort delivery model. End hosts must provide functionality like 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 direction of traversal of the packet. The tunnel initiator sanitizes IP headers.

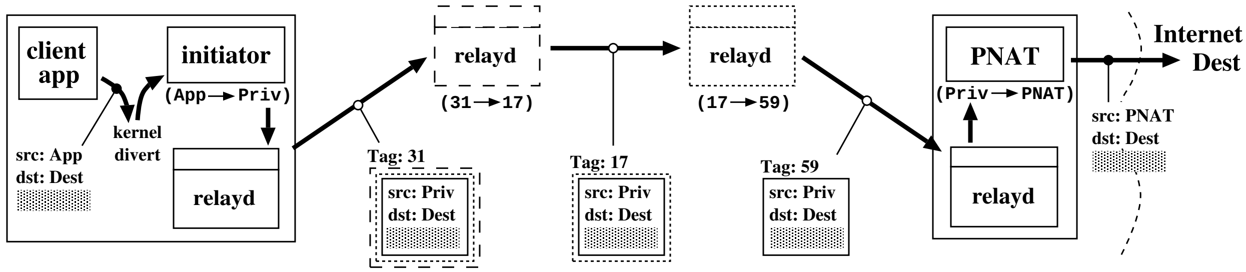

Figure 3-1: Tarzan Architecture Overview: An IP packet is diverted to the local tunnel initiator, which NATs it to a private address space, wraps it in several layers of encryption, and sends it to the first relay in UDP. Based on the packet's flow tag, the relay decrypts one layer of the encryption 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.

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 [22]) uniquely identifies each link of each tunnel. A relay rapidly determines how to route a packet based on its tag. Symmetric encryption protects data on a per-relay basis, with separate keys being 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 encryption per tunnel relay, and encapsulates the result in a UDP packet. More precisely, if the tunnel consists of a sequence of nodes $T = (h_1, h_2, \ldots, h_l, h_{pnat})$[1] and the forward key for each node is k_{h_i}, the originating node produces the encrypted block $\{\{\cdots\{\{p\}_{k_{h_{pnat}}}\}_{k_{h_l}}\cdots\}_{k_{h_2}}\}_{k_{h_1}}$ from the input packet p. The origin tags this block with the first relay's flow identifier and forwards it to h_1. That node's relay will decrypt the block (*i.e.*, strip off one layer of encryption,) retag the packet, and forward it on to the next relay. 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. Note that this design places the bulk of the encryption workload on the node seeking

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

$h_0 = $ *initiator*;
$h_1 \in_R \{nodes\}$;
for $i = 1$ **to** l
    $h_{i+1} \in_R \{nodes\}$
    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*;
            $h_{i+1} \in_R \{nodes\}$;
            send *reset_forward_request*(h_{i+1}) **to** h_i;
            $rc =$**wait for** *reset_forward_response*;
send *establish_response*(h_l) **to** h_{pnat};

Figure 3-2: Pseudocode for tunnel establishment protocol

anonymity. Nodes that are merely relaying will perform only a single symmetric key operation per packet that is processed.[2]

3.2 Tunnel setup

When forming a tunnel, a Tarzan node randomly selects a series of nodes from existing peers in the network. Each relay publishes a public key that it generates locally the first time the it enters the network. We rely on this relay being the only one that knows the corresponding private key. Section 3.5 describes how Tarzan discovers nodes and their corresponding public keys.

Tunnels are established iteratively one relay at a time. The tunnel source-routing entry-point is responsible for setting up the entire tunnel, which consists mainly of generating and distributing the symmetric keys.

The high-level establishment algorithm is shown in Figure 3-2. An establish request sent to node h_i is relayed as a normal data packet from h_1 through h_{i-1}. Node h_i cannot distinguish whether the packet originated from node h_{i-1} or from one of that node's predecessors; node h_{i-1} cannot distinguish successive establish requests from ordinary tunneled data. The establish request contains the forward decryption key that h_{i-1} will use when sending packets to h_i and the encryption key that should be used for sending packets received from h_{i+1}. Additionally, it establishes the flow identifiers that will be used to tag packets going in each direction. The initiating node uses the

[2] Section 3.7 also describes an additional pair-wise encryption-decryption used between nodes for stronger anonymity properties.

public key of node h_i to encrypt the initial forward session key and then this session key to encrypt the subsequent reverse key, node addresses, and flow identifiers. When h_i has successfully stored the state for this request, it responds to the origin for an end-to-end check of correctness.

For path length l, this algorithm takes $O(l)$ public-key operations and $O(l^2)$ inter-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 and ships them over a Tarzan tunnel. The client 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 origin port numbers for TCP and UDP packets. Chapter 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, any two users can communicate anonymously 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 sends ping messages to the PNAT through the tunnel and waits for acknowledgments. Upon multiple unsuccessful retries, the initiator attempts to reconstruct a tunnel. If the PNAT is the point of failure, *i.e.*, h_l still responds to pings, the initiator simply selects a new h_{pnat} for the tunnel. Otherwise, it attempts to rebuild the tunnel to the original PNAT, so that higher-level connection-based protocols, such as TCP, do not die upon tunnel failure.

To reconstruct a broken tunnel, the initiator selects a random i uniformly from $[1, l]$. It attempts to rebuild the tunnel from h_i forward, $T' = (h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_l, h_{pnat})$. If h_{i-1} cannot be contacted to update next relay pointers and distribute new keys, the initiator decrements i by one and repeats the reconstruction attempt.

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 4.2.1 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 than recent peer-to-peer lookup protocols [24], which spend much 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. 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.

We model this problem as a directed graph: vertices represent Tarzan nodes; edges correspond to the relation that node a knows about node b. Edges are added to the graph as nodes discover other peers. Node a can communicate with b only if they are neighbors, *i.e.*, $(a, b) \in E$. A node is reachable only if its in-degree ≥ 1. 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*.

Note that our use of "gossiping" is a slight misnomer. Traditional gossiping protocols generally assume a fully-connected or fixed communication network and seek to optimize the broadcast of extra information, such as link state.

Our technique to grow this network graph is similar to the Name-Dropper resource discovery protocol [15]. 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, the in-degree of each node increases. Nodes will transmit neighbor sets whose elements are largely known by 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-nary searches on prefix-aggregated hashes of the elements. A full description of this mechanism is outside the scope of this thesis; we briefly discuss it in Section 4.2.1.

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 4.2.1).

Figure 3-3 shows the main gossip protocol. To join the system, a new node a contacts some existing node b to learn 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, nodes quickly learn about all other nodes in the network (see Section 4.2.1).

3.6 Peer selection

Having built a peer database, we want to select nodes from this database for building cover traffic links in the next section.

One may be tempted to simply choose nodes completely at random from V_a. However, while an adversary can potentially run as many Tarzan relays as IP addresses to which he has access, these addresses are rarely scattered uniformly through the IP address space. Instead, they are often located

```
// Node a joins the network
// Node b is an arbitrary node in the network known to a
a.join(b)
    U_a = {b}; V_a = ∅;
    a.gossip();

// Main Tarzan discovery protocol
// 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 = ∅), U_a = V_a;
        b ∈_R U_a;
        if (|V_a| < (1/c)|V_b|)
            b.busy ? a.redirect(b) : a.initialize(b);
        else if (|V_b| < (1/c)|V_a|)
            a.busy ? b.redirect(a) : b.initialize(a);
        else
            a.maintain(b);   b.maintain(a);

a.initialize(b)
a.redirect(b)
a.maintain(b)
    [rc, P] = get peers via RPC to b;
    if rc ∈ {!ok, timeout}
        V_a.remove(V_a[b]);
        U_a.remove(U_a[b]);
    else
        a.gossip_success(b, P);

// validate b and record newly-learned peers
a.gossip_success(b, P)
    // validate b as returning successfully
    V_a.insert(U_a[b]);
    U_a.remove(U_a[b]);
    for p ∈ P
        if p ∉ V_a ∪ U_a
            // p is a newly-discovered peer
            U_a.insert(p);
```

Figure 3-3: Pseudocode for the peer discovery protocol

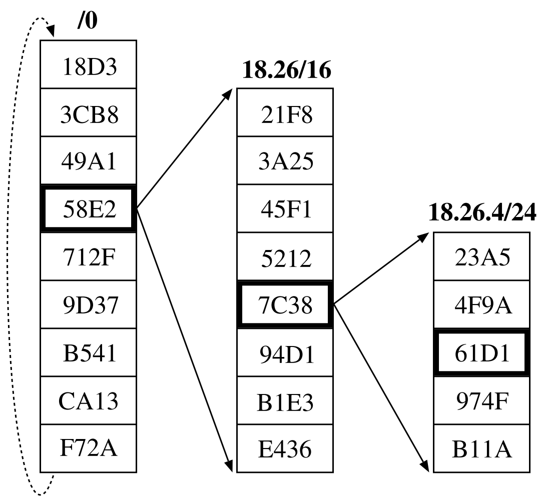

Figure 3-4: Peer selection on validated nodes. Shown is a lookup on key 541A, resulting in the node tuple with hash value 61D1, which belongs to the domain 18.26.4/24.

in the same IP prefix space. Thus, we choose among *distinct* IP prefixes, not among all known IP addresses.

Therefore, we select nodes by choosing randomly among the populated *domains* at each level of the table in Figure 3-4. 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* at each level. An IP address or domain prefix is transformed to an *identifier* via *hash*$(addr, date)$, where *hash* is a cryptographic hash function and *date* is the day-of-the-month according to GMT. These identifiers are ordered modulo $2^{|id|}$. Under the random oracle assumption, they are randomly distributed on this ring.

Therefore, Tarzan's *lookup*(k) method selects peers as follows: Node a first generates a key k and iteratively calls *successor*(k) on identifier rings on increasing specificity. *successor*(k) selects the smaller identifier $\geq$ key (with wrap-around) on a ring. We explain this complexity in Section 4.2.1.

Note that a node executes *lookup* completely locally, based on information already accumulated in its peer database. This local operation differs from peer-to-peer distributed hash tables [24]. Therefore, two nodes may have slightly different lookup structure replicas, which can yield temporarily inconsistent results. We return to the impact of these possible inconsistencies in the next section.

Tarzan uses the date when computing address identifiers to provide a daily reordering of ring elements. This randomization stops any particular domain or address from owning a larger space in the ring for any duration, which might otherwise tempt an adversary. Furthermore, this rebalancing reorders the validated set daily, randomizing how nodes propagate their neighbors during *maintain* operations.

3.7 Cover traffic mechanisms

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. The value k is a global parameter. 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.

Each node has κ mimics, where κ has an expected value of $2k$: Node a selects k nodes at random as its mimics, and an expected k nodes select a as they look for their own mimics.

This mimic relationship must be symmetric for two 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. Second, a otherwise would not be incentivized to provide cover traffic on its incoming links.

This mimic relationship must be universally verifiable in order 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 M_i^a, as the peer returned by *lookup*(K_i), where $K_i = hash_i(a.IPaddr, date)$. $hash_i$ is a cryptographic hash function recursively applied i times, $i \leq (k+1)$. Node a sends a mimic request, including the tuple $\{K_i, a.IPaddr, i\}$, to 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:

- $1 < i \leq (k+1)$
- $K_i = hash_i(a.IPaddr, b.date)$
- $b.lookup(K_i) = b$

If the *lookup* check fails, we must consider two cases. First, node b can have a different view of the network than a; its execution of *lookup*(K_i) can map to a different domain or peer. This inconsistency occurs when b knows an identifier in the ring between K_i and b of which a is unaware. Node b declines the mimic request but returns the identifier's corresponding node, to which a sends a new mimic request.

Second, a may initially contact node c and receive no response, signifying failure. Node a removes c from V_a; its execution of *lookup*(K_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 the failure, 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*(K_i).

3.7.2 Tunnelling 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 3-5 shows this manner of tunnel establishment over mimics.

To initiate a tunnel, node a chooses a mimic M_i^a as its first tunnel relay. This node M_i^a has its own mimics $\mathcal{M} = \{M_1^{M_i^a} \ldots M_\kappa^{M_i^a}\}$, 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 M_i^a's key should result in the same set. Node a randomly selects $M_j^{M_i^I}$ from the resulting set and

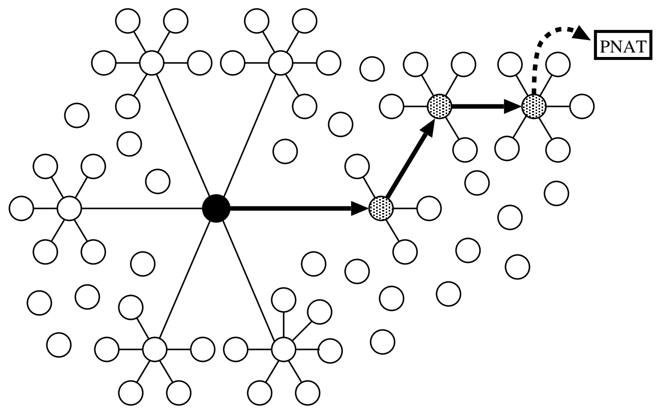

Figure 3-5: Mimic topology and traffic flows for $k = 3$: each node has $\kappa \approx 6$ mimics (shown connected by solid lines); a node establishes its tunnel over mimic links (arrows in bold) and completes it with a random PNAT (dotted line). Nodes inject cover traffic with data traffic to yield 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.

continues by tunneling an establish request to $M_j^{M_i^I}$ via M_i^I. 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.[3] 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 and its mimic b:

[3]Our implementation pads to two different sizes as a performance optimization and thus manages two separate queues. See Section 5.

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

For a node's behavior to appear innocuous, the following relations must hold. Let $\mathcal{M}$ be the set of a node's mimics, and let $\mathcal{T}_I^{\mathcal{M}} = \{\mathcal{T}_I(b) : \forall b \in \mathcal{M}\}$ be the set of total incoming traffic rates from $\mathcal{M}$.

$$f(\mathcal{T}_I^{\mathcal{M}}) \leq \mathcal{T}_O(b) \leq \max(\mathcal{T}_I^{\mathcal{M}}), \forall b \in \mathcal{M} \tag{3.1}$$

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

Ideally, each node provides "enough" traffic to cover its mimics' actions. Per the lower-bound in Equation 3.1, a correct 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. The upper-bound specifies a maximal rate to prevent a node from being a clear source of traffic: its outgoing volume should always be accounted for by a corresponding incoming volume.

This total outgoing traffic level is independent of actual *data* traffic, which itself is constrained:

$$\mathcal{D}_O(b) \quad \leq \quad f(\mathcal{T}_I^{\mathcal{M}}), \forall b \in \mathcal{M} \tag{3.2}$$

Intuitively, Equation 3.2 limits the outgoing data rate on any link to a function of the incoming traffic rate. This function allows the node to still send data at reasonable speeds if up to $\frac{1}{3}$ of its mimics are slow, either maliciously faulty or not.

Each time period, a relay batches some number of data packets from the previous period (per Equation 3.2), creates cover packets to round up to its packet-rate requirement (per Equation 3.1), randomly permutes this set, and writes them to the link-layer. The relay starts dropping data packets once its data queue begins to fill. Any congestion control and retransmission are pushed back to the higher-lever protocol between communicating end-hosts.

Chapter 4

Security analysis

This section analyzes Tarzan's anonymity properties. Unfortunately, our formality is limited by the lack of any standardized anonymity proof model. In order to make quantitative statements supporting our negative goal—that a tunnel initiator's identity is not exposed—we take the following approach.

First, we present a restricted attack model for a theoretical adversary. Then, we analyze Tarzan under this model and calculate both the probability that an initiator escapes identification and its expected *anonymity set* size.

Next, we describe a set of attacks on Tarzan's peer-to-peer design and argue that Tarzan prevents these attacks. If these claims are true, our strict attack model actually subsumes many practical attacks.

Lastly, we describe attacks for which we offer no guaranteed protections, perhaps because they should be handled at a higher level.

4.1 A simple attack model

Our attack model considers a *static* adversary (or set of colluding static adversaries). At some time t_i, he corrupts some number of independent physical machines and thus present m malicious nodes (identities) to the system. Let M be the number of malicious colluding domains in the system, equivalent to the number of corrupted physical machines, yet $M \ll m$ in likelihood.

For the rest of this section, we assume a network that is guaranteed to transmit messages without failure or modification. This network includes both correct and malicious nodes. n is the number of nodes in the network. N is the number of domains containing these nodes.

We consider domains in our analysis, not nodes, so as to argue about distinct system entities: Nodes within the same domain can be corrupted in a correlated fashion. Tarzan's node-selection mechanism of choosing randomly from known domains supports this counting method. Note that this model for counting *colluding* adversaries differs from the Byzantine fault tolerance concern with the *total* number of "faulty" nodes.

Node a runs Tarzan's peer discovery protocol to learn all n nodes in the network. a establishes connections with κ mimics at time t_j, for $t_i < t_j$. These mimics are malicious with independent probability $\frac{M}{N}$. Node a then initiates and constructs a tunnel of length l. Relays on this tunnel are also malicious with probability $\frac{M}{N}$.

At some later time t_k, the malicious nodes activate and begin analyzing their traffic flows. These colluding nodes can monitor and analyze the message contents, sizes, rates, and volumes of their own traffic. They can use timing analysis to determine that packets are on the same tunnel (but not to estimate the tunnel distance between relays).[1] For now, these nodes can perform no other attacks.

To a first-order approximation, this simple attack model subsumes many of the threats that Tarzan should expect to weather. In some sense, the model is what remains after we consider Tarzan's protections against certain threats and assume away other more powerful attacks.

Against malicious participants, Tarzan prevents attacks on its peer-to-peer design that would bias its its choice of tunnel relays. However, Tarzan cannot defend against an adversary that can adaptively compromise arbitrary nodes (can a defense ever be possible?); we thus ignore such a threat in our analysis. Against a global eavesdropper—which perhaps itself is an overly strong threat model—Tarzan protects information leakage from the tunnel through the use of cover traffic and argues that information leakage from the application-layer should be handled by that layer. We discuss these issues later in this section.

Our analysis examines two issues. First, we derive the probability that a malicious node can identify a as the tunnel initiator. That is, given that a chooses a tunnel length randomly, established through several potentially malicious nodes, what is the probability that a malicious relay's predecessor is actually a? Second, we calculate a's expected anonymity set within the network: While the malicious nodes cannot deterministically identify a, how many nodes do they suspect may be the initiator a?

[1]As transmission time is dominated by wire propagation delays (see Section 6), we expect it difficult to estimate tunnel length beyond a margin of error $\pm$ several hops, likely approximate to l itself.

4.2 Measuring anonymity

Tarzan provides sender and recipient anonymity through its *relay homogeneity*: All nodes both originate and forward traffic. If an eavesdropper observes some node forwarding traffic, the eavesdropper can distinguish neither whether the packets are cover or data traffic, nor whether the nodes initiates the traffic or merely relays it.

We now analyze the probability that a set of malicious nodes on a tunnel can correctly conclude that the first non-malicious node upstream of them is the originator. Let h_i be the first such malicious relay on a's tunnel. It can distinguish between cover and data traffic from h_{i-1}, but cannot determine whether h_{i-1} is initiating the traffic (and is thus a) or merely relaying the data. Crowds [21] provides a similar argument for sender anonymity, but without the security benefits of layered-encryption, source-routing, or cover traffic.degree.

Let I be the event that h_i's observed predecessor is actually the initiator a. Let S_r denote the event that $(r-1)$ malicious participants succeed h_i on the same tunnel. If these nodes cannot link traffic on the tunnel through timing analysis, they require $\rho = r$ contiguous relays on the tunnel. However, if colluding nodes can always tell when they are relays on the same tunnel, they only need to directly control $\rho = \Omega(\lceil \frac{r+1}{3} \rceil)$ relays, as every node knows at most its two correct neighbors.

Given our model that a malicious relay is chosen independently with probability $\frac{M}{N}$,

$$Pr(S_r) = \prod_{i=0}^{\rho-1} \Big(\frac{M-i}{N-i}\Big) < \Big(\frac{M}{N}\Big)^{\rho}$$

For simplicity, hereafter we say $Pr(S_r) = \big(\frac{M}{N}\big)^{\rho}$.

If the malicious relay h_i is the immediate successor of a, then I is true and the tunnel length l actually equals r. As Tarzan allows variable path lengths, l normally takes some probability distribution L. In general, the probability that $l \geq r$ is simply $\sum_{i=r}^{\infty} Pr(L(i))$, or one minus the cumulative density function at $L(r)$. Thus,

$$Pr(l = r | l \geq r) = \frac{Pr(l{=}r)Pr(r{\geq}l)}{Pr(l{\geq}r)} = \left(\frac{Pr(L(r))}{1 - cdf(L(r))}\right)$$

The probability an adversary at h_i sees the tunnel initiator:

$$Pr(I \cup S_r) \quad = \quad 1 - \prod_{r=1}^{\infty} \Big(1 - Pr(S_r) Pr(l = r | l \geq r)\Big)$$

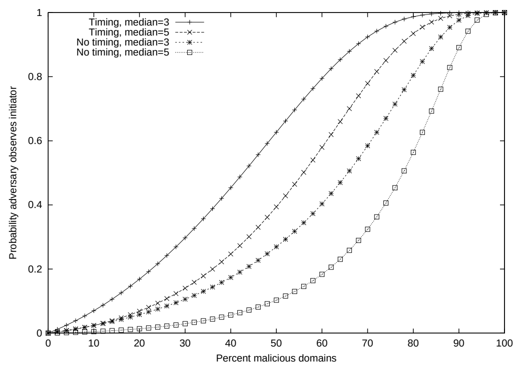

Figure 4-1: Probability of exposure given tunnel length taking a Lognormal distribution with median lengths 3 and 5, σ of 0.5. We plot the case in which adversaries can perform timing analysis ($\rho = \lceil \frac{r+1}{3} \rceil$) and the case in which they cannot perform such analysis ($\rho = r$).

$$= \; 1-\prod_{r=1}^{\infty}\left(1-\left(\frac{M}{N}\right)^{\rho}\left(\frac{Pr(L(r))}{1-cdf(L(r))}\right)\right)$$

Figure 4-1 shows this probability of exposure plotted against increasing adversarial compromise of the network. We model $L(\cdot)$ as a Lognormal (heavy-tailed) distribution and plot probability of exposure when malicious nodes can and cannot use timing analysis to determine that they belong to the same tunnel.

Tarzan's mimic mechanism ensures that every node has incoming traffic from an expected κ different mimic nodes. For our analysis, we assume that κ is fixed and n is large enough such that there is no appreciable overlap in mimics. Thus, while the malicious relay h_i identifies h_{i-1} uniquely, it has $(\kappa-1)$ possible suspects for h_{i-2} (one of h_{i-1}'s mimics is h_i). An adversary must recursively back-trace these mimic links as far as the suspected remaining path length, increasing the set of possible initiators—the anonymity set—exponentially.

If none of these recursively counted mimics are malicious, the initiator's anonymity set is simply $(\kappa-1)^{l-r}$ for some fixed tunnel length l and a compromised sequence of relays of length r.

As a node's mimics may be malicious, the fan-out factor of possible initiators reduces to $\kappa - (\prod_{j=0}^{\kappa-1}\frac{M-j}{N-j}) \approx \kappa - (\frac{M}{N})\kappa$. Therefore, we calculate the expected anonymity set size given malicious nodes:

$$E(|AS|) \;=\; \sum_{r=0}^{\infty} Pr(S_r) \sum_{i=r+1}^{\infty} Pr(l=i \,|\, l \geq r)\Big[|AS|\Big]$$

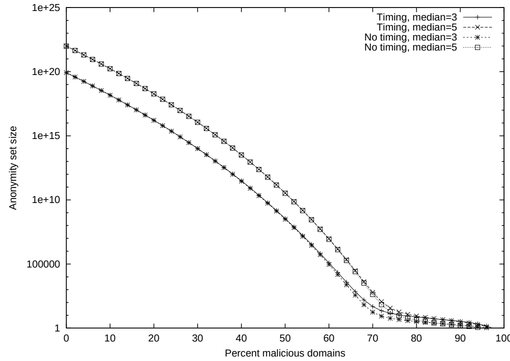

Figure 4-2: Anonymity set size with $\kappa = 6$. The number of nodes in the network is unbounded: there is no overlap between possible initiators. We plot the cases in which adversaries can and cannot perform traffic analysis.

$$|AS| = \begin{cases} \left(\frac{N-M}{N}\right)^{l} \kappa\,(\kappa-1)^{(l-1)} & \text{if } r = 0 \\ \left[\left(\frac{N-M}{N}\right)(\kappa-1)\right]^{(l-r)} & \text{otherwise} \end{cases}$$

Figure 4-2 shows an initiator's expected anonymity set size for an unbounded network.

4.2.1 Protecting relay selection

Our attack model makes several claims that allow us to quantitatively analyze the anonymity of Tarzan tunnel initiators. In this section, we argue that these claims are reasonable given Tarzan's design.

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

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

Together, these claims imply that an initiator selects a malicious node as a tunnel relay with probability $\frac{M}{N}$. This probability is used when calculating a's chance of exposure and its expected anonymity set size.

Consider what is required to support these strong claims: Node a must select its mimics at random and in an unbiased manner from the entire set n; it also must build tunnels only through other nodes' mimics that are similarly randomly-selected.

However, malicious nodes may attempt to do one or more of the following after joining the network, in order to bias a's mimic selection and increase the frequency of use of malicious relays in its tunnel.

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

- **Leverage open admission policy:** An adversary tries to squat on important IP addresses or run multiple Tarzan nodes on different IP addresses or with different public keys.

- **Ignore neighbor-selection mechanism:** A malicious node attempts to select malicious nodes as its mimics, to ensure correct nodes will build tunnels through sequences of malicious nodes.

In this section, we describe how Tarzan protects against these types of attacks on Tarzan's open peer-to-peer design.

4.2.2 Securing resource discovery

In an attempt 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. Correct 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 ($h_{i+1} \dots h_j$), even though it might not control their relevant domains. Therefore, this attack causes an initiator to select a malicious relay with probability $> \frac{M}{N}$.

A node must learn all n nodes in the network to protect against a bias towards malicious nodes in its peer database. This bias arises as correct nodes return both correct 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 n. If a colluding malicious node receives a tunnel establishment request, only α possible nodes could have initiated the message. Therefore, 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 bound is actually tighter:

$$(\kappa-1)^{l-r}\frac{\binom{\alpha}{r}}{\binom{n}{r}} = (\kappa-1)^{l-r}\frac{\alpha(\alpha-1)(\alpha-2)...(\alpha-r+1)}{n(n-1)(n-2)...(n-r+1)} \approx \frac{(\kappa-1)^{l-r}\alpha^r}{n^r}$$

To uniquely identify the initiator, an eavesdroppers needs this value to be ≤ 1, or $\alpha \leq n/(\kappa - 1)^{\frac{l-r}{r}}$. In general, for any $\alpha < n$, this discovery profiling attack learns some information. Therefore, our discovery protocol need ensure that a node learns close to all n neighbors.

Lastly, a node attempts to check the validity of other nodes' mimics during tunnel establishment. For this check to succeed often, the node must also know nearly all nodes in the network.

Theorem 1. *A node will discover all n nodes in the network with high probability in connection complexity $O(n)$ and pointer complexity $O(n \log_k n)$.*

Proof. A full proof is beyond the scope of this thesis. Tarzan's discovery protocol propagates knowledge the same as Name-Dropper [15]. 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 determine one set difference as follows. The ith element of a node's sorted set V_a has the hash value $h(\ldots h(h(V_a[1]){+}V_a[2])\ldots{+}V_a[i])$, generated by recursively aggregating prefixes. If two nodes have the same last element $V_a[|V_a|]$, their sets are identical and the comparison terminates. Otherwise, the nodes compare their value for $V_a[\frac{|V_a|}{2}]$ 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.

Thus, *maintain* passes $O(\log_k n)$ pointers to learn one new neighbor. Each node receives $O(n \log_k n)$ pointers to learn all n nodes.

A node's initial entry point into the network must be correct; otherwise, all tuples the node learns thereafter could correspond to malicious nodes. $\square$

4.2.3 Hardening the open admissions policy

A entity can present multiple identities in Tarzan. To perform this attack, an adversary uses different public keys or runs multiple instances of Tarzan on different IP addresses. There are no known techniques to prevent this type of attack in a large-scale distributed system without some trusted centralized authority [8, 9].

However, 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 node distributes its public key through gossip channels beyond its direct control. Such indirection protects a tunnel initiator against this key-mapping attack. Note that this attack arises from the decentralized, untrusted nature of peer-to-peer systems: Tarzan does not provide an authenticated key-to-identity binding with public keys.

Second, consider alternative approaches to peer selection. One approach is a local variant of Chord's distributed hash table [24], which maps IP addresses onto a single identifier ring via a hash function. $\mathit{successor}(random(ID))$ selects some random element from this ring. This ID-space selection approach has obvious load-balancing and randomness benefits, but suffers from the admission control problem [9]: An adversary's multiple addresses are distributed randomly around the identifier ring, each having an equal probability of selection as individual correct nodes.

Alternatively, a node can simply map IP addresses onto the identifier ring, rather than mapping their hash values. This IP-space approach offers better protection against the admission control problem, as a node selects contiguous IP addresses with low probability. However, certain IP addresses now would be much more "important" than others, given a non-uniform distribution of Tarzan nodes in IP space. These nodes subsume more of the IP ring according to *successor*: for example, if Xerox's $/8$ has few Tarzan nodes. Such nodes could become overloaded and would be a clear target for attack.

Tarzan takes a hybrid approach with its identifier rings. Nodes hierarchically select among domains (IP prefixes), not nodes (IP addresses). Therefore, an adversary does not benefit much by running multiple intra-domain nodes. Furthermore, as domain prefixes map to random locations on the identifier ring, we assign equivalent importance to all individual domains.

4.2.4 Enforcing proper mimic selection

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

Tarzan ensures verifiable mimic-selection through the following theorem:

Theorem 2. *A tunnel initiator will only select among a node's correctly-chosen mimics with high probability when extending a tunnel.*

Proof. Assume to the contrary that the initiator will accept an arbitrary node w that malicious node u specifies as one of its mimics. Then, the initiator's *lookup* on node u's key K_i will result in node w. This result occurs if either the initiator does not know of node v which lies between K_i and w on the identifier ring, or node w is actually a correctly-chosen mimic. The first point contradicts Theorem 1 that the initiator knows all n nodes in the network with high probability. The second point contradicts the ability of node u to return an incorrectly-chosen node. □

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.

4.3 Considering independent node failure

The attack model in Section 4.1 conceals two underlying assumptions. First, domains fail independently. Second, passive observation yields no additional information. We consider these assumptions in the next two sections.

Tarzan introduces the concept of a *domain* to capture some notion of a network security point of failure. Nodes within a domain certainly experience correlated failures, *e.g.*, by the compromise of their gateway router or by the corruption of a machine on a unswitched LAN. Tarzan assumes that nodes in different domains fail independently of one another.

For domains to fail independently, we require a *static* adversary. An adaptive adversary can otherwise pick-and-choose specific nodes and compromise them at will. In our attack scenario, the malicious relay h_i could otherwise compromise his predecessor h_{i-1}, and so on, until it back-traces the tunnel to a. Similarly, h_i could DoS a's mimics to reduce its mimic fan-out to zero, making its anonymity set exactly $\{a\}$.

Virtually all security and cryptographic proofs make our assumption that correct nodes have trusted, uncorruptible private execution environments.

In reality, standard system attacks like buffer overflow are much easier and likely than widespread network compromise and monitoring. While Tarzan's peer-to-peer architecture manages and scales to large numbers of participants, centralized systems like Mixmaster [10], Onion Routing [25], and Freedom [12] provide only a small number of available nodes. Thus, while these systems do not need to handle some of the security concerns in Section 4.2.1, limited targeted attacks can have a much greater impact on them. This divide becomes greater as the number of Tarzan users increases.

Tarzan takes the traditional stance that distributing trust among fully-decentralized participants benefits fault-tolerance and security. We imagine several hundred nodes in early stages of deployment and possibly thousands or more in later stages.

Tarzan's domain-driven selection process constructs tunnels that emphasize *host diversity*. It spreads relays across jurisdictional and operational boundaries; these relays probably run a variety of operating systems and could potentially use different software implementations.

4.4 Passive observation and cover traffic

Our analysis assumes that passive observation leaks no identifying information. This section discusses how Tarzan protects against information leakage within the Tarzan overlay itself, and notes how some attacks may be successful by using additional application-layer information.

4.4.1 Information leakage from 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.

Prior system proposals have several shortcomings: they protect only the core of the static mix network and thus allow traffic analysis on its edges [2, 25], they simulate full synchrony and thus trivial DoS attacks halt their operation [7], or they require central control and knowledge of the entire network [14].

If an eavesdropper can observe all network traffic, relay homogeneity alone does not provide sender or relationship anonymity. An eavesdropper wishing to identify users that contact a particular Internet host can 1) identify a connection's PNAT easily by the packets' IP address, 2) observe which Tarzan nodes communicate with this PNAT and learn the set of potential previous relays, and 3) iterate this process yields 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 in randomly building tunnels within the network, its anonymity set is naively $(n-m)$. However, such systems cannot guarantee some number of 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 exchanges cover traffic. This fixed allocation makes the generation of cover more practical, at the cost of a loss of freedom for tunnel initiators. A pair of mimics exchange a fixed volume of traffic per time period, consisting of a variable amount of real data padded with enough cover to reach the fixed volume. All packets are of the same size and are link-encrypted between cooperating mimics.

How might 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 3.2 balances the threat of denial-of-service when using slow mimics with the risk of reducing anonymity fan-out when using malicious mimics: It sets the rate of any individual outgoing data flow $\mathcal{D}_O(b)$ to at most the 33^{rd} percentile of incoming traffic rates.

Alternatively, if node a rate-limits $\mathcal{D}_O(b)$ to its minimum incoming traffic rate, one misbehaving mimic could effectively stop the node from transmitting any data.

On the other hand, if a limits $\mathcal{D}_O(b)$ 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 correct mimics. (This attack extends to when malicious nodes comprise two-thirds $(1 - f(\cdot))$ of a's mimics and a originates data faster than any incoming traffic from its correct 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 by slowing incoming rates.

4.4.2 Information leakage from application-layer

Designed at the IP-layer, Tarzan offers no protection against information that leaks through application-layer interaction.

First, an adversary may time application-layer event, such as the period between queries and responses, to estimate tunnel length. This information would change our analysis in Section 4.2 as it tightens the probabilistic distribution of tunnel lengths. 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' online/offline behavior to trace them by observation over a long period. An analysis of anonymity degradation by this *intersection* attack is presented in [30]. If a tunnel's first and last relays collude, and these relays determine they are on the same tunnel (through timing analysis), the adversaries require $2(\frac{N}{M})^2 \ln n$ *rounds* to uncover the initiator's identity with high probability.

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, this attack may become increasingly difficult.

We note that Tarzan's iterative tunnel establishment and partial reconstruction protocol offers some protection against the *set-up attack* proposed in [30] and described as an open problem: An initiator chooses a new tunnel successor h_1 only with probability $1/l$ or when h_2 fails.

Chapter 5

Implementation

We have implemented Tarzan in `C++` on Unix to validate our approach. SFS libraries [17] provide callback-style functionality for fast asynchronous I/O and automate marshalling data into the standard XDR wire representation.

The core component of Tarzan 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 [23] key. Each packet includes an 8-byte random initialization vector. Tarzan uses the Rabin public-key cryptosystem [29] to encrypt the forward-path session key for tunnel establishment. The Rabin implementation is secure against adaptive chosen-ciphertext attacks and is plaintext-aware.

A pseudo-random generator based on DSS [19] generates symmetric-keys, IVs, and packet flow tags. See [17] for details of the seeding mechanism. Tarzan uses SHA-1 [20] for all cryptographic hash operations.

Nodes send a burst of randomly-permuted packets to their mimics once every 20 msec. The encrypted payload of these packets are one of two sizes—184 bytes or 1456 bytes—in equal proportion each time period. The smaller number is sufficient for the 40-byte encapsulated packet header, 20 bytes of payload (as is common in interactive protocols), and layered encryption for up to five hops. Packets requiring fewer layers of encryption fit more payload. The 1456-byte choice reflects Ethernet MTU and cipher-block size. Prior to link encryption, a Tarzan node pads packets up to the relevant fixed size.

Chapter 6

Performance

This section examines the overhead added by Tarzan's packet handling and crypto processing, as well as 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 `relayd` running on a 1.2 GHz Athlon box with 128 MB of RAM running FreeBSD 4.3. Tables 6.1 and 6.2 summarize our results.

Table 6.1 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. 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.

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

Efficient concerns for per-packet public-key operations motivate Tarzan's design criteria for separating tunnel setup from packet forwarding. Table 6.2 shows the overhead incurred by setting up multi-hop tunnels using public-key operations.

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,

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 6.1: Per-hop latency and forwarding rate. Packet sizes shown are the actual number of bytes transmitted: packets have not been 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 6.2: Setup latency (msec) and its variance for the construction of multi-hop tunnels.

we incur a setup cost of 20 msec per hop. This data suggests that the cost of latency incurred on the underlying network will dominate latency even during tunnel setup.

In summary, a Tarzan relay enjoys a fast packet forwarding rate, high throughput, and reasonable tunnel-setup latency. 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 perceived efficiency compared to its non-anonymous counterpart.

Chapter 7

Integration

A primary goal of Tarzan's design 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. In order for this to work, Tarzan should be as transparent as possible, and integrate with applications and higher-level protocols as painlessly as possible. This section presents examples of transparent and painless integration of Tarzan with existing protocols and applications.

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 source IP address in the IP header. Protocols that put identifying information in packet payloads need additional work to anonymize. The following are examples of how Tarzan could help anonymize existing protocols:

- **HTTP:** 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.

- **DNS:** Since DNS requests do not identify the sender, Tarzan can transparently anonymize them.

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

- **FTP and IRC:** Some protocols explicitly encode IP address and port information in packet payloads. Tarzan uses existing NAT code which takes care of this in the case of common protocols such as FTP and IRC.

- **telnet, ssh, and SSL:** These interactive and/or authentication protocols do not leak any additional information. Any login authentication mechanism could be used to identify the user pseudonymously. Tarzan can handle these protocols transparently.

- **Instant messaging:** IM systems such as ICQ, AIM, and Jabber implement their own namespaces at the application-layer. Tarzan could be used for pseudonymous instant messaging.

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. We note that Free Haven [8] and Tangler [27] specifically cite the need for a system like Tarzan for their anonymity requirements.

Chapter 8

Related Work

A wide body of existing work focuses on how to achieve anonymity on the Internet. This prior work falls into two categories: systems which provide application-specific anonymity, and systems that offer a more generic transport framework.

8.1 Application-specific systems

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

8.1.1 Anonymous email

The theoretical pursuit of anonymity on the Internet began with David Chaum's 1981 proposal for a system called a "mix network" [3]. A mix network is a collection of routers, called mixes, that use a layered encryption technique to encode data and routing information as messages pass through the network. Each mix in a chain between sender and receiver strips off the identifying marks on incoming messages, randomly permutes the messages, and then sends the messages to the next mix using this new ordering.

Until the rise of proxy-based and TCP/IP-based systems, the most popular form of anonymous communication was the *anonymous remailer*: a form of mix which works for e-mail sent over SMTP. The evolutions of remailers has led to the Mixmaster Type II remailer [10]. When receiving a message, all of which are padded to the same length, a mix decrypts the messages header, which includes the next-hop header, and places the message in a "message pool." Once enough messages have been placed in the pool, the node picks a random message to forward.

Remailers also provide *reply blocks* for users to create and maintain pseudonyms which receive e-mail. By prepending the reply block to a message and sending the two together to the first remailer in the chain, a user can send a message to a party without knowing his real email address. Pseudonym servers, like nym.alias.net [16] automate management of such reply blocks. While Tarzan's PNAT is conceptually similar to the pseudonym mail servers, PNATs are widely distributed and serve their function for any given node only transiently.

8.1.2 Anonymous web browsing

Proxy services provide one of the most basic forms of anonymity: A sender inserts a centralized trust third party party between him and the recipient of a given message. The Anonymizer was one of the first examples of a *form-based web proxy* [1]. Users point their browsers at the Anonymizer page at `www.anonymizer.com` and enter their destination URL into a form displayed on that page. The Anonymizer acts as an HTTP proxy for these users, stripping off all identifying information from HTTP requests and forwarding them on to the destination URL.

Crowds [21] was named for collections of users that are used to achieve partial anonymity for Web browsing. A user initially joins some crowd via a central rendezvous server and begins peering messages within that crowd. To instantiate communications, the user creates some path through the crowd by a random-walk of other peers, in which each peer has some small probability of sending the actual HTTP request to the end server. A symmetric key is shared amongst peers in the path to ensure link-encryption within a crowd.

Web Mixes [2] is another more recent effort to apply a mix network to web browsing, using "mix-cascades" to form anonymous paths. While traditional mixes source-route messages, chains of servers pre-establish these mix-cascades. A sender will only select the first server of a mix-cascade, and his message will be routed deterministically through this cascade.

8.1.3 Anonymous publishing, storage, and file-sharing systems

The Rewebber [13] system encodes mix paths in URLs. These URLs contain the name of a Rewebber server and a packet of encrypted information. The server decrypts the packet to uncover the address of another Rewebber server or a legitimate web site. The Publius [28] censorship-resistant publishing system uses Rewebber to provide anonymous retrieval for document readers.

The Freenet system [5] seeks to build a network for file caching and retrieval. Peers chain file requests and responses through other peers in the system. In a technique reminiscent to Crowds, intermediate peers decide how to route messages based on some local knowledge of possible file location.

Free Haven [8] and Tangler [27] seek censorship-resistant publishing and storage. They require some underlying anonymous communication channel such as Tarzan to achieve their anonymity goals for all concerned parties.

8.2 Towards anonymous IP routing

While a number of systems have focused on application-specific Internet anonymity, only a few systems attempted to achieve anonymity for low-level, real-time communication.

The Onion Routing system [25, 26] creates a mixnet for TCP/IP connections. A sender creates a mix path by successively encrypting a control packet, or *onion*, with the public keys of several mix servers. These servers will store symmetric-key state information to handle normal data packets in the system. When a user places a message into the system, an *onion proxy* determines a route through the anonymous network and layer encrypts the message accordingly with the pre-established symmetric keys. This design motivates Tarzan's separation of tunnel establishment and packet forwarding.

Zero-Knowledge Systems (ZKS) developed the first commercial anonymous-communication system, known as the Freedom network [12]. The Freedom network was a series of nodes deployed by ZKS at various ISPs to route traffic between them, using the standard layered encryption mechanism over UDP. ZKS shut down the network in mid-2001.

Application integration is achieved in Onion Routing using separate application proxies; they propose at least sixteen such proxies for the second-generation system. Additionally, 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 and ties client-side integration closely to the operating system. Operating transparently at the network level by diverting packets using standard firewall rules, Tarzan can anonymize any client traffic.

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 cental 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 system for authentication and access control. Tarzan does not place such barriers at the network level and instead provides a library that 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, allowing 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.

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.

The Onion Routing system further proposes a means whereby users can supply *reply onions* similar to Mixmaster reply blocks, with which users can send recipient-anonymous messages. This mechanism is 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.

Chapter 9

Conclusion

Tarzan provides a flexible layer for sender, recipient, and relationship anonymity, even when operating in real-time. It sustains its anonymity properties 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, benefitting both anonymity and availability.

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

Tarzan's ability to participate in multiple kinds of traffic furthers its usefulness and, hopefully, adoption. We believe a free, open-source, and common 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.

Bibliography

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

[2] Oliver Berthold, Hannes Federrath, and Stefan Köspell. Web mixes: A system for anonymous and unobservable internet access. In Federrath [11], pages 115–129.

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

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

[5] Ian Clarke, Oscar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Federrath [11], pages 46–66. `http://freenet.sourceforge.net`.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Banff, Canada, October 2001.

[7] Wei Dai. Pipenet. `http://www.eskimo.com/˜weidai/pipenet.txt`.

[8] Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed anonymous storage service. In Federrath [11], pages 67–95. `http://freehaven.net`.

[9] John R. Douceur. 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] Hannes Federrath, editor. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

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

[13] Ian Goldberg and David Wagner. TAZ servers and the Rewebber network: Enabling anonymous publishing on the World Wide Web. *First Monday*, 3(4), 1998.

[14] Y. Guan, C. Li, D. Xuan, R. Bettati, and Wei Zhao. Preventing traffic analysis for real-time communication networks. In *Proceedings of Milcom '99*, November 1999.

[15] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '99)*, Atlanta, Georgia, 1999.

[16] David Mazières and M. Frans Kaashoek. The design, implementation and operation of an email pseudonym server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-5)*, pages 27–36, San Francisco, California, November 1998.

[17] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124–139, December 1999.

[18] July 1993.

[19] National Institute of Standards and Technology. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, 1994.

[20] National Institute of Standards and Technology. Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/NIST, 1995.

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

[22] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*, January 2001. RFC 3031.

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

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

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

[26] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In Federrath [11], pages 96–114.

[27] Marc Waldman and David Mazières. 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.

[28] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, August 2000.

[29] Hugh C. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, 1980.

[30] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. An analysis of the degradation of anonymous protocols. In *Network and Distributed System Security Symposium*, February 2002. San Diego, CA.