The Onion Name System

Tor-powered Distributed DNS for Tor Hidden Services

Jesse Victors
7449 S. Babcock Blvd.
Wasilla, AK 99623
jvictors@jessevictors.com

Ming Li
Utah State University
Logan, UT 84321
ming.li@usu.edu

ABSTRACT

Tor is a third-generation low-latency onion router that provides its users with online privacy, anonymity, and resistance to traffic analysis. In recent years its userbase, network, and community has grown significantly in response to revelations of international electronic surveillance, and it remains one of the most popular anonymity networks in use today. Tor also provides access to anonymous servers known as hidden services – servers of unknown location and ownership who achieve anonymity through Tor circuits. These hidden services can be accessed through any Tor-enabled web browser but they suffer from usability challenges due to the algorithmic generation of their addresses.

In response to this difficulty, in this work we introduce the Onion Name System (OnioNS), a privacy-enhanced distributed DNS that allows users to reference a hidden service by a meaningful globally-unique domain name chosen by the hidden service operator. We introduce a new distributed self-healing database and construct OnioNS as an optional backwards-compatible plugin for Tor on top of existing hidden service infrastructure. We simplify our design and threat model by embedding OnioNS within the Tor network and provide mechanisms to allow clients to verify the authenticity or non-existence of domain names with minimal networking costs. Our reference implementation demonstrates that OnioNS successfully addresses the major usability issue that has been with Tor hidden services since their introduction in 2004.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

As the prevalence of the Internet and other communication has grown, so too has the development and usage of privacy-enhancing systems. These are protocols that provide privacy by obfuscating the link between a user's identity or location and their communications. Privacy is not achieved in traditional Internet connections because SSL/TLS encryption cannot hide IP and TCP headers, which must be exposed to allow routing between two parties; eavesdroppers can easily break user privacy by monitoring these headers.[**?**] Following a general distrust of unsecured Internet communications and in light of the 2013-current revelations by Edward Snowden of Internet mass-surveillance by the NSA, GCHQ, and other members of the Five Eyes, users have increasingly turned to these tools for their own protection.

Today, most anonymity tools descend from mixnets.[**?**] Mixnets have inspired the development of many varied mixnet-like protocols and have generated significant literature within the field of network security.[**?**] Mixnet descendants can generally be classified into two distinct categories: high-latency and low-latency systems. High-latency networks typically delay traffic packets, increasing their resistance to global adversaries who monitor communication entering and exiting the network. By contrast, low-latency networks do not delay packets and are thus better suited for common Internet activities such as web browsing, instant messaging, or the prompt transmission of email.[**?**] In this work, we detail and introduce new functionality within low-latency protocols.

Onion routing[**?**][**?**] is the most popular low-latency descendant of mixnets in use today. In onion routing, a user selects a set network nodes, typically called *onion routers* and together a *circuit*, and encrypts the message with the public key of each router. Each encryption layer contains the next destination for the message – the last layer contains the message's final destination. As the *cell* containing the message travels through the network, each of these onion routers in turn decrypt their encryption layer, exposing their share of the routing information. The final recipient receives the message from the last router, but is never exposed to the message's source. The sender therefore has privacy because the recipient does not know the sender's location, and the sender has anonymity if no identifiable or distinguishing information is included in their message.

1.1 Tor

Tor[**?**] is a third-generation onion routing system and is the most popular onion router in use today. Tor's threat model assumes that the capabilities of adversaries are limited to traffic analysis attacks; they may observe or manipulate portions of Tor traffic, that they may run onion routers themselves, and that they may compromise a fraction of other existing routers. Tor's design centers around usability and defence against these types of attacks.

Tor assumes a dynamic network topology and introduced a small set of semi-trusted *directory authority* servers to distribute network information and as a form of PKI. Periodically, Tor routers upload digitally-signed *descriptors* – containing routing information, cryptographic keys, bandwidth history, and other information – to these authorities. Once an hour, the directory authorities aggregate, sign, and republish the descriptors as a *network status consensus* document. Clients download essential portions of these descriptors from the directory authorities or from known routers redistributing the consensus, forming *microdescriptors*. These are aggregated into three files, *cached-certs*, containing each directory authority's long-term identity RSA key, medium-term signing key chained to the identity key, and validity timeline for these keys; *cached-microdescs*, containing public RSA and Curve25519[1] keys (used for TAP[2] and NTor[**?**] circuit construction protocols, respectively); and *cached-microdesc-consensus*, containing all microdescriptors. Although Tor provides no guarantee that the entire network has the same consensus at the same time, the consensus can be verified retrospectively, allowing the consensus to be referenced by its time of publication.

1.2 Hidden Services

Tor also supports *hidden services* – anonymous servers that intentionally mask their IP address through Tor circuits and cannot normally be accessed outside the context of Tor. Tor hidden services provide bidirectional anonymity where both parties remain anonymous and never directly communicate with one another. Hidden services are only known by their public key and accessed in any Tor-enabled browser by their 16-character base32-encoded address with the .onion pseudo-TLD; the address is the first 16 bytes of the base32-encoded SHA-1 hash of the server's RSA key. This builds a publicly-confirmable one-to-one relationship between the public key and its address and allows hidden services to be referenced by their address in a distributed environment.

At startup, hidden services
publish their existence by uploading
[**?**][**?**]

Let Bob be a hidden service. At startup, Bob randomly selects several router and builds Tor circuits to them. He then creates a hidden service descriptor, consisting of his public key B_K and a list of these routers. He signs the descriptor and sends a distributed hashtable within the Tor network, enabling the routers he chose to act as his *introduction points*. When Alice, a Tor client, obtains Bob's hidden service address though a backchannel, she queries this hashtable for Bob's address. Once she obtains Bob's hidden service descriptor, she then builds a circuit to one of the introduction points. Simultaneously, Alice also selects and builds a circuit to another relay, R_A. She encrypts R_A and a nonce with B_K and gives the result to R_A. Bob decrypts the message and builds a circuit to R_A (for security reasons, he uses the same guard router) and sends the nonce to Alice. Alice can then confirm Bob's authenticity and the two can begin communication over six Tor nodes: three established by Alice and three by Bob.

As Tor's hidden service addresses are algorithmically derived from the service's public RSA key, there is at best limited capacity to select a human-meaningful name. Some operators use vanity key generators to find by brute-force an RSA key that generates a partially-desirable hidden service address (e.g. "example0uyw6wgve.onion") and although some alternative encoding schemes have been proposed, (section) the problem generally remains. The usability of hidden services is severely challenged by their non-intuitive and unmemorable base58-encoded domain name. For example, 3g2upl4pq6kufc4m.onion is the address for the DuckDuckGo search engine, suw74isz7wqzpmgu.onion is a WikiLeaks mirror, and 33y6fjyhs3phzfjj.onion and vbmwh445kf3fs2v4.onion are both SecureDrop instances for anonymous document submission. These addresses maintain strong privacy as no personally-identifiable information is ever needed from a hidden service operator. However, this privacy comes at a cost: there is a strong disconnect between the address and the service's purpose and it is impossible to classify or label a hidden service's purpose in advance, facts well known within Tor hidden service communities. Over time, third-party directories – both on the Clear and Dark Internets – have appeared in attempt to counteract this issue, but these directories must be constantly maintained and furthermore this approach is neither convenient nor does it scale well. This suggests the strong need for a more complete and reliable solution.

2. CHALLENGES

2.1 Zooko's Triangle

This problem is illustrated in Figure **??** and summarized by Zooko's Triangle, a conjecture proposed by Zooko Wilcox-O'Hearn in 2003. The conjecture states a persistent naming system can achieve at most two of these properties: it can provide unique and meaningful names but not be distributed, it can be distributed and provide unique names that are not meaningful, or it can be distributed and provide meaningful names that are not guaranteed to be unique.[**?**][**?**]

Some examples of naming systems that achieve only two of these properties include:

- **Securely unique and human-meaningful** — Internet domain names are memorable and provably collision free, but the Internet DNS with a hierarchical structure with central authorities under the jurisdiction of ICANN.
- **Decentralized and human-meaningful** — Human names and nicknames are legal and social labels for each other, but we provide no protection against name collisions.
- **Securely unique and decentralized** — Tor hidden service .onion addresses, PGP keys, and Bitcoin/Namecoin addresses use the large key-space and the collision-free properties of cryptographic hash algorithms to ensure uniqueness, but do not use meaningful names.

Petnames are systems that achieve all three properties of Zooko's Triangle.

2.2 Non-existence Verification

In our design requirements we specify that clients of a naming system should be able to verify the authenticity of domain names. On the Internet, this is achieved through SSL certificates: clients first receive from the destination server a digital certificate, they verify that the certificate matches the domain name they requested, and finally they check the certificate against Certificate Authorities via a Chain of Trust. Of equal importance, however, is the capability to verify the non-existence of domain names. Specifically a resolver may falsely claim that a domain name cannot be resolved because it does not exist, but a client has no mechanism by which to verify this claim besides changing resolvers. This is a weakness often overlooked in other DNSs and resolving this problem is not easy. However, we in section **??** we introduce a data structure and a protocol that allows a resolver to proof non-existence in constant time.

3. PROBLEM STATEMENT

3.1 Assumptions and Threat Model

1. Let Alice is a Tor user and let Bob be a recipient. If Alice constructs a three-router Tor circuit via NTor and sends a message m to Bob, we assume that the Tor circuit preserves Alice's privacy and provides her with anonymity from Bob's perspective. Namely, we assume that out of Alice's identity, location, and knowledge of Bob, an outside attacker Eve can obtain at most one, and that Bob can know no more about Alice than the contents of m. We assume that neither Bob nor Eve can force Alice into breaking her privacy involuntarily. The protection of a Tor circuit relies on Tor's assumption that Eve only has access to some Tor traffic; no defence is made by Tor nor by OnioNS to defend against a global adversary.

2. Adversaries cannot break standard cryptographic primitives.

   (a) We assume that they cannot efficiently break AES ciphers, SHA-384 hashes, RSA-1024, Curve25519 ECDHE, Ed25519 digital signatures, or have hardware-level attacks against the scrypt key derivation function.

   (b) We assume that they maintain no backdoors or other software breaks in the Botan or the OpenSSL implementations of the above primitives.

   (c) We assume that they are not capable of breaking cryptographic protocols built on the primitives such as TAP and NTor.

3. We assume that not all Tor nodes are honest. We believe it reasonable to say that at least some Tor nodes are run by malicious operators, curious researchers, experimenting developers, or government organizations. They may also be wiretapped, exploited, broken into, or become unavailable. This assumption is also made by Tor's developers and therefore we must conclude that any new functionality added to existing Tor nodes must also fall under their assumption. However, we assume that the majority of Tor nodes are honest and reliable. We consider this reasonable because it is a prerequisite for the security of Tor circuits.

4. We assume that the percentage of dishonest routers within the Tor network does not increase in response to the inclusion of OnionNS into Tor infrastructure. This assumption simplifies our threat model analysis but we consider it realistic because Tor traffic is purposely kept secret and is therefore much more valuable to Eve. OnioNS uses public data structures so we don't consider the inclusion of OnioNS a motivating factor to Eve. Moreover, we assume that Eve cannot predict in advance the identities of the Tor routers that act as authorities for OnioNS.

5. Let C be the set of Tor nodes that have the Fast and Stable flags and let Q be an M-sized set chosen randomly from C. Q may be under the influence of one or more adversaries who may cause subsets of Q to collude, but our final assumption is that the largest subset of Q is acting honestly according to specifications.

4. DESIGN OBJECTIVES

Tor's high security environment introduces a distinct set of challenges that must be met by additional functionality. Privacy and anonymity are of paramount importance. Here we enumerate a list of requirements for any security-enhanced DNS applicable to Tor hidden services. In Chapter **??** we analyse several existing prominent naming systems and show how these systems do not meet these requirements. In Chapters **??** and **??** we demonstrate how we overcome them with OnioNS.

1. **The system must support anonymous registrations.** It must be hard for any party to infer the identity or location of the registrant, unless the registrant chooses to disclose that information. Hidden services are anonymous servers and the registration of a domain name should not compromise that privacy.

2. **The system must support privacy-enhanced lookups.** The resolver should not be able to identify a client nor track their lookups. In other words, clients should ideally have anonymity and be indistinguishable from other clients from the perspective of a resolver.

3. **Clients must be able to authenticate registrations.** The users of the system must be able to verify that the information they received from the service has not been forged and is authenticate relative to the authenticity of the server they will connect to.

4. **Domain names must be provably or have a near-certain chance of being unique.** Any domain name of global scope must point to at most one server. In the case of naming systems that generate names via cryptographic hashes, the domain name key-space must be of sufficient length to be remain resistant to at least collision and second pre-image attacks.

5. **The system must be distributed.** Systems with root authorities have distinct disadvantages compared to distributed networks: specifically, central authorities have absolute control over the system and root security breaches could easily compromise the integrity of the entire system. Root authorities may also be able to easily compromise user privacy or may not allow anonymous registrations. For these reasons, naming

systems with central authorities can safely be considered dangerous and ill-suited for hidden services.

6. **The system must be simple and relatively easy to use.** It should be assumed that users are not security experts or have technical backgrounds. Tor's developers go to great lengths to ensure that Tor tools are straightforward, and the success of Tor's low-latency onion routing model demonstrates the effectiveness of convenience within high-security settings. Any naming system used by Tor must be equivalently simple, resolve protocols with minimal input from the user, and hide non-essential details.

7. **The system must be backwards compatible with existing protocols.** Just as the Internet's DNS did not introduce any changes in the TCP/IP layer, naming systems for Tor must preserve the original Tor hidden service protocol. DNS is optional, not required.

Additionally, we specify several performance-enhancing objectives, although these are not requirements.

1. **The system should not require clients to download the entire database.** It can safely assumed that in most realistic environments clients have neither the network capacity nor the storage to hold the system's database. While they may choose to download the database, it should not be required for normal functionality or to meet any of the above objectives.

2. **The system should not introduce significant burdens to the clients.** Despite the rapid and exponential growth of consumer-grade hardware, in most environments not every client is on a high-end machine. Therefore the system should not burden user computers with significant computation or memory demands.

3. **The system should have low latency** and resolve lookup queries within a reasonable amount of time.

5. EXISTING WORKS

In Section **??** we listed seven design requirements that ideally must be met by a naming system in order to be applicable to Tor hidden services. We now examine several workarounds and existing naming systems against these objectives.

5.1 Address Manipulation

Although they are not separate naming systems in their own right, several systems have been proposed to directly improve the readability of hidden service addresses. These include vanity key generators and different encoding schemes.

Shallot is a vanity key generator which uses OpenSSL to generate by brute-force many RSA keys in attempt to find one that has a desirable hash.[**?**] For example, a hidden service operator may wish to start his service's address with a meaningful noun so that others may more easily recognize it. Shallot has been used successfully for both common and high-profile hidden services, such as blockchainbdgpzk.onion, an official mirror of Blockchain.info; facebookcorewwwi.onion, Facebook's hidden services; and freepress3xxs3hksomion, the Freedom of the Press's SecureDrop instance. However, Shallot is only partially successful at enhancing readability because the size of the domain key-space is too large to be fully brute-forced in any reasonable length of time.[**?**] This situation is expected to get worse over time as Tor plans to increase the length of hidden service addresses.[**?**] If the address key-space was reduced to allow a full brute-force, the system would fail to be guaranteed collision-free.

Nicolussi suggested changing the address encoding from base58 to a delimited series of words, using a dictionary known in advance by all parties.[**?**] We consider this approach very similar to vanity key generation. Like Shallot, Nicolussi's encoding partially improves the recognition and readability of an address but does nothing to alleviate the logistic problems of manually entering in the address into the Tor Browser. The key-space is not changed and is again too large for hidden service operators to select many meaningful words.

These proposals are cosmetic and do not significantly change Tor's hidden service infrastructure and its intersection with our design requirements; they still support anonymous registrations, privacy-enhanced lookups, publicly verifiable registrations, collision-free addresses, and are distributed by nature. However, they only meet one side of Zooko's Triangle and thus fail our usability requirements; although these are partial solutions, they do not achieve a full correlation between an address and the service's purpose.

5.2 Centralized or Zone-Based DNS

5.2.1 Internet DNS

The Internet's Domain Name System was designed in 1983 to map Internet domain names to IP addresses. A domain name consists of a series of labels, delimited by dots. Each label subdivides the system into hierarchical zones, where the root zone that contains TLDs (top-level domains such as .com, .org, or .gov) is managed by IANA and NTIA. Each label can be up to 63 characters in length with the entire domain name up to 253 characters in length. The Internet DNS is a critical component to the usability of the Internet as it abstracts away IP addresses, allowing servers to be referenced by an easily-memorized human-meaningful name.

Despite its widespread use and extreme popularity, the Internet DNS suffers from several significant shortcomings and security issues that make it inappropriate for use by Tor hidden services. With the exception of extensions such as DNSSEC, the Internet DNS by default does not use any cryptographic primitives. However, as DNSSEC is primarily designed to prevent forgeries and DNS cache poisoning from intermediary DNS resolvers, it preserves the hierarchical structure and does not provide any degree of query privacy.[**?**] Additional extensions and protocols such as DNSCurve[**?**] have been proposed, but DNSSEC and DNSCurve are optional and have not yet seen widespread full deployment across the Internet. Traditional DNS lookups may be intercepted and modified by MITM attacks, user privacy may be compromised by wiretapping DNS lookups, and the system is by default vulnerable to DNS cache poisoning.

The lack of default security in Internet DNS and the financial expenses involved with registering a new TLD casts significant doubt on the feasibility of using for Tor hidden services. Furthermore the system meets only a few of our design requirements: although the system is easy to use, the system is hierarchical but not truly distributed, domain

registrars typically require the owner to reveal significant amounts of identifiable information, registrations are not confirmable except through expensive SSL certificates issues by a central authority, and lookups occur by default without any privacy enhancements. These issues make the Internet DNS ill-suited for Tor hidden services.

5.2.2 GNU Name System

The GNU Name System[?] (GNS) is a decentralized privacy-enhanced replacement for the Internet DNS. GNU describes a hierarchical zones of with each user managing their own zone and distributing zone access peer-to-peer within social circles without any central authority. Every GNS use (let Alice be one such user) chooses a nickname for themselves which they announce to the network via a distributed hashtable. A peer Bob selects a name for Alice, which is typically Alice's nickname unless there is a collision. Then Bob adds Alice's zone into his zone such that he delegates all requests for her nickname to Alice. In this way, Bob acts as a resolver to Alice and Bob's name for Alice (or her nickname for herself) is guaranteed unique within Bob's zone. Bob's peers can likewise add Bob and Alice's zone to their zone and delegate requests for them. Assuming each user has selected a unique name for themselves, (which they are incentivized to do) there will be no collusions within each zone. GNS uses the .gnu pseudo-TLD.

While GNS' design guarantees the uniqueness of names within each zone and users are capable of selecting meaningful nicknames for themselves, GNU does not guarantee that names are *globally* unique. The selection of a trustworthy zone to use would be a significant challenge for using GNS for Tor hidden services. The user maintaining that zone is a central point of failure and could severerly disrupt the system if they acted maliciously. However, GNS does support anonymous registrations, privacy-enhanced queries, authenticated domain names, simplicity, and backwards compatibility. For these reasons, we consider GNS a very impressive system and recommend GNS as a possible fallback from OnioNS.

5.3 Distributed Systems

Several systems have been developed that aim to serve names in a decentralized environment. Cachin and Samar[?] extended the Internet DNS and decreased the attack potential for authoritative name servers via threshold cryptography, but the lack of privacy in the Internet DNS and the logistical difficulty in globally implementing their work prevents us from using their system for hidden services. Awerbuch and Scheideler,[?] constructed a distributed peer-to-peer naming system, but like GNS, made no guarantee that domain names would be globally unique. More recently, Zooko's Triangle was proven false by practical demonstration by the development of Namecoin, a cryptocurrency discussed below.

5.3.1 Namecoin

Namecoin is a decentralized information registration and transfer system, developed pseudo-anonymously in early 2011. It was the first fork of Bitcoin[?] and inherits most of Bitcoin's design and capabilities. Namecoin holds digitally-signed information transactions in a data structure known as a block; each block links to a previous block, forming a public ledger known as a blockchain. Each participant in the Namecoin network holds a copy of the blockchain and listen for and broadcast new blocks peer-to-peer. New Namecoins are generated at a fixed rate in a process known as mining, which is the computationally-expensive generation (due to proof-of-work) of a new block. Namecoins are then transferred peer-to-peer, and the registration of a domain name has some cost as it consumes a quantity of Namecoins. Domain names use the .bit pseudo-TLD and expire approximately every 250 days and so must be renewed periodically to preserve exclusive ownership. In 2014, Namecoin was recognized by ICANN as the most well-known example of a PKI and DNS system with an emphasis of distributed control and privacy.[?]

As previously stated, Namecoin is noteworthy for being the first naming system to demonstratively achieve all three properties of Zooko's Triangle. It supports privacy-enhanced lookups as anyone can obtain a copy of the blockchain and perform queries against it, names are guaranteed to be unique as each participant (and thus the network as a whole) will reject new blocks that contain names already contained in their blockchain, and it is distributed by nature. While Namecoin is often advertised as capable of assigning names to Tor hidden services, it has several practical issues that make it generally infeasible to be used for that purpose and it fails several of our design requirements. First, to authenticate registrations, clients must be able to prove the relationship between a Namecoin owner's secp256k1 ECDSA key and the target hidden service's RSA key: constructing this relationship is non-trivial. Second, Namecoin generally requires users to download the blockchain before use which introduces logistical issues; as of April 2015 Namecoin's blockchain is 2.45 GB.[?] Third, although ownership and transfer of Namecoins leaks no more than an ECDSA key, obtaining the Namecoins in the first place (and thus being able to claim a domain name) anonymously is non-trivial due to the logging in the blockchain. Finally, unlike the aforementioned naming systems, Namecoin is not based on published works and literature on it is sparse. Although we also reject Namecoin as a possible naming system for Tor hidden services, OnioNS' design shares some design principles with Namecoin.

6. SOLUTION

6.1 Overview

We propose the Onion Name System (OnioNS) a Tor-powered distributed naming system that maps .tor domain names to .onion addresses. The system has three main aspects: the generation of self-signed claims on domain names by hidden service operators, the processing of domain information within the OnioNS servers, and the receiving and authentication of domain names by a Tor client.

First, a hidden service operator, Bob, generates an association claim between a meaningful domain name and a .onion address. Without loss of generality, let this be example.tor $\rightarrow$ example0uyw6wgve.onion. For security reasons we do not introduce a central repository and authority from which Bob can purchase domain names; however, domains must not be trivially obtainable and Bob must expend effort to claim and maintain ownership of example.tor. We achieve this through a proof-of-work scheme. Proof-of-work systems are noteworthy for their asymmetry: they require the issuer to spend effort to find an answer to a moderately hard com-

putational problem, but once solved can be easily verified correct by any recipient. The requirement of proof-of-work fulfils three main purposes:

1. Significantly reduces the threat of record flooding.
2. Introduces a barrier-of-entry that encourages the utilization of domain names and the availability of the underlying hidden services.
3. Increases the difficulty of domain squatting, a denial-of-service attack where a third-party claims one or more valuable domain names for the purpose of denying or selling them en masse to others. In Tor's anonymous environment, this vector is particularly prone to Sybil attacks.

Bob then digitally signs his association and the proof-of-work with his service's private key.

Second, Bob uses a Tor circuit to anonymously transmit his association, proof-of-work, digital signature, and public key to OnioNS nodes, a subset of Tor routers. This set is deterministically derived from the volatile Tor consensus documents and is rotated periodically. The OnioNS nodes receive Bob's information, distribute it amongst themselves, and later archive it in a sequential transaction database, of which each OnioNS node holds their own copy. The nodes also digitally sign this database and distribute their signatures to the other nodes. Thus the OnioNS Tor nodes maintain a common database and have each vouched for its authenticity.

Third, a Tor client, Alice, uses a Tor client to anonymously connect to one of these OnioNS nodes and request a domain name. Without loss of generality, let this request be example.tor. Alice receives Bob's "example.tor → example0uyw6wgve.onion" association, his proof-of-work, his digital signature, and his public key. Alice then verifies the proof-of-work and Bob's signature against his public key, and hashes his key to confirm the accuracy of example0uyw6wgve.onion. Alice then looks up example0uyw6wgve.onion in Tor's distributed hashtable, finds Bob's introduction point, and confirms that its knowledge of Bob's public key matches the key she received from the OnioNS nodes. Finally, she finishes the Tor hidden service protocol and begins communication with Bob. In this way, Alice can contact Bob through his chosen domain name without resorting to use lower-level hidden service addresses. The uniqueness and authenticity of the Bob's domain name is maintained by the subset of Tor nodes.

7. SECURITY ANALYSIS

Now we examine and compare OnioNS' central protocols against our security assumptions and expected threat model.

7.1 Quorum Selection

The Quorum nodes have greater attack capabilities than any other class of participants in OnioNS. They have active responsibility over the front of the Pagechain and they must receive, flood, and process new Records from hidden service operators. In our threat model, we assume that an attacker, Eve, already has control of some fixed number f_E of routers in the Tor network, and that her nodes may maliciously collude. We also assume that Eve does does not have motivation to compromise Tor in response to the presence of OnioNS and that she cannot predict future Quorums. It is also impossible to determine which Tor routers are under Eve's control and which are honest in advance, so we examine our Quorum protocols and explore the likelihood of attacks within a probabilistic environment.

The Quorum Derivation protocol selects an L_Q-sized subset of routers from the set of Quorum Candidates, and rotates this selection every Δq days. The optimal selection of L_Q and Δq is dependent on both security and performance analysis; our security analysis introduces a lower bound on both L_Q and Δq. For the following evaluations, we feel it safe to discard threats that have probabilities at or below $\frac{1}{2^{128}} \approx 10^{-38.532}$ — the probability of Eve randomly guessing a 128-bit AES key, a threat that would violate our security assumptions.

Our security analysis assumes that L_Q will be selected from a pool of 5,400 Quorum Candidates — the number, as of April 2015, of Tor routers with the Fast and Stable flags, whom we assume are all up-to-date Mirrors. Let L_E be the number of Quorum nodes under Eve's control. Then Eve controls the Quorum if the L_E routers become the largest agreeing subset in the Quorum, which can occur if either more than $\frac{L_Q - L_E}{2}$ honest Quorum nodes disagree or if $L_E > \frac{L_Q}{2}$. The second scenario can be statistically modelled.

Quorum selection is mathematically an L_Q-sized random sample taken from an N-sized population without replacement, where the population contains a subset of f_E entities that are considered special. Then the probability that Eve controls k Tor routers in the Quorum is given by the hypergeometric distribution, whose probability mass function (PMF) is $\frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}}$. Then the probability that $L_E > \frac{L_Q}{2}$ is given by $\sum_{x=\lceil \frac{L_Q}{2} \rceil}^{L_Q} \frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}}$. Odd choices for L_Q prevents the possibility of network disruption when the Quorum is evenly split in terms of the current Page. We examine the probability of Eve's success for increasing amounts of f_E in Figure 1.

Figure 1 shows that a choice of $L_Q = 31$ is suboptimal: the probabilities are above the $10^{-38.532}$ threshold for even small levels of collusion. $L_Q = 63$ likewise fails with approximately two percent collusion, although choices of 127, 255, and 511 fail at levels above approximately 8, 16, and 25 percent, respectively. The figure also suggests that larger Quorums are superior with respect to security. Small Quorums are also less resilient to DDOS attacks at the Quorum in general.

If we assume that Eve controls 10 percent of the Tor network, then we can examine the impact of the longevities of Quorums; over a fixed period of time, slower rotations suggests a lower cumulative chance of selecting any malicious Quorum. If w is Eve's chance of compromise, then her cumulative chances of compromising any Quorum is given by $1 - (1 - 2)^t$. This gives us a bound estimate on Δq. We estimate this over 10 years in Figure 2.

Figure 2 suggests that while slow rotations (i.e a period of 7 days) generates orders of magnitude less chance than fast rotations, the choice of L_Q is far more significant. Like Figure 1, it also shows that $L_Q = 31$ and $L_Q = 61$ are relatively poor choices.

If a selected Quorum is malicious, fast rotation rates will minimize the duration of any disruptions, as shown in Figure

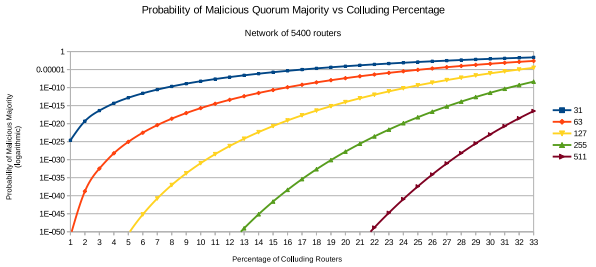

Figure 1: The probability that Eve controls the majority of the Quorum is given by the PMF of the hypergeometric distribution. We fix N at 5,400 nodes and graph Eve's success probability as a function of an increasing percentage of Eve-controlled colluding routers. We examine five selections for L_Q: 31, 63, 127, 255, and 511. We do not consider percentages beyond 33 percent as 33 percent represents a complete compromise of the Tor network: it is near 100 percent that the three routers selected during circuit construction are under Eve's control, a violation of our security assumptions.

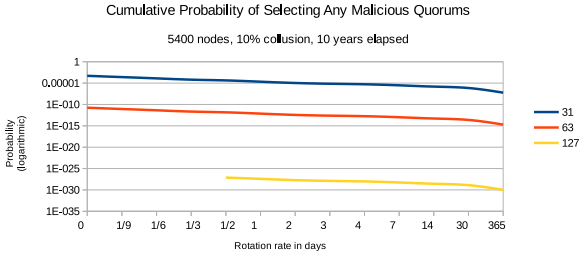

Figure 2: The cumulative probability that Eve controls any Quorum at different rotation rates. We assume 10 percent collusion in a network of 5400 Tor routers, and view across 10 years. We do not graph L_Q values of 255 or 511 as they generate probabilities far below our $10^{-38.532}$ threshold; $L_Q = 255$ and $L_Q = 511$ produce values less than 10^{-58} and 10^{-134}, respectively.

3. This figure suggests that fast rotations are optimal in that respect, a contradiction to Figure 1. However, given the very low statistical likelihood of selecting a malicious Quorum, we consider this a minor contribution to the decision.

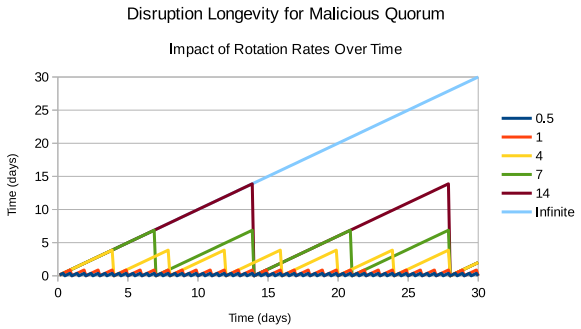

Figure 3: The duration of malicious Quorums as a function of different rotation rates. Quorums that have very short livetimes (and are thus rotated quickly) minimize the duration of any malicious activity.

Although a malicious Quorum would have the capabilities to deploy a variety of attacks on the network, the proper selections of $L_Q \geq 127$ and $\Delta q \geq 1$ reduces the likelihood of this occurring to near-zero probabilities. We consider this a stronger solution than introducing countermeasures to those attacks. Based on our security analysis, we suggest $L_Q \geq 127$ and $\Delta q \geq 1$. However, the networking and performance load scales linearly with Quorum size. Based on this balance and our above analysis, we suggest 127 or 255 for values of L_Q and 7 or 14 for Δq.

7.2 Entropy of Tor Consensus Documents

We use Tor's consensus documents as a sources of entropy agreed upon by all parties, however we have not yet demonstrated that the network status contains enough entropy to provide reasonable assurance that Eve cannot guess the next Quorum in advance. If Eve could predict future Quorums, Eve can subvert the Quorum Derivation protocol in a variety of attack vectors. However, this would fail our security assumption against adaptive compromise in the presence of OnioNS. Rather than introducing defences against these attack vectors, we nevertheless believe that ensuring sufficient entropy in the consensus documents is a superior defence.

In section **??** we detailed the significant contents of the three consensus documents relevant to OnioNS, *cached-certs*, *cached-microdesc-consensus*, and *cached-microdescs*. As we stated in section **??**, the Quorum Derivation protocol only utilizes the *cached-certs* and *cached-microdesc-consensus* documents for reasons we discuss below.

7.2.1 cached-certs

The *cached-certs* document contains long-term directory identity keys and medium-term signing keys. While Eve cannot predict the public half of new signing keys in advance, the keys rotate every 3-12 months[**?**] so *cached-certs* is not a timely source of entropy. Nevertheless, if an attacker can predict the network status described in *cached-microdesc-consensus*, the rotation timeline of the signing keys in *cached-certs* places an absolute upper-bound on the duration of Quorum predictability.

7.2.2 cached-microdesc-consensus

The *cached-microdesc-consensus* document describes the network status and is our main source of entropy. In the header, the *vote-digest* header is the hash of a directory authority's status vote. Each directory operates independently and there are nine directory authorities, so we consider this value unpredictable. Router descriptors follow the header in the body of the document.

The r field in a router's descriptor contains routing information and time of last restart. Tor routers have no guarantee of availability and routers may restart for a variety of reasons. As of April 2015 Tor's network consists of approximately 7,000 routers [**?**] and assuming that a given router restarts every 60 days, statistically the number of unpredictable r fields each day is given by $\frac{7000}{60} \approx 116.66$. Tor displays the restart time down to the second, so each restart adds approximately six bytes of entropy, for an estimated total of $\frac{7000*6}{60} = 700$ bytes of short-term entropy from the r field.

The v field describes the version of Tor that the router is running. Although the versions of Tor are publicly known and the official and unofficial Linux repositories are publicly accessible, Eve cannot predict when an administrator will upgrade their router to the next available Tor version. Although new versions of Tor are not frequently released and routers are upgraded infrequently as shown in Figure **??**, the v field still introduces a degree of medium-term entropy into the document.

The w field contains the router's estimated bandwidth capacity as calculated by bandwidth authorities. Clients use this field during circuit construction; routers with a higher bandwidth capacity relative to the rest of the network have a higher probability of being included in a circuit. Although it is likely that a given router will have similar bandwidth measurements between consecutive consensus documents, Eve cannot predict the exact performance of a router from the perspective of the bandwidth authority. Eve's capacity to predict the performance of all routers falls outside of Tor's and OnioNS' security assumptions; Eve must therefore be a global attacker who would be capable of compromising the Tor network as a whole anyway.

We note that significant amounts of additional entropy could be trivially added into *cached-microdesc-consensus* if each router added a single random byte to their descriptor or if the directory authorities each contributed entropy.

7.2.3 cached-microdescs

Although we did not detail it in section [**?**], in practice Tor places client-side timestamps inside *cached-microdescs*. These timestamps would significantly divide the network, causing *cached-microdescs* to be ill-suited for inclusion in the Quorum Derivation protocol. Although *cached-microdescs* contains long-term router keys and the fingerprints of routers in the same family which would both serve as a source of long-term entropy, *cached-microdesc-consensus* contains the SHA-256 hashes of the router descriptor. We therefore do not include it when hashing the documents in the Quorum Derivation protocol.

7.3 Malicious Entropy Reduction

The Quorum Derivation protocol describes initializing the Mersenne Twister with a 384-bit seed. If we assume that Eve desires that the Quorum Derivation protocol produce a Quorum pleasing to Eve (such as including her malicious routers in the Quorum or rejecting specific honest routers from the Quorum) then Eve can find k seeds that generates a desirable scrambled list in 2^{192} operations on average, or 2^{384} operations in the worst case. The chance of any of those seeds being selected is $\frac{2^{384}}{k}$.

If we also assume that Eve can predict some fraction $f \in (0, 1)$ of the contents of the consensus documents, then Eve may attempt to manipulate her router's descriptors such that the Quorum Derivation protocol produces one of the k hashes. SHA-384's strong resistance to preimage and second preimage attacks requires 2^{192} operations on average for Eve to find one of the k hashes. This difficulty is compounded by the limited combinatorics within the set of valid router descriptors. The number of operations involved in this attack vector is significantly more than the operations involved in breaking AES, so we disregard the possibility of manipulating the Quorum Derivation protocol in this way. We do not consider the possibility of Eve controlling the entire consensus document, this would severely compromise the integrity of the Tor network and thus violates our security assumptions.

7.4 Sybil Attacks

Eve may also attempt to increase her probability of including her malicious nodes in the Quorum via Sybil attacks. We offer no defence against this type of attack, although Tor does. The attack is difficult to carry out in practice due to the slow build of trust within the Tor network. Directory authorities would give Eve's nodes the Fast and Stable flags after weeks of continual uptime and a history of reliability. For large-scale Sybil attacks, this introduces a significant time and financial cost to Eve. We also note that choices of

L_Q and Δq also offer significant statistical defences against Sybil attacks, as illustrated in Figures 1 and 2, shown above.

7.5 Hidden Service Spoofing

OnioNS does not require a hidden service operator to reveal any personally-identifiable information. Hidden services are only known by their public key and domain names, and we assume that the hidden service is authentic. We have also observed spoofed hidden services in the wild, suggesting that this problem already exists in Tor's environment. We do not introduce a reputation system or distributed verification system, and note that it is difficult if not impossible to construct a reliable defence against hidden service spoofing attacks due to their anonymous nature.

One possible solution, although it severely compromises anonymity, is to register a hidden service with an SSL Certificate Authority and apply an SSL certificate to the server. In this way, TLS communication provides an authenticity check against the hidden service, although TLS also sets up a redundant encryption layer that may decrease performance. However, in practice this solution is very rarely seen; out of approximately 25,000 hidden services,[**?**][**?**] to date only three hidden services have browser-trusted SSL certificates: Blockchain.info at https://blockchainbdgpzk.onion, Facebook at https://facebookcorewwwi.onion, and The Intercept's SecureDrop instance at https://y6xjgkgwj47us5ca.onion. In future work we may study the security implications of signing the .onion address or the .tor OnioNS domain name. Due to their severe privacy concerns and the security controversy surrounding the centralized CA Chain of Trust, generally speaking we do not recommend the application of SSL certificates to Tor hidden services.

7.6 Outsourcing Record Proof-of-Work

The Record Generation protocol can safely take place within an offline machine under the operator's control. We designed the Record Generation protocol with the objective of requiring the hidden service operator to also perform the scrypt proof-of-work. However, our protocol does not entirely prevent the operator from outsourcing the computation to secondary resource in all cases.

Let Bob be the hidden service operator, and let Craig be a secondary computational resource. We assume that Craig does not have Bob's private key. Then,

1. Bob creates an initial Record R and completes the *type*, *nameList*, *contact*, *timestamp*, and *consensusHash* fields.
2. Bob sends R to Craig.
3. Let *central* be *type* $\|$ *nameList* $\|$ *contact* $\|$ *timestamp* $\|$ *consensusHash* $\|$ *nonce*.
4. Craig generates a random integer K and then for each iteration j from 0 to K,
   (a) Craig increments *nonce*.
   (b) Craig sets PoW as PoW(*central*).
   (c) Craig saves the new R as C_j.
5. Craig sends all $C_{0 \le j \le K}$ to Bob.
6. For each Record $C_{0 \le j \le K}$ Bob computes
   (a) Bob sets *pubHSKey* to his public RSA key.
   (b) Bob sets *recordSig* to $S_d(m, r)$ where $m = central \| pow$ and r is Bob's private RSA key.
   (c) Bob has found a valid record if $H(central \| pow \| recordSig) \le 2^{difficulty * count}$

Our protocol ensures that Craig must always compute more scrypt iterations than necessary; Craig cannot generate *recordSig* and thus cannot compute if the hash is below the threshold. Moreover, the scrypt work incurs a cost onto Craig that must be compensated financially by Bob. Thus the Record Generation protocol places a lower bound on the cost paid by Bob.

7.7 DNS Leakage

Human mistakes can also compromise user privacy. One such security threat is the accidental leakage of .tor lookups over the Internet DNS. This vulnerability is not limited to OnioNS and applies to any alternative DNS; users may mistakenly attempt lookups over the traditional Internet DNS. Mohaisen and Thomas observed .onion lookups on root DNS servers at a frequency that corresponded to external global events, highlighting the human factor in those leakages.[**?**] For OnioNS, this may occur if their client software was not properly configured, if their browser was not properly configured to or could not communicate with Tor, or for other reasons. We offer no defence against this attack vector and note that any defence against it would need to introduce lookup whitelists or blacklists into common browsers such as Chrome, Firefox, and Internet Explorer to prevent them from attempting lookups for pseudo-TLDs.

8. OBJECTIVES ASSESSMENT

OnioNS achieves all of our original requirements:

1. **The system must support anonymous registrations** — OnioNS Records do not contain any personal or location information. The PGP key field is optional and may be provided if the hidden service operator wishes to allow others to contact him. However, the operator may be using an email address and a Web of Trust disassociated from his real identity, in which case no identifiable information is exposed.
2. **The system must support privacy-enhanced lookups** — OnioNS performs Domain and Onion Queries through Tor circuits, and under our original assumption that circuits provide strong guarentees of client privacy and anonymity, resolvers cannot sufficiently distinguish users to track their lookups.
3. **Clients must be able to authenticate registrations** — OnioNS Records are self-signed, enabling Tor clients to verify the digital signature on the domain names and check the public key against the server's key during the hidden service protocol. This ensures that the association has not been modified in transit and that the domain name is authentic relative to the authenticity of the destination server.
4. **Domain names must be provably or have a near-certain chance of being unique** — Tor hidden services .onion addresses are cryptographically generated with a key-space of $58^{16} \approx 2^{93.727695922}$ and domain names within OnioNS are provably unique by anyone holding a complete copy of the Pagechain.

5. **The system must be distributed** — The responsibilities of OnioNS are spread out across many nodes in the Tor network, decreasing the load and attack potential for any single node. The Pagechain is likewise distributed and locally-checked by all Mirrors, and although its head is managed by the Quorum, these authoritative nodes have temporary lifetimes and are randomly selected. Moreover, Quorum nodes do not answer queries, so they have limited power.

6. **The system must be simple and relatively easy to use** — Domain Queries are automatically resolved and require no input by the user. From the user's perspective, they are taken directly from a meaningful domain name to a hidden service. Users no longer have to use unwieldy .onion addresses or review third-party directories, OnioNS introduces memorability to hidden service domains.

7. **The system must be backwards compatible with existing protocols** — OnioNS does not require any changes to the hidden service protocol and existing .onion addresses remain fully functional. Our only significant change to Tor's infrastructure is the mechanism for distributing hashes for the Quorum Qualification protocol, but our initial technique for using the Contact field minimizes any impact. We also hook into Tor's TLD checks, but this change is very minor. Our reference implementation is provided as a software package separate from Tor per the Unix convention.

Finally, we meet our optional performance objectives:

1. **The system should not require clients to download the entire database** — Only Mirrors hold the Pagechain, and clients do not need to obtain it themselves to issue a Domain Query. Therefore clients rely on their existing and well-established trust of Tor routers when resolving domain names. However, clients may optionally obtain the Pagechain and post Domain Queries to localhost for greater privacy and security guarantees.

2. **The system should not introduce significant burdens to the clients** — Record verification should occur in sub-second constant time in most environments, and Ed25519 achieves very fast signature confirmation so verifying Page signatures at level 1+ takes trivial time. However, clients also verify the Record's proof-of-work, so for some scrypt parameters the client may spend non-trivial CPU time and RAM usage confirming the one scrypt iteration required to check. We must therefore choose our parameters carefully to reduce this burden especially on low-end hardware.

3. **The system should have low latency** — Domain Queries without any packet delays over Tor low-latency circuits. Its exact performance is largely dependent on circuit speed and the client's verification speed.

We therefore believe that we have squared Zooko's Triangle; OnioNS is distributed, enables hidden service operators to select human-meaningful domain names, and domain names are guaranteed unique by all participants.

9. FUTURE WORK

In future work we will expand our implementation of OnioNS and develop the remaining protocols. While our implementation functions with a fixed resolver, we will deploy our implementation onto larger and more realistic simulation environments such as Chutney and PlanetLab. When we have completed dynamic functionality and the remaining protocols, we will pursuit integrating our implementation into Tor. We expect this to be straightforward as OnioNS is designed as a plugin for Tor, introduces no changes to Tor's hidden service protocol, and requires very few changes to Tor's software. In future developments, Tor's developers may also make significant changes to Tor's hidden services, but OnioNS' design enables our system to become forwards-compatible after a few minor changes.

Additionally, several questions need to answered in future studies:

- Should the Quorum expire registrations that point to non-existent hidden services, and if so, how can this be done securely?

- How can we reduce vulnerability to phishing/spoofing attacks? Can OnioNS be adapted to include a privacy-enhanced reputation system?

- How can OnioNS support domain names with international encodings? A naïve approach to this is to simply support UTF-16, though care must also be taken to prevent phishing attacks by domain names that use Unicode characters that visually appear very similar.

- What other networks can OnioNS apply to? We require a fully-connected networked and an global source of entropy. We encourage the community to adapt our work to other systems that fit these requirements.

10. REFERENCE IMPLEMENTATION

Alongside this publication, we provide a reference implementation of OnioNS. We utilize C++11, the Botan cryptographic library, the Standard Template Library's (STL) implementation of Mersenne Twister, and the libjsoncpp-dev library for JSON encoding. We develop in Linux Mint and compile for Ubuntu Vivid, Utopic, Trusty, and their derivatives. Our software is built on Canonical's Launchpad online build system and is available online at `https://github.com/Jesse-V/OnioNS`.

10.1 Prototype Design

We have developed an OnioNS prototype that implements the Domain Query protocol. In this initial prototype, we use a static server, a single fixed Create Record, and a hidden service that we deployed at onions55e7yam27n.onion. Although our implementation is primarily a separate software package, we made necessary modifications to the Tor client software to intercept the .tor pseudo-TLD, pass the domain to the OnioNS client over inter-process communication (IPC), and receive and lookup the returned hidden service address. We diagram the relationships between our prototype's components in Figure **??**.

Our prototype's works as follows:

1. The user enters in "example.tor" into the Tor Browser.

2. The Tor Browser sends "example.tor" to Tor's SOCKS port for resolution.

3. The Tor client intercepts "*.tor", places the lookup in a wait state, and sends "example.tor" to the OnioNS client over a named pipe.

4. The OnioNS client communicates with Tor's SOCKS port and negotiates a connection to the static OnioNS server.

5. The OnioNS client performs a level 0 Domain Query to the server.

6. The server responds with the Create Record containing "example.tor".

7. The client writes "onions55e7yam27n.onion" to the Tor client over another named pipe.

8. The Tor client resumes the lookup and overrides the original "example.tor" lookup with "onions55e7yam27n.onion".

9. The Tor client contacts the OnioNS hidden service and passes the webpage to the Tor Browser.

10. The Tor Browser displays the website contents and preserves the "example.tor" domain name.

10.2 Performance

We created a rudimentary form of the Record Generation and Client Verification protocols and conducted performance measurements on two different machines. We tested these protocols on Machine A, which has an Intel Core2 Quad Q9000 @ 2.00 GHz; and machine B, which has an Intel i7-2600K @ 4.3 GHz. We chose machines with this hardware in order to more accurately determine the performance with out target audience; Machines A and B represent low-end and medium-end consumer-grade computers, respectively. To minimize latency on our end, both machines were hosted on 1 Gbits connections at Utah State University. As the Record Generation protocol requires a hidden service private key, we created a hidden service and hosted it on Machine B, using Shallot, a vanity key generator, to generate a recognizable hidden service address, `http://onions55e7yam27n.onion`. Then on Machine A, we measured the time required to connect to our hidden service over OnioNS and over the more direct hidden service protocol.

We selected the parameters of scrypt such that it consumed 128 MB of RAM during operation. This is an affordable amount of RAM by even low-end consumer-grade machines. This RAM consumption scales linearly with the number of scrypt instances executed in parallel. We used all eight cores on Machine B to generate a Record for our hidden service and observed approximately 1 GB of RAM consumption, matching our expectations. We set the difficulty of the Create Record such that the Record Generation protocol took only a few minutes on average to conduct, but in the future we may change it so that the protocol takes several hours instead.

10.2.1 Processing Time

We measured the CPU wall time required for different parts of client-side protocols. We measured how long it takes the client to build a Record from a JSON-formatted textual string, which involves parsing and assembly of the various fields; the time to check the proof-of-work PoW, and the time to check the *recordSig* digital signature.

Description	A Time (ms)	B Time (ms)	S
Parsing JSON into a Record	0.052	0.0238	
Scrypt check	896.369	589.926	
Check of $S_d(m, r)$ RSA signature	0.06304	0.0267	

Machine B, as expected, performed much faster than Machine A at all of these tasks. Parsing and signature checks both took trivial time, though the total time was dominated by the single iteration of scrypt. Record Validation protocol is single threaded and consumes 128 MB of RAM due to scrypt.

10.2.2 Latency

We compared the load latency between an OnioNS domain name with a traditional hidden service address. Our tests measured the time between when a user entered in "example.tor" into the Tor Browser to the time when the browser first began to load our hidden service webpage. We also tested `http://onions55e7yam27n.onion`, the destination of "example.tor". We performed our experiment 15 times with a different client-side Tor circuit for each by restarting Tor at each iteration. To prevent browser-side caching, we restarted the Tor Browser between tests as well.

Lookup	Fastest Time	Slowest Time	Mean Time
.tor	6.1	8.5	7.1
.onion	9.3	12.2	10.2

The latency is circuit-dependent and heavily depends on the speed of each Tor router and the distance between them. To avoid the latency cost whenever possible, we implemented a DNS cache into the OnioNS client-side software to allow subsequent queries to be resolved locally, avoiding unnecessary remote lookups.

10.3 Future Work

We have several plans for future work. First, we will port the OnioNS client to Windows and migrate the Tor-OnioNS IPC from Unix named pipes to TCP sockets. Although named pipes work reliably in Unix-like operating systems, they are not easily compatible in the Windows family of operating systems. Second, we will find more optimal parameters for scrypt to increase performance and work to reduce the latency. Finally, we will expand our implementation and collaborate with Tor developers to merge it into Tor's repositories when it is complete.

11. CONCLUSION

We have introduced OnioNS, a Tor-powered distributed DNS that maps unique .tor domain names to traditional Tor .onion addresses. It enables hidden service operators to select a human-meaningful domain name and provide access to their service through that domain. We preserve the privacy and anonymity of both parties during registration, maintenance, and lookup, and furthermore allow Tor clients to verify the authenticity of domain names. Moreover, we rely heavily upon existing Tor infrastructure, which simplifies our design assumptions and narrows our threat model

largely to attack vectors already well-understood throughout the Tor literature.

We use the Pagechain distributed data structure to prevent disagreements from forming within the network. Furthermore, every participant can verify the uniqueness of domain names. The Pagechain also has a fixed maximal length, which places an upper bound on the networking, computational, and storage requirements for all participants, a valuable efficiency gain especially noticeable long-term.

OnioNS achieves all three properties of Zooko's Triangle: it is distributed, allows hidden service operators to select meaningful domain names, and all parties can confirm for themselves the uniqueness of domain names in the database. We provide a reference implementation in C++ that should enable Tor developers to deploy OnioNS into the Tor network with minimal effort. We believe that OnioNS will be a useful abstraction layer that will significantly enhance the usability and the popularity of Tor hidden services.

12. ACKNOWLEDGEMENTS

We would also like to thank Tor developers Mark Ellzey, Yawning Angel, and Nick Mathewson for their assistance with libevent and Tor technical support and Sarbajit Mukherjee for his commentary.

13. REFERENCES

[1] D. J. Bernstein. Curve25519: new diffie-hellman speed records. In *Public Key Cryptography-PKC 2006*, pages 207–228. Springer, 2006.

[2] I. Goldberg. On the security of the tor authentication protocol. In *Privacy Enhancing Technologies*, pages 316–331. Springer, 2006.