# Scalable Onion Routing with Torsk

Jon McLachlan
University of Minnesota
Minneapolis, MN 55455
mcla0181@umn.edu

Andrew Tran
Carnegie Mellon University
Pittsburgh, PA 15213
qtran@andrew.cmu.edu

Nicholas Hopper
University of Minnesota
Minneapolis, MN 55455
hopper@cs.umn.edu

Yongdae Kim
University of Minnesota
Minneapolis, MN 55455
kyd@cs.umn.edu

## ABSTRACT

We introduce Torsk, a structured peer-to-peer low-latency anonymity protocol. Torsk is designed as an interoperable replacement for the relay selection and directory service of the popular Tor anonymity network, that decreases the bandwidth cost of relay selection and maintenance from quadratic to quasilinear while introducing no new attacks on the anonymity provided by Tor, and no additional delay to connections made via Tor. The resulting bandwidth savings make a modest-sized Torsk network significantly cheaper to operate, and allows low-bandwidth clients to join the network.

Unlike previous proposals for P2P anonymity schemes, Torsk does not require all users to relay traffic for others. Torsk utilizes a combination of two P2P lookup mechanisms with complementary strengths in order to avoid attacks on the confidentiality and integrity of lookups. We show by analysis that previously known attacks on P2P anonymity schemes do not apply to Torsk, and report on experiments conducted with a 336-node wide-area deployment of Torsk, demonstrating its efficiency and feasibility.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

## General Terms

Algorithms, Security

## Keywords

Anonymous Communication, Peer-to-Peer Networks

## 1. INTRODUCTION

Anonymity on the Internet, the ability to conceal the identity of one or both parties to a communication, safeguards both freedom of speech and privacy. Constructing a system that provides anonymous communication in the face of attacks is a challenging security problem. One source of difficulty in building anonymity services is the essential fact that having a large number of users is a necessary, though not sufficient, condition for anonymity. Thus design choices that impact "secondary characteristics" such as performance, availability, reliability, and so on, also impact the security of a deployed scheme through their effect on user participation. One major line of research has thus focused on providing low-latency anonymity schemes that can support many desirable applications over a common interface. These schemes typically achieve end-user anonymity by passing messages through a series of relays, performing cryptographic processing at each hop, before forwarding connections to arbitrary Internet locations on behalf of the initiator.

Two related design variables within this general framework are the organization and selection of relays. Completely centralized schemes [2, 4, 7], where all traffic flows over a small, constant number of routes, offer trivial relay discovery, but have severe limitations in terms of the number of users they can support. At the other end of the spectrum, completely decentralized P2P schemes [15, 17, 24, 28] are potentially more scalable since every client also acts as a relay. However, the dynamic nature of peer participation, combined with a variety of attacks based on nodes having imperfect knowledge of the set of relays [8, 9, 36, 5], complicates discovery. An additional problem for such schemes is the fact that many nodes may be unable or unwilling to participate as relays.

As of 2009, Tor is the most widely-used low-latency anonymity scheme, with approximately 100,000 simultaneous users [19], and it represents a compromise between these approaches: Tor maintains a client-server architecture, with roughly 2000 nodes that relay traffic for all clients. Tor is an open system with centralized node discovery: any node can join as a relay or client, while a small set of trusted directory servers maintains a signed, authoritative list of active relays. This architecture has several advantages. The client-server architecture increases reliability: since nodes volunteer to act as relays, only relatively reliable nodes tend to do so. At the same time, participation is not harmed, since nodes that are unable or unwilling to reliably relay traffic can still participate as clients. The open nature of the scheme allows more nodes to volunteer and scale the available bandwidth as more clients join, while also diffusing the trust required in any single node. Furthermore, the centralized directory service makes the "selection" problem easy, since there is a global authoritative list of all relays.

**Tor's Quadratic Cost.** While Tor's design offers several benefits, it also has drawbacks compared to the potential benefits of pure P2P designs. The most prominent drawback is scalability: although treating clients and relays separately yields some improvement, Tor still has bandwidth costs that scale quadratically with the number of users. In particular, every client must periodically download a list

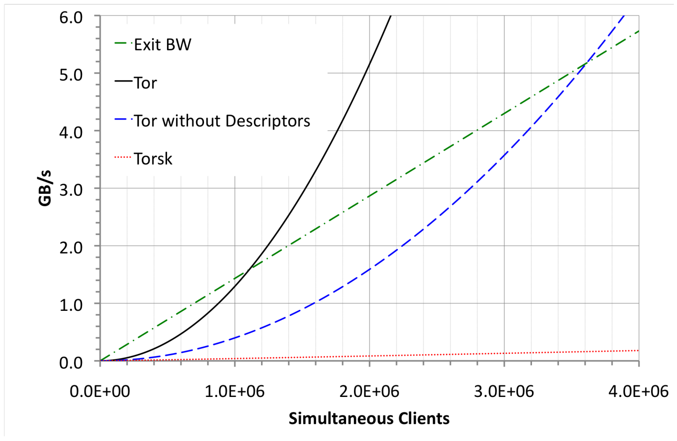

**Figure 1: Exit traffic, Discovery traffic (version 3 and without "descriptors"), and Torsk relay selection and circuit building traffic delivered, calculated by simulation, vs number of clients**

of all relays: if there are $n$ clients and $r$ relays then the bandwidth expenditure for discovery scales as $O(nr)$ whereas the bandwidth available for downloads scales as $O(r)$.

Figure 1 illustrates the limitations of this approach. As of July 2009, the estimated number of simultaneous Tor clients was roughly 100,000 [19]. The number of simultaneously available relays was roughly 1500 out of 2000, and the average bandwidth advertised by "exit nodes" that deliver data from the rest of the Internet to the Tor network was 137 MB/s. Each client downloads a 120KB "network status document" every 3 hours, and 1.25MB of router "descriptors" spread over 18 hours, for an aggregate bandwidth cost of roughly 1.1MB/s. However, if the number and bandwidth of routers scales linearly with the number of simultaneous users, increasing the number of clients by a factor of $c$ increases the bandwidth available by $c$ but the "directory" bandwidth by a factor of $c^2$. Thus, the two bandwidths become equal at roughly 1.2M clients; if Tor grows to the same size as popular P2P services such as Pirate Bay [1] (12M simultaneous) or Skype [37] (15M simultaneous), relays spend nearly an order of magnitude more bandwidth relaying descriptors to clients than relaying connections.[1]

**Introducing Torsk.** In this paper, we introduce Torsk, a new design for a low-latency anonymous networking scheme. Torsk improves the scalability of Tor, while retaining the benefits of its client-server architecture, and introduces no new attacks. In particular, the total bandwidth spent on relay selection is $O(n \log r)$, enabling Torsk networks to scale practically to much larger sizes. While the bandwidth spent by Torsk relays at the current network size is similar to that spent by Tor relays, the cost to clients is considerably smaller, and a modest increase in size dramatically increases the overhead of Tor compared to Torsk, as shown in Figure 1. Moreover, Torsk is incrementally deployable: relays running a single process can simultaneously serve Torsk and Tor clients.

To accomplish this, we distribute most of the Tor directory service over a distributed hash table, or DHT. DHTs have been suggested as a solution to the selection problem before [40, 17, 24, 21], but we note that it is nontrivial to build a secure random relay selection scheme from even a secure DHT, due to a variety of possible attacks [25, 8, 9, 22]. Thus part of our solution is to implement a DHT that is secure in the sense that lookups are guaranteed to correctly identify the node responsible for a key, even with adversarial interference. For this task, we adapt the cryptographic "neighborhood certification" tools from the Myrmic DHT algorithm [38] to the Kademlia DHT, which has been deployed in several systems with over a million simultaneous users. Myrmic guarantees security against an adversary that controls a constant fraction of the DHT, and requires a trusted authority, which we implement via the directory authority.

Every Tor relay joins this DHT, and we modify Tor circuit construction so that nodes pick "next hops" by finding the Tor relay closest to a randomly-chosen key. The Myrmic "root verification" algorithms allow a client to verify that a relay is the correct result without joining the network. Because of this, the client can use a partial tunnel to randomly select next hops. In order to preserve anonymity, we introduce a second mechanism whereby a Tor relay asks another, secret relay (its "buddy") to resolve a key in a way that, combined with cover lookup traffic, provably ensures the privacy of the selection procedure. We implement these procedures as an extension to Tor using the "control port" interface. Thus relays and clients can join the Torsk network without changing Tor binaries, allowing for incremental deployment. We argue that any attack against the anonymity provided by Torsk can be applied to Tor with nearly the same probability of success.

In terms of performance, the main implication is that extending circuits becomes more time-consuming. Each circuit extension requires a DHT query, two additional round-trips through a (partial) Tor circuit, and sends about 2KB of additional data through the partial circuit (a single "neighborhood certificate" of about 1KB and a router "descriptor" of about 1KB). In most cases this cost can be avoided by proactive circuit-building; however, startup time and hidden service performance will be noticeably impacted.

Eliminating the authoritative list of relays has several additional benefits. Since clients no longer need to obtain the network status list plus most of the current router descriptors (around 500KB currently) in order to startup, lower bandwidth clients can participate more easily in Torsk. Blocking access to the directory is also no longer feasible, except by blocking the much larger Kad network.

**Outline.** Section 2 sketches the details of the Tor, Kademlia, and Myrmic protocols necessary for understanding Torsk. Section 3 lays out the design and security goals of our system. We then discuss the design in more detail in section 4, and give a brief security analysis in Section 5. Section 6 reports on simulations and performance measurements of our implementation of Torsk. We discuss related work in Section 7 and conclude with Section 8.

## 2. BACKGROUND

### 2.1 Tor

Tor is a low-latency and bandwidth-efficient anonymous relay service for TCP streams. Its deployment and continued growth have provided a valuable testbed for research ideas in anonymity and traffic analysis, in addition to demonstrating the demand for anonymity. The central service provided by Tor is an encrypted *circuit*, over which all communication during a given session takes place. Anonymity is achieved by establishing a circuit through three nodes: an entry node, an intermediary (middleman), and an exit node. In principal, the entry node knows the identity of the client contacting it, in the form of its IP address, but not the identities of the servers the client connects to; the middleman node knows only the identities of the entry and exit nodes; and the exit node, which provides a gateway between the Tor network and the Internet, knows the server(s) a circuit connects to but not the client or

[1]We note that the Tor project is in the process of implementing a "microdescriptor" service that will reduce or eliminate the cost of downloading descriptors. This changes the constant in $O(nr)$ but not the quadratic behavior, as shown in Figure 1.

entry node. In this manner, no single Tor node knows the identities of both communicating parties associated with a given circuit.

The construction and use of circuits involves three types of nodes: clients (referred to in the Tor documentation as "Onion Proxies", or OPs), relays ("Onion Routers" or ORs), and the Directory Servers (DS). Circuits are established iteratively by the client, who begins each session by downloading the list $L$ of all Tor relays, in the form of "descriptors" that specify the long-term public key, IP address, exit policy, and other details of a relay, from the Directory Service. In the first iteration, the client selects a Tor relay $R_1$ from $L$, and performs an authenticated[2] Diffie-Hellman key exchange with $R_1$ to establish an encrypted connection. At each subsequent iteration $i \in \{2, \ldots, N\}$, the client randomly chooses another relay, $R_i$, from $L$, and sends $R_{i-1}$ an encrypted key exchange message to be relayed to $R_i$, extending the relay chain to $R_i$. Tor uses a default value of $N = 3$. Thus $R_1$ is the "entry node", $R_2$ is the "middleman node" , and the final relay, $R_N$, is the "exit node" and serves as the interface between the circuit and any TCP-based Internet services the client might wish to contact.

We note that clients reuse each circuit for multiple TCP streams, and proactively build new circuits so that when older circuits are retired – after 10 minutes – the new circuits can be used immediately. Thus circuits are *not* built on-demand, and in most cases the multi-second latency incurred to build a circuit does not impact the latency of connections made via Tor.

Relays participate in circuits by responding to requests and forwarding encrypted cells as described above. In addition, a relay periodically tests its reachability by building a tunnel in which it serves as a relay. The results of this status testing, along with a new, complete "descriptor" are reported every 18 hours to the directory servers, which update the directory appropriately.

Directory servers do not participate directly in circuits but must be periodically contacted either directly or through a "directory cache" by every client and every relay. They periodically (every hour) sign an updated "consensus network status" list giving the status and hashed descriptors of all relays, which are downloaded at regular intervals (on average, every three hours) by all clients.

## 2.2 Kademlia and Myrmic

Kademlia [20], or *Kad* is a distributed hash table (DHT) algorithm that has been implemented and deployed in several systems with millions of simultaneous users. As is the case with any DHT, Kad implements a distributed binding service that stores a list of "values" for each "key" in a virtual hash table. In Kad, every node has a uniformly chosen, 128-bit ID; every data item (i.e., a [key, value] binding) stored by the Kad network has a 128-bit key. All bindings for key $x$ are stored at nodes with IDs close to $x$, where the distance between IDs $k_1$ and $k_2$ is the integer $k_1 \oplus k_2$.

To efficiently route queries for a given key, every Kad node maintains a routing table with $O(\log(N))$ entries where $N$ is the size of the network. The $i$-th routing table entry is a list of up to $k$ nodes that share at least an $i$-bit prefix with the node ID of the owner. Kad uses parallel, iterative lookup: When node $Q$ queries key (or node ID) $x$, it finds the $\alpha$ contacts from its routing table closest to $x$. $Q$ consults these contacts in parallel, which each return $k$ of their contacts close to $x$. Next, $Q$ picks the $\alpha$ closest contacts from this set, repeating this procedure until it finds $root(x)$, the node closest to $x$.

A seemingly obvious approach to solving the relay selection problem would be to have Torsk relays form a DHT and allow clients to select relays by picking a random key $k$ and searching for $root(k)$. However, Kad lookup has several vulnerabilities that would allow adversarial nodes to bias the outcome of this procedure.

[2]by the public key specified in $R_1$'s descriptor

Myrmic [38] is a DHT routing protocol provably robust against active attacks. A key feature distinguishing Myrmic from other DHTs is a *root verification protocol* that allows anyone to verify that the node responding to a query for key $k$ is indeed the "correct" holder of the key. Myrmic has the same semantics as Chord and in a network with no malicious nodes it has message cost and latency that are provably at most twice the cost of Chord with recursive routing. Wang *et al.* [38] demonstrate both experimentally and analytically that good performance is maintained even when a large fraction (for example, 30%) of nodes behave maliciously. However, Myrmic lacks the wide deployment of Kad. Therefore, in Section 5, we show how to overlay a Myrmic network within a Kad DHT to provide an efficient yet provably robust DHT routing algorithm.

Myrmic requires a new online authority, called the *Neighborhood Authority (NA)*, which only participates in DHT network management by issuing *Neighborhood Certificates (nCerts)* to small sets of nodes after DHT membership events such as joins and leaves. The $NA$ is *not* involved in any other aspect of DHT routing, and in particular queries do not involve the $NA$. The $NA$ has a public/private key pair for signing certificates and it is assumed that its certificate is publicly available. If the $NA$ goes offline for some period of time, there are two effects: new nodes will be unable to join the network (but can still route queries through existing network nodes), and the proportion of nodes that become "faulty" due to churning out will increase.

Like Kad, Myrmic uses *iterative routing* in order to allow a querier to monitor query progress and to find alternative routes in case its query is mis-routed or dropped. Securing iterative DHT routing requires mechanisms to verify that a query for key $k$ makes progress and terminates at the correct destination. Myrmic allows a querier to verify that node $n$ is currently responsible for key $k$ using an nCert issued by the $NA$, which attests to the set of nodes closest to $n$ (and thus the range of keys $n$ is responsible for). As long as nCerts can be revoked when membership changes, this prevents a malicious node from claiming to be responsible for a key outside its range or routing a query to an incorrect node. nCert revocation is handled by storing the most recent nCert for a node at its neighbors, which are then listed in the node's signed nCert. When a nCert is invalidated by a change in membership, those neighbors are informed. Hence as long as a malicious node has one honest neighbor,[3] it cannot use a revoked nCert since the querier contacts every neighbor directly when searching for a more recent certificate. We stress that Myrmic's membership protocols maintain perfect consistency between nodes' nCerts by imposing the view of a single party on the entire network.

## 3. REQUIREMENTS

**Functional Requirements.** The goal of Torsk is to improve the scalability of Tor's relay selection algorithm by eliminating the authoritative list of relays. In contrast to most existing work on P2P anonymity, we explicitly retain Tor's central authority and its client/relay distinction. This leads to the following design goals:

*Reduced relay overhead.* Relays should not be required to know the full, current membership of the Torsk network, and overhead for relay selection should grow slowly with the number of relays.

*Low-cost bootstrapping.* Clients should be able to bootstrap and build circuits by knowing or finding a single Tor relay in the Torsk network. This significantly reduces the bandwidth requirement for startup, allowing Tor to be used over limited-bandwidth channels.

[3]Myrmic also includes protocols that allow the DHT to quickly recover from the occasional event that all the nodes in a neighborhood become faulty.

At the same time, additional delay in circuit construction and OR discovery should be minimized.

*Incremental Deployability.* The protocol should be incrementally deployable: relays should be able to participate in both networks with no overhead while providing at least the same level of anonymity as a standalone network.

**Security Requirements.** The security goal of Torsk is to maintain the level of anonymity provided by Tor, while allowing improved scalability. Thus we are concerned with the same essential threat model as Tor: a "local" adversary that can control a small fraction $f$ (e.g. $10\%$) of ORs and possibly see the traffic at one end (but not both) of a tunnel, but who cannot directly observe or interfere with the traffic between honest nodes.

Thus our goal is that any attack against Torsk should correspond to an attack with similar probability of success against a Tor network of similar size. Since we do not alter Tor's mechanisms for packet forwarding, key exchange, or encryption, we focus on possible attacks caused by the decentralization of route selection. This use of DHT-style routing, in Torsk as well as other schemes [21, 24, 17, 40], introduces the possibility of clever, new attacks that require only a small fraction of colluders. Here we outline the known classes of generic attacks on such P2P schemes:

*Route capture by misrouting.* Malicious nodes may return only malicious nodes as the result of lookups; by the design of the routing protocol, no previous node in the lookup will know a node closer than the closest malicious node. DHT lookup mechanisms not specifically designed for security [34, 29, 26, 20] are vulnerable to this attack, first described by Castro *et al.* [6]. Several possible countermeasures have been proposed, including "density tests" [6], redundant routing [24], "root verification" [38], and byzantine fault-tolerant simulation of nodes [14, 18]. These countermeasures make tradeoffs between efficiency, trust, and adversarial capabilities, and require careful evaluation when applied to anonymity.

*Passive Logging.* Every query in a DHT involves $O(\log n)$ peers in the DHT. In DHTs with iterative lookup, like Kad and Myrmic, the node making the query communicates directly with each of these nodes, and even when "recursive" lookups are employed some information about the source of a request is often leaked [25, 22], which may be exacerbated by redundant lookup mechanisms. An adversary that controls a constant fraction of nodes will thus participate in an overwhelming fraction $(1-1/n^c)$ of all lookup requests. Even if correctness of these lookups is enforced by the protocol, it is natural to consider attacks based on correlating the set of keys queried by nodes at various times. Thus, for example, it is obviously insecure for the client to query the DHT directly for all of the relays in a circuit, since then a malicious exit node can use the time relationship between the client's query and a subsequent circuit to infer the origin of a circuit. More subtly, if the client searches for an entry node, and then the entry node searches for a middleman, and the middleman searches for an exit node, the timing of these events might still reveal the origin to a malicious exit node.

*Selective Dropping.* Even assuming the source-privacy and integrity of lookups, an adversary might try to bias result of lookups by selectively dropping query requests that are not directed towards adversarial nodes. Even if only a constant fraction of requests are dropped in this way, the resulting bias will be noticeable. Furthermore, some mechanisms proposed to mitigate query dropping actually result in decreased anonymity against selective dropping attacks. For example, Borisov *et al.* [5] show that such an attack can have devastating consequences for Salsa [24].

*Route Fingerprinting.* Danezis and Clayton [8, 9] show that if each node selects its routes from a consistent subset of all nodes, and these subsets are known to the adversary, then he can use this knowledge to uniquely identify the source of a tunnel by any pair of consecutive links on the tunnel. Since each node in a P2P scheme has limited knowledge of the network and node discovery is potentially observable a P2P scheme must avoid leaking this knowledge in order to preserve anonymity.

## 4. DESIGN OF Torsk

As explained above, in Torsk, we replace the directory service with OR lookups via Kad, using Myrmic to secure the lookup process. With this change, clients no longer have any communication with the directory service or caches. ORs, however, are required to join the Kad network and must obtain nCerts from the $NA$, which is implemented by the directory servers. Relays are then discovered by ORs via Kad search during circuit construction. Several additional challenges must be addressed to meet our security requirements. We outline the details of the Torsk design in this Section, and discuss its security and other considerations in Section 5.

**Torsk nCerts.** The nCert for node $R$ has the format:

$$\begin{aligned} nCert_R &= Sign_{sk_{\mathbf{ds}}}\{nList_R, rList_R, t_{issue}, t_{exp}, Desc_R\} \\ nList_R &= \{I_{n^l(R)}, \ldots, I_{n^1(R)}, I_R\}, \\ rList_R &= \{I_{r_1}, I_{r_2}, \ldots, I_{r_d}\}, \\ I_S &= (nodeID_S, pk_S, IP_S) \end{aligned}$$

where $n^j(R)$ denotes the $j$-th closest (by XOR distance) Torsk node to $R$. An nCert is signed by the $NA$ (the Directory Server) using a secure digital signature $Sign_{sk_{\mathbf{ds}}}(\cdot)$. The nCert also includes its issue time, its expiration time, and $R$'s Tor descriptor, $Desc_R$. Finally, $nCert_R$ contains two lists of Torsk nodes, $nList_R$ and $rList_R$, encoded as triples $(nodeID_i, pk_i, IP_i)$ which enable others to make direct IP connections to the nodes. The $nList_R$ nodes are the nodes with nodeIDs closest to $R$'s and allow us to verify whether $R$ is the root of a given key $x$ as in Myrmic. The $rList_R$ nodes are randomly chosen by the $NA$ and facilitate the "buddy selection" protocol described below.

**Secure Kad Lookup with nCerts.** Suppose that all ORs on the Kad network have nCerts and they are uniformly distributed over the Kad network.[4] We improve the security of Kad lookup as follows. We first locally modify Kad's iterative lookup process. In the original Kad protocol, lookup is parallel. However, each round of the parallel lookup uses only the best results, allowing an attacker to perform a denial of service attack by returning many non-existent nodes with IDs close to the target [39]. Torsk iterative lookup fixes this problem by maintaining distinct "best result" lists for each of the three parallel paths. Once the querier finds a Torsk node that is close to its query, it can request the node's nCert and use the nCert to identify the root, $R$. The querier then verifies that $R$ is the correct result by verifying the nCert digital signature and contacting $R$'s neighbors to verify that the nCert is fresh. We also improve Kad search performance by using "soft timeouts" as in Myrmic: when the querier does not receive a reply from node $R$ within a short period (e.g. 120 msec, the median latency between US PlanetLab nodes), it sends a query to another node immediately rather than waiting for Kad's one-second "hard timeout."

**Joining the Torsk Network** A new OR $R$ joins the Torsk network as follows. It first generates a Kad nodeID. It then performs the secure Kad lookup described above on its own ID, which allows it

[4]Bootstrapping involves simultaneously issuing nCerts to a small initial set of $4l+1$ ORs, after which every OR can join the network and obtain an nCert.

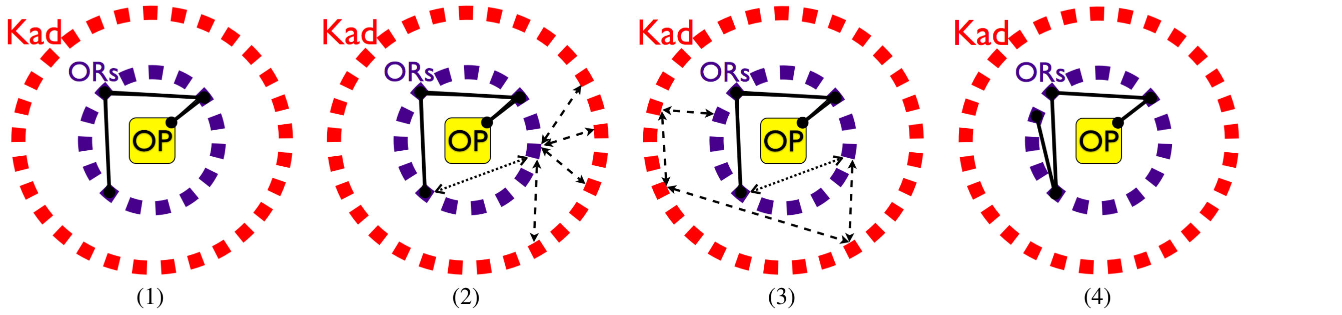

**Figure 3: Extending a circuit: (1) Initially, the OP has a partially constructed circuit. (2) The OP sends a tunnel request to the ultimate OR to lookup $x$, a random Tor NodeID. The OR's buddy includes $x$ among its $l$ Kad lookups at its next lookup time. (3) One of the buddy's lookups finds $R$, the node closest to $x$. (4) The OP verifies the NA's signature on $nCert_R$ and extends the partially constructed circuit to a random node from $nList_R$.**

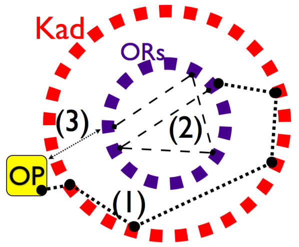

**Figure 2: Client bootstrapping process: (1) search for a random Torsk NodeID $R$ in Kad (2) buddy selection starting from $root(R)$ (3) Use buddy as Entry Guard.**

to find the nCert of its closest neighbor, $R'$. $R$ then sends its ID and descriptor along with $nCert_{R'}$ to the directory server, who verifies the signature and freshness of $nCert_{R'}$, builds a new neighborhood map including $R$, chooses fresh $rList$ nodes,[5] and issues new nCerts for all nodes affected; each fresh nCert is sent to every node that appears in $nCert.nList$ and $nCert.rList$. $R$ also follows the standard Kad bootstrapping protocol to join the Kad network. Once an OR has joined the Torsk network it periodically pings its neighbors; if it detects that a node has left it informs the Directory Server, which repeats the joining process to issue new nCerts.

**Buddy Selection Protocol** The last missing piece before we introduce our decentralized circuit construction is the buddy selection protocol, which plays the important role of preventing passive correlation attacks. Each OR maintains a list of "lookup buddies," which will be used to perform secure Kad lookup of random ORs on behalf of clients. Upon joining the Torsk network, $R$ begins the following "random walk" process to select a buddy. Initially it sets $R_0 = R$ and chooses a walk length $\ell$ as described in Section 5.2. At each step $i \in \{0, \ldots, \ell\}$, $R$ asks $R_i$ for a list $S_i$ containing the nCerts of all incoming and outgoing $rList$ entries for $R_i$, as well as $R_i$'s nCert. If all of the nCerts have valid signatures, and are consistent with the previous hop, then $R_{i+1}$ is chosen uniformly from $S_i$. The list $S_i$ is "consistent" with the previous hop if either: (1) the node $R_i$ was selected from $rList_{R_{i-1}}$ and $S_i$ contains $nCert_{R_{i-1}}$; or (2) $S_{i-1}$ contains an nCert for $R_i$ with $R_{i-1} \in rList_{R_i}$ and the $nCert_{R_i}$ contained in $S_i$ is at least as fresh. If at any time an invalid nCert (e.g. the signature does not verify) is obtained or an inconsistent $S_i$ (that does not include the previous hop) is found, the random walk starts over. Otherwise, the output of the buddy selection protocol is the nCert of the node $R_\ell$. An OR repeats this process to find new buddies as needed.

[5]The $rList$ nodes can be chosen via the buddy selection protocol, or for efficiency, the directory server can cache the list of current ORs, as in our current implementation.

**Decentralized Circuit Construction.** Once the ORs have bootstrapped the Torsk network, clients can begin building circuits. We note that we have provided two methods of finding a random Torsk node, buddy selection and secure Kad lookup; each has slightly different security properties. Kad lookups are "loud:" an adversary that can run even a small constant fraction of the Kad nodes will see a large fraction of lookup attempts; however, the results are also verifiable to a third party. Random walks are private, in that even nodes that are intermediate stops on the walk cannot distinguish the final node from a random one; however, the result of a random walk is not verifiable to an external party.

For the first hop, shown in Figure 2, random walks are ideal: the client can perform the walk herself and be convinced of the randomness of her selection. This reveals the identity of the client node, but in Tor the first hop *always* knows the identity of the client. The client can use Kad lookup or caching to find an nCert for use as a starting point in the random walk. As in Tor, Torsk clients can cache a small number of nodes to use as entry guards, and avoid this first lookup process in most cases.

To extend a circuit, there are two possible lookup methods and several candidate nodes to perform the lookup. We can rule out several possibilities: the possibility of passive monitoring means that the client should not select next-hops by Kad lookup. Also, the client should not select next-hops by random walk, because the timing of the random-walk request and the circuit extension will reveal the identity of the client. Having the current last hop select a next-hop by random walk would allow route capture, since the client can't verify the randomness of the result, and having the current last hop select a next-hop by Kad lookup leaks the sequence of a circuit through the time correlation of lookups – for example, if node $R$ does a lookup for node $S$ and then node $S$ does a lookup for $T$ then $R \rightleftarrows S \rightleftarrows T$ is a likely circuit.

Our solution, shown in Figure 3, uses a combination of both search types along with cover traffic to avoid these weaknesses. To discovery a next-hop OR for a circuit, the client generates a nodeID for a random OR. This key, when searched for, should yield a randomly selected OR. Instead of searching for this key herself, the client passes it through the partially constructed Tor circuit to the ultimate OR $R_i$. $R_i$ then asks one of its buddies (discovered via random walk) to perform a Kad lookup on $k$. The buddy returns the resulting nCert to $R_i$. $R_i$ returns the nCert to the client via the established tunnel, and the client finally chooses $R_{i+1}$ uniformly from $nCert_{root(k)}.nList$. The OR then discards the buddy.

**Buddy Lookups.** ORs ask buddies to perform lookups in order to hide the correspondence between the current end of a tunnel and its next hop. However, the timing of lookups could also leak this correspondence: if client $C$ does a lookup for $R_1$, and this is immediately followed by a lookup for $R_2$, which is followed by a lookup for $R_3$, it is easy to infer the parties to the constructed tunnel. In order to prevent this, Torsk nodes use constant-rate cover traffic to conceal the correlation. Each node performs $\kappa$ lookups for Torsk kadIDs at a global constant period of $\tau$ seconds. When a node receives a "buddy lookup" request for key $k$, the key is queued and released at the next lookup with an available lookup slot; a minimum of one slot plus any additional empty slots are filled by randomly chosen Torsk Kad IDs. Since an uncorrupted buddy is essentially uncorrelated to the OR requesting the key, and the requested keys and cover keys are independently chosen from the same distribution, the resulting set of lookups contains no information about the correspondence between routers and next-hops.

## 5. SECURITY ANALYSIS & DISCUSSION

### 5.1 Parameter Selection and Impact

In this section, we explore the secure and efficient selection and impact of several important system parameters, assuming $n$ clients and $r$ routers, of which a fraction $f$ are compromised. We note that selection of Myrmic-specific parameters such as the size of the $nList$ and lifetime of an nCert is addressed by the original Myrmic paper [38]: setting $nList$ size to $\log \frac{1}{f}\left(\log \frac{1}{\delta} + \log \log_2 r\right)$ will ensure that lookups fail with probability at most $\delta$ when $f$ fraction of routers are corrupted; and nCert lifetimes should exceed the median router session time to minimize unnecessary updates.

**Random Walk Parameters:** Let $d$ be the length of the $rList$ field in each nCert. We model the undirected graph that has an edge between $n_1$ and $n_2$ if either node appears on the other's $rList$ as a random graph of average degree $2d$. We want to pick $d$ and the length $\ell$ of the random walk so that the distribution on the final node $R_\ell$ of the walk is close to the uniform distribution. The relevant bound we will use is the well-known fact that if the graph $G$ has normalized second eigenvalue $\lambda(G)$, then the statistical distance between $R_\ell$ and the uniform distribution is at most $\lambda^\ell$. Solving to make this distance smaller than $\frac{\epsilon}{\kappa r}$ will guarantee that the joint distribution on buddies is $\epsilon$ statistically close to uniform.

| $d$ | $r$ | $\lambda$ | $\lambda_{\max}$ | $\sigma$ |
|---|---|---|---|---|
| 4 | 1000 | 0.6578 | 0.6656 | 0.0027 |
| 4 | 2000 | 0.6592 | 0.6641 | 0.0017 |
| 8 | 1000 | 0.4810 | 0.4894 | 0.0023 |
| 8 | 2000 | 0.4824 | 0.4859 | 0.0014 |
| 16 | 1000 | 0.3446 | 0.3492 | 0.0018 |
| 16 | 2000 | 0.3462 | 0.3496 | 0.0011 |

**Table 1: Average ($\bar{\lambda}$), max ($\lambda_{\max}$) and standard deviation ($\sigma$) in, normalized second eigenvalues, 200 random $rList$ graphs.**

Thus it remains to find the appropriate values of $d$ and $\ell$. It is known that for almost all *regular* graphs of degree $\mathfrak{d}$, the normalized second eigenvalue $\lambda(G)$ is $2/\sqrt{\mathfrak{d}}$ [16], and Feige and Ofek [13] have shown that for graphs with constant average degree $\mathfrak{d}$, there is some constant $c'$ such that almost all such graphs have $\lambda(G) \leq c'/\sqrt{\mathfrak{d}}$. Since the $rList$ graph is similar but not identical to these distributions with $\mathfrak{d} = 2d$, we performed a series of simulations using the exact distribution; the results are summarized in Table 1. We found that the known bounds for regular graphs are a good estimate for $rList$ graphs; for instance using $d = 8$ with a network of size 2000 resulted in an average second eigenvalue of 0.4824 and a maximum of 0.4905 compared to the expected upper bound of $\frac{1}{2}$ in case of a degree-16 regular graph. Thus a good choice by these criteria, for instance, would set $d = 16$, $\ell = \lceil \frac{2}{3}(\log_2 r + \log_2 \kappa + \log_2 \frac{1}{\epsilon})\rceil$.

There are two additional considerations. First, adversarial nodes could try to alter the expansion properties of the $rList$ graph by, for instance, not reporting incoming links. However, an adversarial node that drops a single incoming link will be caught with probability roughly $1/2d$, the probability that the honest node followed the given link to the adversary. Once such dropping is caught, the honest node should remove the adversarial node from its view of the $rList$ graph; it has been shown that such "adversarial" node faults have a very small effect on the expansion properties of a graph [3], and we have verified experimentally that the expected value of $\lambda(G)$ after removing up to 10% of nodes has little change - for instance in 10 such runs with $r = 512, d = 8$ the average $\lambda(G')$ was 0.4965. Thus, such local dropping affects at most a constant number of random walks per node.

The second consideration concerns the adversary's ability to deduce how many steps $R$ will walk after contacting an adversarial node. If this number is small, the adversary will have a much higher confidence in the identity of $R$'s buddy. Our solution is to add a geometrically distributed "tail" with expected length $\ell$ to the random walk, so that at any step the node $R$ can be expected to walk another $\ell$ steps. The expected length of the walk is then $2\ell = \frac{4\log_2 \frac{r\kappa}{\epsilon}}{3}$.

**Cover Lookup parameters.** Let Torsk nodes build circuits with frequency $\phi$ (this frequency is $1/600$ Hz in Tor). Then in order to serve the demand of clients, the total rate of lookups by routers must exceed the total rate by clients, e.g. $\frac{\kappa-1}{\tau}r > n\phi$. Subject to this constraint, the period $\tau$ between "cover lookups" by a Torsk node represents a tradeoff between circuit construction latency and bandwidth overhead – a shorter period means lower latency but higher cover traffic – and the number $\kappa$ of parallel lookups (which must be at least two to ensure uniform cover traffic) trades off cover traffic for the ability to deal with "bursty" circuit building in which multiple ORs request a lookup during the same time period. For our implementation we chose $\kappa = 3$ and $\tau = 20$ seconds. The bandwidth consumed by these lookups, $O(r \log r)$, matches the dominating term in the overall bandwidth consumption of Torsk, but independent cover traffic is a key element in our security argument. An important open question is whether a different mechanism for cover traffic can provide similar guarantees at lower cost.

### 5.2 Security Analysis

We discuss briefly how Torsk deals with the attacks from Section 3, assuming fraction $f$ of malicious ORs, and then give a brief argument for security against other attacks on the anonymity of Torsk lookups. One type of attack that we explicitly *do not* consider is attacks based on *resource consumption*, for example "filling up" a router's "buddy lookup queue" to increase the latency of circuit construction. Such attacks are not considered for the basic reason that similar attacks can be performed against Tor (e.g. requesting multiple circuit extensions through a router) and that if such attacks become a problem, they can be mitigated using standard rate-limiting techniques.

**Route capture by misrouting:** Myrmic nCerts prevent an adversary from incorrectly claiming to be the root of a key, with high probability. However, in the rare case that a node chooses a random Torsk ID $k$ such that: (1) currently, $root(k)$ is not malicious; (2) however, $root(k)$ recently joined the network and its nearest neighbor $N$ is malicious; and (3) the buddy, $B$, performing the lookup is malicious, $B$ can return the revoked but unexpired nCert showing $N$ as $root(k)$. The probability of this event is $f^2\rho$, where $\rho$ is the fraction of nodes in the network with revoked but unexpired nCerts, a function of OR churn and nCert lifetime. To eliminate this bias, the client picks a random node from $nCert_{root(k)}$. In expectation, this node will be malicious with probability $f$.

**Passive Logging:** An adversary that controls fraction $\mathfrak{f}$ of the Kad nodes, will see $1 - (1 - \mathfrak{f})^{O(\log k)}$ of all Kad lookups. Thus if $\mathfrak{f} = \Omega(1/\log k)$, the adversary will see a constant fraction of all Kad lookups. As discussed in Section 4, the sources and timing of these lookups could potentially leak information about a tunnel, so Torsk specifically hides this information. To prevent the leakage of a tunnel's current endpoint, Torsk performs lookups through uncorrelated, single-use buddy nodes. To prevent the leakage of a tunnel's next endpoint, we use cover traffic to generate $\Omega(r)$ independently distributed lookups.

**Route Fingerprinting:** Route fingerprinting and other epistemic attacks [9] work based on an adversary's ability to infer the small set of nodes known to each client. Torsk is not directly vulnerable to such attacks because this set is unknown to the adversary; the combination of random walks, buddy lookups and cover traffic make the adversary's view of this distribution $\epsilon$-close to uniform.

**Selective Dropping:** Since Myrmic guarantees that $99.99\%$ of lookups correctly identify the root of a key even with a constant fraction of nodes behaving adversarially, selective dropping of DHT queries is not a concern. However, the introduction of buddies into the lookup process increases the risk of selective aborting of circuits compared to Tor: five nodes have the opportunity to drop a circuit rather than 3. In particular, a buddy can cause a lookup failure either because the entry node is not malicious, or the nCert that will be used to select an exit node does not contain enough malicious nodes. Figure 4 shows a comparison, between Torsk and Tor, of the fraction of compromised tunnels as a function of the fraction of malicious nodes. We note that the effect of the attack is very slight for $f \leq 0.15$. Finally, we note that the attack can be mitigated by replicating all lookups through $m$ independent buddies, so that dropping a lookup requires collusion between the buddies; this change in the lookup distribution requires a corresponding change in cover traffic. Figure 4 shows the case $m = 2$.

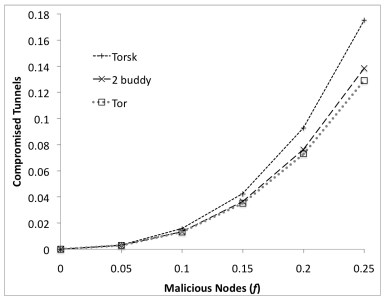

**Figure 4: Fraction of compromised tunnels vs malicious nodes under selective DOS attack.**

**Other Attacks.** The introduction of a formal model that covers all attacks on low-latency anonymity schemes is beyond the scope of this paper, but we argue informally that additional attacks against Torsk that do not apply to Tor are unlikely. First, we note that because Myrmic converts arbitrary DHT misbehavior to query dropping with low probability of success, any attacks that bias the selection of relays can only have a small effect on the success probability compared with an attack on Tor. Second, we note that the protocols for buddy selection and cover traffic provably produce lookup transcripts that are statistically close to uniform. Thus any attack that uses this information to attack the anonymity provided by Torsk will work with nearly the same probability (plus or minus the statistical distance to uniform) against Tor, when provided with simulated lookup and random walk requests drawn independently from the uniform distribution. The remaining aspects of the Torsk and Tor protocols are identical.

## 5.3 Practical Considerations

**Exit Policies and Load Balancing.** As described so far, our design forces clients to choose circuit participants from the uniform distribution. However, two important components of the success of Tor are its ability to allocate traffic proportionately to available bandwidth and its ability to support *exit policies* stating to which ports a node will route IP traffic. We briefly describe one mechanism to address these issues.

By default, Tor clients selects a router with probability equal to the fraction of total available bandwidth that the router provides (or claims to provide). Recently Snader and Borisov [30] have proposed an alternate design that selects routers according to their relative ranking. Torsk can naturally and flexibly support either policy by assigning routers Kad IDs that encode this information. Specifically, the Tor bandwidth allocation mechanism can be simulated by assigning Kad IDs so that nodes with similar bandwidth have similar prefixes. The density of a prefix and the weight assigned to a prefix can then be adjusted so that the probability of picking a node (its prefix weight divided by the number of nodes it shares the prefix with) is proportional to its share of the bandwidth. Likewise, the Snader-Borisov allocation method can be simulated by setting the first $k_{prefix} \leq (\log_2 r)$ bits of a router's ID to the most significant bits of its relative rank and then choosing a rank according to the appropriate probability distribution.

A similar approach can be used to deal with exit policies. On April 20, 2009, there were 1412 active Tor routers in the "consensus" network status. Of these, 769 did not support exit traffic, 169 supported "whitelists" allowing a small number of commonly used ports and the remainder supported "blacklists" disallowing a small number of ports. There were 12 port "ranges" supported by fewer than 50% of the exit nodes. Thus a reasonable strategy would be to assign distinct two-bit prefixes to non-exit nodes, "whitelist" nodes, and "blacklist" nodes. Since Torsk builds circuits pro-actively each client can always maintain a circuit ending at a router of each type. Streams can be assigned to the most restrictive circuit supporting the appropriate port, causing the client to proactively build the same type of circuit at the next circuit-building interval. Finally, the Torsk DHT can be used to store a list of routers supporting "exceptional" ports, and clients can maintain a "partial" circuit to be extended to a third hop supporting such ports if needed.

Recent work [12, 23] has suggested that there are additional variables that may be important to the security and utility of path selection, such as AS paths, Internet exchanges, and interhop latency. We note, however, that currently there are no known methods of efficiently incorporating such considerations into Tor path selection[6], and leave the adaptability of Torsk to these concerns as future work.

**Public vs Private DHT.** An interesting consequence of using the Kademlia DHT algorithm is that the Torsk DHT may be run independently or embedded inside an existing Kad DHT. The advantage of doing so is that there are several well-studied implementations, and several Kad DHTs with millions of simultaneous users [35, 33, 32, 31], which can considerably raise the cost (in terms of bandwidth) and difficulty of conducting passive eavesdropping attacks, identifying Torsk routers, and identifying Tor connections through IP profiling. The potential disadvantages of doing so include the need to route DHT traffic for non-Torsk clients, and the potential for the existing DHT to develop methods to detect and drop Torsk requests. To demonstrate the feasibility of embedding Torsk in an existing Kad DHT, our prototype was developed to interoperate with the aMule/eDonkey DHT [10], which supports a file-sharing network with 2 million concurrent users. One technical issue that arises is making the small set of Torsk routers easy to find among

[6]Tor avoids using two nodes with the same /16 IP prefix, but this approach is neither necessary nor sufficient to deal with AS or Internet-exchange level adversaries

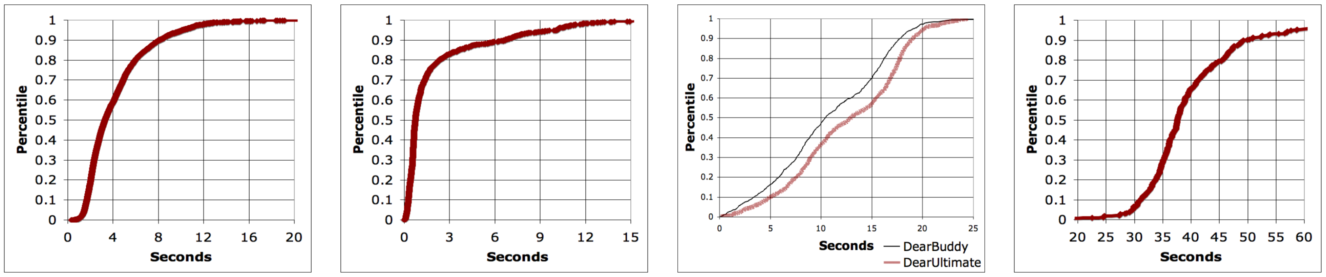

**Figure 5: Results of PlanetLab run with 336 nodes, with 40 constructing circuits. CDFs of (a) 16-hop random walk latency, 6400 measurements/node. (b) Kad search latency, 2200 per node. (c) Per-hop, and (d) Total circuit construction latency, 1.4k circuits**

the larger DHT. Our implementation resolves this issue using special Kad IDs, as described in the Appendix.

## 6. IMPLEMENTATION & EXPERIMENTS

As a proof-of-concept, we have implemented Torsk and deployed it on a private PlanetLab-based network. We report the key features of our implementation and its performance.

**Architecture Overview.** Our prototype, which spans approximately 3000 lines of new code, has four modules that run as separate processes and communicate via IPC on each Torsk OR, as well as the Neighborhood Authority, which runs on the Directory server. These modules are Tor, Kad, Myrmic, and the Controller, or "Main" module. Here we briefly describe each module.

Main Module The multi-threaded Main module, written in Python, implements the core circuit construction and random walk functions and also manages the initialization, bootstrapping, and IPC between all components. All protocol logic specific to Torsk is contained in the Main module.

Kad and Myrmic. The Kad and Myrmic modules are responsible for all DHT functions. The Kad module is based on the KadC and aMule libraries, and implements the modified search process described in Section 4. The Myrmic module is implemented in C and features minor changes necessary to interact with Kad and Main; its primary function is to validate nCert signatures and freshness.

Tor. The Tor module remains completely out of box, with only minor changes to the default configurations in the Tor setup file (allow for ControlPort, etc). Furthermore, since the Tor codebase remains untouched, we are certain that no regression in Tor performance, functionality, or security occurred during implementation. The Main script controls circuit extension and node discovery.

Neighborhood Authority. The NA deviates from the out-of-the-box Myrmic protocol to handle the changes in nCert format required by Torsk. It also implements the optimization described in section 4, maintaining a list of the current Torsk membership and periodically checking for liveness.

**Experiments** To test our performance, we deployed our Torsk implementation on 336 PlanetLab nodes, using a private Tor network but connected to the public Kad network. We ran the NA on a server in our lab. Here we report on the performance of each component.

Random Walks. Each of the 336 nodes performed at least 100 random walks of length 16. Figure 5(a) presents their cumulative distribution; average latency per hop was 0.2 sec.

Kad Search. Every Torsk node performed 2200 Kad lookups, including bootstrapping messages, circuit building, and cover traffic, according to the modified Kad lookup algorithm described in Section 4. Figure 5(b) presents the overall performance of this protocol from Main's perspective. The average time to complete a lookup, up through finding the root's nCert, was 2.001 seconds.

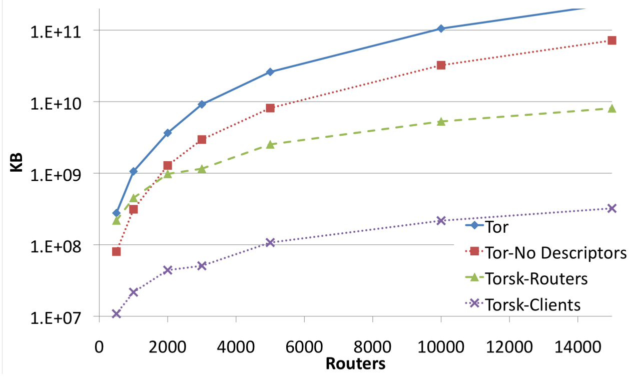

**Figure 6: Total KB after 72 hours of simulation.**

Circuit Building. Extending a circuit requires the successful exchange of a series of messages: (1) DearUltimate messages from the Client requesting the end of the circuit to lookup a key; (2) DearBuddy messages from the endpoint to its buddy; (3) Kad Lookup of the key; (4) (Tunneled) Descriptor Request of the node chosen from the resulting nCert. Once the descriptor has been received, the circuit can be extended by one hop. Thus the time to build a circuit is dominated by the time to complete two DearUltimate requests. These steps are shown in Figure 3, step 2.

In our run, 40 nodes constructed a total of 324 circuits of length 3. To simulate the existence of an entry guard, our circuit construction only involved lookups for the last two hops. Figures 5(c) and (d) depict our overall circuit building performance in PlanetLab; the average time overall was 39.914 seconds. The circuit building overhead is further broken down into the overhead of individual message types, including DearUltimate and DearBuddy, with average durations of 13.2453 and 10.949 seconds, respectively, and finally the nCert verification time averaged 1.088 seconds. We note that while the median circuit building time is significantly higher than reported Tor circuit latencies, this has no effect on stream latency in most cases due to proactive circuit construction.

**Simulation Results.** In order to evaluate the scalability of Torsk we designed simulators for the relay selection and circuit-building bandwidth of both Tor and Torsk. For comparison, we also simulated Tor without router descriptors. The various message sizes were measured empirically. The simulations also include churn of both clients and routers; the churn model for routers was derived empirically from a 1000-hour survey of the Tor network, while the churn model for clients was estimated using arrival rates measured by [19] and session times measured by [11]. The resulting networks had a client to router ratio of approximately 60 : 1. We simulated both protocols at network sizes ranging from 500 to 15000 simultaneous routers. The results of these simulations are shown in the Figures 6 and 7. The maximum observed rate of router churn with 15000 simultaneous routers (900K simultaneous users) was 2064

| Scheme | Disc BW | Discovery Vulnerabilities |
|---|---|---|
| Tor | $O(nr)$ | — |
| I2P | $O(n \log r)$ | route capture, passive logging |
| Salsa | $O(n \log r)$ | Selective DoS, passive logging |
| AP3 | $O(n \log n)$ | passive logging |
| Cashmere | $O(n \log n)$ | passive logging |
| Crowds | $O(n^2)$ | — |
| MorphMix | $O(n)$ | route capture, fingerprinting |
| Tarzan | $O(n^2)$ | — |

**Table 2: Comparison of P2P Anonymity Schemes**

joins and leaves per hour; the Myrmic implementation in [38] was reported to handle over 100 joins and leaves per second.

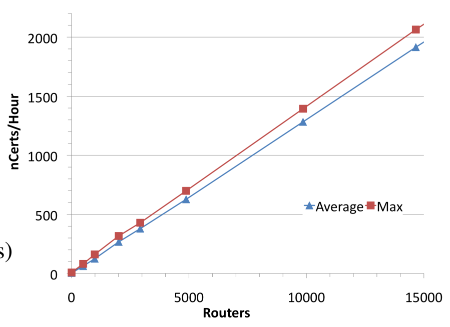

**Figure 7: Membership churn events per hour vs Torsk network size.**

In terms of bandwidth, these results show that for all network sizes simulated, client bandwidth in Torsk is lower than in Tor (where clients have essentially the same bandwidth as routers) with or without descriptor downloads. Moreover, when comparing router bandwidth, we see that Torsk and Tor have similar bandwidth levels at 500 routers, and that Torsk outperforms Tor when there are 1000 routers. As mentioned previously, a constant fraction of the bandwidth cost in Tor is due to the need to periodically download router descriptors; if this cost is ignored, then Tor routers expend less bandwidth than Torsk routers for networks of less than 1000 routers; the costs are comparable for networks of size 1000-2000 routers, and above 2000 routers the bandwidth costs exceed the costs for Torsk. Finally, we note that these cutoff points are naturally sensitive to several Tor parameters. In particular, circuit building is more expensive in Torsk so increasingly frequent circuit construction will improve the relative performance of Tor; likewise, the most expensive operation in Tor is network status distribution, and increasing the frequency of this operation would improve the relative performance of Torsk.

## 7. RELATED WORK

Table 2 compares several closely-related schemes to Torsk. Perhaps the most directly related low-latency anonymity schemes to Torsk are Salsa [24] and I2P [17]. Similar to Torsk, Salsa proposes to build a low-latency anonymous network using Tor's encryption and forwarding routines, but replacing the centralized lookup mechanism by a decentralized scheme. Salsa nodes form a DHT structure very similar to Chord, and node discovery is accomplished through a constant number of redundant, recursive lookups performed by tunnel endpoints. However, Borisov *et al.* [5] show that Salsa's lookup scheme is highly vulnerable to selective DoS. I2P uses its own layered encryption relay scheme that is similar to Tor's, and replaces the centralized directory server with a distributed directory stored on an independent Kad network. Unfortunately, unmodified Kad is vulnerable to misrouting [39] and this leads to relatively straightforward route capture attacks.

Several schemes have proposed the use of a recursive DHT query as an aggregate "hop" along a circuit. In AP3 [21], the client picks a random key and forwards a request through the $O(\log n)$ hops to the root of that key; the recipient then flips a coin to decide whether to service the request or forward it to another random key. An adversary who controls a constant fraction of the network will observe, and thus have the opportunity to drop or record, any request with high probability, leaving the scheme vulnerable to DoS. Cashmere [40] works similarly, but forms "groups" of nodes near a given ID and allows any node in the group to forward a message to the next relay, in order to provide better protection against node churn and adversarial dropping; we note that both schemes have latency that increases logarithmically with the number of peers and are vulnerable to passive logging attacks.

Several other P2P schemes use less structured lookups and routing, such as MorphMix [28], Tarzan [15], and Crowds [27]. MorphMix peers can build circuits knowing as few as 2 other peers; each hop in a tunnel is discovered by asking the current endpoint to "suggest" a next hop, and a "witness" is used to detect attempts at collusion. Tabriz and Borisov [36] demonstrate that as few as 14 peers can collude to deceive this mechanism and capture arbitrary routes passing through an attacker node. Tarzan, on the other hand, employs a gossip discovery protocol, in which nodes regularly gossip about other nodes, with the goal of allowing each node to know the membership of the entire network.[7] Crowds clients connect, via a mutually authenticated channel, to a server called a "blender," essentially a directory service, to download the complete list of participating peers. Both of these schemes thus exhibit quadratic behavior similar to Tor. Finally, Danezis and Syverson [9] propose partitioning into networks of fixed size; this requires a scalable partitioning mechanisms and reduces the anonymity set.

## 8. CONCLUSION

In this paper we introduced Torsk, a secure, scalable, and incrementally deployable node discovery and circuit construction mechanism for Tor. The use of a DHT helps our scheme achieve scalability, while a combination of root verification, buddies and cover traffic ensure that no additional vulnerabilities are introduced. Significantly reducing the client bandwidth requirements also provides opportunities for bandwidth-restricted users to access Tor.

However, we note that several important practical issues remain to be addressed. Torsk circuit-building exhibits high latency due both to the delays imposed by cover traffic and to the additional bandwidth (roughly 2KB) and rounds of communication necessary to perform a DHT search and download router descriptors on demand; while in most cases this latency can be hidden by proactively constructing circuits there are some cases in which the user experience will be impacted. The most important of these cases include hidden services and circuit failures due to unexpected OR churn. A second concern for future work is the transition from a single trusted NA to Tor's 6-server threshold trust model.

**Acknowledgments.** We thank our shepherd, Roger Dingledine, and several anonymous reviewers for their helpful comments on earlier drafts of this paper. We thank Eric Chan-Tin, Eugene Vasserman, and Hun-Jeong Kang for their assistance with PlanetLab experiments, and James Tyra for his assistance with the Kad module. This research was supported by the NSF under grants CNS-0546162, CNS-0716025, and CNS-0917154.

## 9. REFERENCES

[1] The pirate bay. `http://thepiratebay.org/`, November 10 2008.
[2] AN.ON: Anonymity online.
[3] BAGCHI, A., BHARGAVA, A., CHAUDHARY, A., EPPSTEIN, D., AND SCHEIDELER, C. The effect of faults on network expansion. *Theor. Comp. Sys. 39*, 6 (2006), 903–928.
[4] BERTHOLD, O., FEDERRATH, H., AND KØPSELL, S. Web MIXes: A system for anonymous and unobservable

[7]As pointed out by Danezis and Clayton [8], a preliminary implementation of Tarzan used a DHT for node discovery in a way that enabled route fingerprinting attacks.

Internet access. In *Designing Privacy Enhancing Technologies, volume 2009 of LNCS* (July 2000), H. Federrath, Ed., Springer-Verlag, pp. 115–129.

[5] BORISOV, N., DANEZIS, G., MITTAL, P., AND TABRIZ, P. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *CCS '07: Proceedings of the 14th ACM conference on Computer and Communications Security* (October 2007).

[6] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., AND WALLACH, D. Security for structured peer-to-peer overlay networks. In *Proc. of the Fifth Symposium on Operating System Design and Implementation (OSDI)* (2002).

[7] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (1981), 84–88.

[8] DANEZIS, G., AND CLAYTON, R. Route fingerprinting in anonymous communications. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 69–72.

[9] DANEZIS, G., AND SYVERSON, P. Bridging and fingerprinting: Epistemic attacks on route selection. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)* (Leuven, Belgium, July 2008), N. Borisov and I. Goldberg, Eds., Springer, pp. 133–150.

[10] eDonkey network. `http://www.edonkey2000.com`.

[11] ERMAN, D., ILIE, D., AND POPESCU, A. Bittorrent session characteristics and models. In *Proc. of HET-NETs 05 - 3rd International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks* (2005).

[12] FEAMSTER, N., AND DINGLEDINE, R. Location diversity in anonymity networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)* (Washington, DC, USA, October 2004).

[13] FEIGE, U., AND OFEK, E. Spectral techniques applied to sparse random graphs. *Random Structures and Algorithms 27*, 2 (July 2005), 251–275.

[14] FIAT, A., SAIA, J., AND YOUNG, M. Making chord robust to byzantine attacks. In *ESA* (2005).

[15] FREEDMAN, M. J., AND MORRIS, R. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security* (New York, NY, USA, 2002), ACM Press, pp. 193–206.

[16] FRIEDMAN, J., KAHN, J., AND SZEMERÉDI, E. On the second eigenvalue of random regular graphs. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing* (New York, NY, USA, 1989), ACM, pp. 587–598.

[17] I2P. Available from `http://66.111.51.110/`.

[18] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An Architecture for Global-Scale Persistent Storage. In *ASPLOS* (2000).

[19] LOESING, K. Measuring the tor network: Evaluation of client requests to the directories. Tech. rep., Tor Project, June 2009. Available online: `https://git.torproject.org/checkout/metrics/master/report/dirreq/directory-requests-2009-06-26.pdf`.

[20] MAYMOUNKOV, P., AND MAZÍERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS* (2001).

[21] MISLOVE, A., OBEROI, G., POST, A., REIS, C., DRUSCHEL, P., AND WALLACH, D. AP3: cooperative, decentralized anonymous communication. *Proceedings of the 11th workshop on ACM SIGOPS European workshop: beyond the PC* (2004).

[22] MITTAL, P., AND BORISOV, N. Information leaks in structured peer-to-peer anonymous communication systems. In *CCS '08: Proceedings of the 15th ACM conference on Computer and Communications Security* (New York, NY, USA, 2008), ACM, pp. 267–278.

[23] MURDOCH, S. J., AND ZIELIŃSKI, P. Sampled traffic analysis by internet-exchange-level adversaries. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)* (Ottawa, Canada, June 2007), N. Borisov and P. Golle, Eds., Springer.

[24] NAMBIAR, A., AND WRIGHT, M. Salsa: a structured approach to large-scale anonymity. *Proceedings of the 13th ACM conference on Computer and communications security* (2006), 17–26.

[25] O'DONNELL, C. W., AND VAIKUNTANATHAN, V. Information leak in the chord lookup protocol. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing* (2004).

[26] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A Scalable Content-Addressable Network. In *SIGCOMM* (2001).

[27] REITER, M., AND RUBIN, A. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security 1*, 1 (June 1998).

[28] RENNHARD, M., AND PLATTNER, B. Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society* (New York, NY, USA, 2002), ACM Press, pp. 91–102.

[29] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware* (2001).

[30] SNADER, R., AND BORISOV, N. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '08* (February 2008), Internet Society.

[31] STEINER, M., EFFELSBERG, W., EN NAJJARY, T., AND BIERSACK, E. W. Load reduction in the KAD peer-to-peer system. In *DBISP2P 2007, 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, September, 24, 2007, Vienna, Austria* (Sep 2007).

[32] STEINER, M., EN NAJJARY, T., AND BIERSACK, E. W. Analyzing peer behavior in KAD. Tech. Rep. EURECOM+2358, Institut Eurecom, France, Oct 2007.

[33] STEINER, M., EN-NAJJARY, T., AND BIERSACK, E. W. A global view of kad. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), ACM, pp. 117–122.

[34] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A peer-to-peer lookup service for internet applications. In *SIGCOMM* (2001).

[35] STUTZBACH, D., AND REJAIE, R. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (2006).

[36] TABRIZ, P., AND BORISOV, N. Breaking the collusion detection mechanism of morphmix. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)* (Cambridge, UK, June 2006), G. Danezis and P. Golle, Eds., Springer.

[37] VARIOUS. Skype. In *Wikipedia, the Free Encyclopedia*. `http://en.wikipedia.org/wiki/Skype`, November 10 2008.

[38] WANG, P., OSIPKOV, I., HOPPER, N., AND KIM, Y. Myrmic: Secure and robust dht routing. Tech. Rep. 2006/20, University of Minnesota DTC Research Report, 2006.

[39] WANG, P., TYRA, J., MALCHOW, T., KIM, Y., HOPPER, N., KUNE, D. F., AND CHAN-TIN, E. Attacking the kad network. In *SecureComm: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks* (September 2008), ACM Press.

[40] ZHUANG, L., ZHOU, F., ZHAO, B. Y., AND ROWSTRON, A. Cashmere: resilient anonymous routing. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (Berkeley, CA, USA, 2005), USENIX Association, pp. 301–314.

## APPENDIX

Since currently there are around one million ($= 2^{20}$) concurrent Kad nodes and roughly $2^{11}$ ORs, it would be difficult to find an OR simply by searching for random nodeIDs. For this reason, in our implementation Torsk nodeIDs have a specific format: $nodeID = r_1||\mathsf{torskID}_1||r_2||\mathsf{torskID}_2$, where $r_1$ and $r_2$ are random strings, and $\mathsf{torskID}_1$ and $\mathsf{torskID}_2$ are strings common to all Tor ORs. Each string is used for different purposes: $r_1$ is used to distribute nodeIDs evenly over the Kad key space. $\mathsf{torskID}_1$ helps clients lookup Tor OR's, and $r_2$ is used to distinguish specific Tor ORs.

In the context of Torsk node IDs, the success of the lookup is dependent on two factors. The first factor is that we want the Torsk nodes to be distributed over all prefixes of the form $r_1||\mathsf{torskID}_1$ with high probability. For a fixed choice of $r_1$, the probability that none of the $2^t$ Torsk nodes chooses that prefix is $(1-1/2^{r_1})^{2^t}$, and by the union bound the probability that there exists such a prefix is at most $2^{r_1} \times (1-1/2^{r_1})^{2^t} \approx 2^{r_1} \times e^{-2^{t-r_1}}$. From this equation we can compute the length $r_1$ required to reach a desired failure probability $\delta$; for example, with the current number of Tor routers $t \approx 11$, so if we want $\delta = 10^{-5}$ we find that $r_1 = 7$ will suffice.

The second factor is that on average there should be at least as many Torsk nodes matching the prefix $r_1||\mathsf{torskID}_1$ as there are Kad nodes, so that Torsk nodes are easy to find. If there are $2^k$ Kad nodes, this translates to the requirement that $k - (\mathsf{torskID}_1 + r_1) \leq t - r_1$. Using $r_1 = 7$, and $\lceil k \rceil = 21$, we find that $\mathsf{torskID}_1 = 10$ will suffice. With these settings, we expect a given prefix to have 16 Torsk nodes and 8-16 regular Kad nodes.

Finally, $\mathsf{torskID}_2$ is Torsk's fixed Tor OR identifier, chosen to make the probability of an accidental choice of a Torsk nodeID negligible – we use 64 bits; and $r_2$ is random filler to pad out to the full 128 bits of a nodeID. The length of $r_2$ is determined by the equation $r_2 = 128 - r_1 - \mathsf{torskID}_1 - \mathsf{torskID}_2$. Using our earlier settings for the lengths of $r_1$, $\mathsf{torskID}_1$ and $\mathsf{torskID}_2$, the length of $r_2$ is 47 bits.