–Independent Work Report Spring, 2015–

DNSCatProxy: A Pluggable Transport based on DNS Tunneling

Irvin Zhan

Advisor: Nick Feamster

April 30, 2015

Abstract

Tor is widely used as a circumvention tool in highly-censored countries. Obfuscation tools used by Tor, called pluggable transports, allow users to still access the Internet even when adversaries monitor TCP traffic and block Tor entry nodes. We propose DNSCatProxy, a new UDP-based pluggable transport that uses DNS tunnels, covert channel that pass all data through DNS queries. We have created and deployed a proof-of-concept that offers blocking resistance, basic active probing resistance, and a low-bandwidth entry into the Tor network. Though successfully implemented, we suggest that before DNSCatProxy is developed further, more research is needed in the robustness of DNS tunnels .

1. Introduction

The Internet has become more available globally with improving technologies and greater accessibility. However, numerous countries are increasingly employing censorship tools to block websites for a variety of reasons, such as to prevent political dissention on social media. The OpenNet Initiative in particular measures global Internet filtering and publishes their data online. Its most recent 2013 update investigated seventy-four countries for any sort of Internet filtering among four content categories: political, social, Internet tools and conflict/security. 38 of these 74 countries employed some sort of filtering, anywhere from openly blocking certain websites found objectionable or disguised attempts as network errors [20]. These types of censorship have numerous implications for all individuals, from journalists and researchers to simply individuals who are unable to access Youtube or Facebook.

In addition to countries that employ censorship tools, other governments monitor user traffic for surveillance reasons. Most notably, recent exposure of NSA's PRISM by Edward Snowden identified a worldwide surveillance program employed by the United States. Other countries use sophisticated surveillance tools to monitor and spy on online traffic, including Iran [15] and Syria [35].

Numerous circumvention and privacy tools have been created and expanded upon as a result. One notable tool is Tor, a protocol first developed in the 1990s based on onion routing, and later expanded on by the Naval Research Laboratory in 2004 [7]. Tor offers a low-latency method for anonymous communication. It conceals a user's location and traffic by forwarding it through a network of thousands of Tor relays. Though privacy is its main concern, the Tor network is also able to function as a circumvention tool as long as a user is able to connect to one of its entry relays. Tor has achieved increasing popularity for hosting numerous anonymous sites, such as being a host for Silk Road, an online anonymous marketplace for illegal goods [3]. With revelations on NSA's surveillance in online traffic by Edward Snowden in June 2013, increased concern for privacy saw a spike in Tor usage as well. Tor's popularity remains strong, and is currently estimated to have around 2 million daily users worldwide [21].

Many countries such as China and Iran have begun blocking Tor usage within their own countries. The employ a variety of methods, from simple techniques like blocking known IP addresses of entry relays, to traffic analysis like deep packet inspection (DPI). In response, Tor has developed a set of relays and protocols that deal with these issues. This paper focuses on pluggable transports (PT) [1], a protocol created by the Tor Project to transform Internet traffic flows to prevent DPI by censors. Pluggable transports in the past have disguised traffic as innocuous network packets or even Skype conversations [18]. However, existing pluggable transports do have potential limitations, and for users of Tor Browsers, all pluggable transports are based on TCP traffic which are usually filtered by proxies in censored countries.

We argue that a pluggable transport that doesn't rely on any TCP-based connections will contribute to the existing arsenal of pluggable transports. Consider the individual who may be behind a proxy blocking all HTTP traffic, such as that of captive portal. The individual would be unable to access the Internet with current tools since existing pluggable transport tools cannot bypass that proxy. A UDP-based pluggable transport in general is useful to have. ISPs are known to throttle encrypted

TCP connections [38], and censored countries have done so in the past. Since DNS queries are UDP-based, such throttling will not affect its usability. We thus propose a pluggable transport designed to work using a DNS channel through DNS queries and replies. Using a technique known as DNS tunneling, these individuals will likely be able to bypass the proxy. If that individual too is in an area where circumvention tools are needed, the DNS queries and replies can further reach beyond the proxy to the Tor network, bypassing all TCP filtering. Though such DNS tunnels have a limit on their bandwidth, this low-bandwidth tunnel can be a last-ditch solution for these individuals. Many DNS tunneling tools already exist; however are specifically built for circumvention in highly-censored countries. In the arms race between circumvention and censorship, such a pluggable transport would be valuable since it uses a completely new channel, thereby increasing the number of orthogonal types of blocking resistance employed by Tor.

We have created, along with this proposal, a proof-of-concept named DNSCatProxy, a Tor pluggable transport acting as a DNS-based circumvention tool, the first of its kind. It uses a DNS tunnel based on a modified version of a newly released DNS-tunneling library named dnscat2 [6] to transform data within the goptlib pluggable transport framework [28]. It features two modes of operations, either by disguising data sent directly to an IP address as DNS packets or sending data recursively as DNS requests. DNSCatProxy is currently deployed and ready for usage, though not freely available to the public.

2. Background and Related Work

2.1. How Tor, Bridges and Pluggable Transports Work

The Tor network is comprised of volunteer-operated servers that passes messages from a source to their destination. While a non-Tor user creates a direct connection to a server, Tor users use the Tor network to tunnel their encrypted information through a series of Tor nodes. First, a Tor user obtains a list of Tor nodes from a directory server, which are servers that keep track of all the servers in the Tor network. This list includes an entry node, the first server part of the Tor network that the Tor user connects to. With this list of Tor relays, a random pathway through several relays is created so that each relay only knows where the data comes from, and where the data should be forwarded. When a circuit is finally negotiated, data can be exchanged from and to a Tor client and a server.

Tor circuits are used to support TCP streams, and refreshed every 10 minutes to decrease the chance that an adversary may link a user's online activities between two timeframes.

The Tor protocol enforces online privacy and anonymity. Relays within the Tor network are unable to link a connection's source and destination, nor are the servers that the Tor users connect to able to figure out the IP address of the Tor user. This is because the Tor protocol uses a layered system of encryption. When a circuit is established, the Tor user negotiates encryption keys for each relay of the circuit, and encrypts the transmitted data several times using these keys in the order in which the message will be passed through relays. Relays that receive a message may use their key to unencrypt one encryption layer, but will not be able to read the contents of the message since there are further layers of encryption. When an exit relay, the last relay of the Tor circuit, finally unencrypts the data, it is the original message of the Tor user that is then passed on to its intended destination. However, the exit relays and all other relays before it only know the previous and next relay of the circuit. The origins of the message thus remains anonymous, lost in the winding paths of relays before it. Compromising the system would require the ability to compromise all Tor relays, an event that is unlikely.

How might a censor block access to the Tor network? Some countries with high levels of censorship, such as China [41], block the Tor Project website, making users unable to download the Tor Browser [30]. The GetTor project was created in response to allow users to download the Tor using email; by sending a quick email to a specific email with the operating system in the body of the text, the email robot will reply with links to download the Tor Browser from popular Cloud services like Dropbox [32]. More sophisticated attacks however involve blocking user access to the entry nodes of the Tor network. Since all public Tor entry nodes are easily available on the Tor directory servers, countries have been trivially blocking the publically listed entry nodes by IP addresses or examining HTTP headers [41]. As a response, the Tor Project created a series of Tor relays that aren't listed in the main Tor directory named bridges [31]. Since there is no complete public list of all Tor bridges, ISPs are unable to block all bridges. The effectiveness of Tor bridges rely on the availability of bridges. The sheer number of Tor bridges must outpace the rate at which censors are able to block them, which has inspired projects such as TorCloud [34], a user-friendly way of deploying bridges on Amazon's EC2 cloud computing platform.

Despite the efforts at circumvention with Tor bridges, censors have still been managing to block

access to bridges and entry nodes using traffic analysis, most notably *deep packet inspection*. Countries such as Kazakhstan [23] and Ethiopia [22] employ this technique, though China and its "Great Firewall of China" has been one of the most notable players in this form of censorship [41]. Deep packet inspection (DPI) is a form of packet filtering that works by inspecting the data and possibly the header of each individual packet. DPI allows censors to quickly classify traffic and identify those that resemble Tor, meaning they can easily detect when a user is connecting to the Tor network and block them accordingly.

Deep packet inspection isn't the only type of censorship tool deployed by China. In 2011, security researcher Tim Wilde uncovered the use of active probing by China to discover bridges, even those that were unpublished [40]. *Active probing* (also known as active scanning) is a network mapping technique that sends traffic to sample IP addresses and ports to see whether it responds to certain types of data [14]. In particular, the Great Firewall of China (GFC) uses active probing to check whether a given server responds to queries involving Tor, thereby revealing uncovering any bridges. The research in active probing was later expanded upon by Philipp Winters in his analysis of the GFC [41], which suggest that in conjunction with DPI, active probing allows Chinese censors to easily detect suspected Tor traffic and confirm the IP address and port of a Tor bridge. The need for effective countermeasures that disguise Tor traffic and is resistant to active probing led to the creation of pluggable transports.

Pluggable transports by its specification [1] sit between a Tor client and Tor bridge, and transform data between them to look like innocuous network traffic. A tool for obfuscation, pluggable transports exist that can disguise traffic to look like Skype calls [18] or even arbitrary formats [8]. Pluggable transports thus make it harder for DPI boxes to categorize traffic and identify that of Tor. Some pluggable transports are even more powerful and are resistant to active probing as well [42]. Pluggable transports are configured to run on a bridge by adding a few lines to the Tor configuration file (known as torrc). And though pluggable transports are built for Tor, some of these pluggable transport implementations are standalone processes that can function alone as a traffic obfuscation tool (e.g. obfsproxy [33]).

We will discuss in-depth more of these pluggable transports in the next section of the paper.

2.2. DNS Tunneling

DNS tunneling is the use of the DNS system, which is normally available to everyone, to tunnel ordinary traffic through it. A network often handles DNS traffic differently than normal HTTP requests. Proxies may capture all HTTP traffic, but DNS traffic typically does not reach the proxies, making it a way to circumvent restrictive firewalls [10]. For instance, captive portals that are typically found in hotels and airports allow for DNS queries to be resolved so a connection can be made to a given url. Once the connection is made however, the session is hijacked to display a login page. For these scenarios in which a network allows no other traffic but DNS lookups, DNS tunneling may be the last resort to getting online.

To tunnel traffic over DNS, an authoritative DNS server over a particular domain is needed to act as a proxy disguised as a DNS server. An authoritative DNS server tells all queries for a given domain to be delegated to a specific server, in this case the disguised proxy server. A client may establish a DNS tunnel by first encapsulating its data and transforming it into a DNS query. Typically, on the client side, this is done by first breaking down the data into several sized chunks, and using base32 or base64 encoding to transform the data into a DNS query. The query will then be sent to the domain that is linked to the authoritative DNS server. When the DNS query finally reaches the proxy server, it can then respond to it with a variety of responses such as TXT record or EDNS0 message, thereby establishing a bidirectional connection.

A DNS tunnel has several limitations. First and most importantly, a DNS tunnel operates at extremely low-bandwidths, often not exceeding speeds at around a few hundred KBps. DNS queries from the client specifically have a small character limit, and cannot be sent too regularly to disguise it as normal DNS traffic. Second, recent research into the irregularity in the DNS traffic has increasingly made it easier for a DNS tunnel to be identified. Recent research has now pointed at the use of sophisticated techniques such as character analysis [4], network flow analysis [9] and artificial intelligence [11] These limitations will be discussed more in-depth in a later section of the paper.

2.3. Related Work

As of April 2015, there are no available circumvention tools intended for anonymous browsing based on DNS tunneling. Similarly, there are currently no released UDP-based pluggable transports,

though UDP-based pluggable transports and UDP streams have been in discussion [19]. A variety of Tor pluggable transports have been implemented that work over HTTP, as well as DNS tunnels that are intended for use for a variety of reasons. A few popular ones will be briefly discussed here.

2.3.1. Pluggable Transports

- **obfs3**[27] obfs3 is a blocking-resistant pluggable transport currently bundled with the Tor Browser and very popular among Tor users that need to use bridges. The successor to an early pluggable transport named obfs2 [26], obfs3 fixes a few issues with man-in-the-middle attacks (MiTM) with a Diffie-Hellman key exchange. obfs3 however is vulnerable to active probing.
- **FlashProxy**[39]

  FlashProxy turns web browsers into bridges using web sockets. Any user with a web browser that runs Javascript and has support for WebSockets can be used as a proxy for Tor relays and a client. By creating ephemeral and a large number of FlashProxy bridges, it can outpace the rate at which censors can block bridges. FlashProxy is currently deployed.
- **ScrambleSuit**[42]

  ScrambleSuit is a lightweight pluggable transport that uses a variety of ways to defend itself against deep packet inspection and active probing. It's polymorphic characteristics make it hard for censors to use flow analysis, and use a shared secret to block against active probing. ScrambleSuit is currently not deployed, but will be soon.
- **meek**[25]

  Meek is a pluggable transport that uses a third-party server that is unable to be blocked by a censor, such as a CDN. It uses domain fronting to talk to Tor relays, and TLS for obfuscation. Meek is currently in the process of being developed.

2.3.2. DNS Tunneling

- **iodine**[12] iodine allows users to tunnel IPv4 data through a DNS server. It is one of the most popular DNS tunnels available, and offers security features such as logins. iodine requires a TUN/TAP device to function properly.
- **dnscat2**[6] dnscat2 is a recently released DNS tunnel for command and control (C&C) servers written in C and Ruby. It is currently in beta and being actively developed. At the time of this writing, dnscat2 lacks support for any TCP connections, and does not implement congestion

control or a reliability layer.

Some discussion among Tor developers in Tor forums [36] and the tor-dev mailing list [37] about implementing a DNS tunneling transport had been made recently as well. These discussions were continued privately via email, and ultimately helped advise this project.

3. Goals and Challenges

Tor and DNS tunneling solve two related problems: anonymous browsing and the availability of Internet. Tor does the anonymity part very well. Though Tor is used in highly censored countries as a way to circumvent nation-level censorship tools, Tor's primary focus is to stay anonymous from adversaries listening in on network traffic. DNS tunneling is excellent for guaranteeing availability of the Internet; it is a circumvention tool that is nearly always available, even in scenarios with firewalls that don't allow any HTTP traffic. Though anonymous browsing and availability of the Internet are two separate goals, these goals often go hand-in-hand. Countries with high rates of censorship are usually areas in which anonymity matter the most. Journalists and activists in Syria and Iran for instance use Tor's anonymous browsing to avoid prosecution, but still need to use Tor's circumvention tools to get around censorship tools.

Our main motivation in our proposal of DNSCatProxy is creating a DNS-based circumvention tool, specifically a pluggable transport for accessing Tor. There is a natural symbiotic relationship from combining these two circumvention tools. The increased availability of DNS tunnels will help users access Tor using a channel that is poorly monitored relative to regular HTTP traffic [4]. If anything, the creation of a DNS-based pluggable transport will help create orthogonal blocking resistance; an adversary would find it more difficult to handle layers of data obfuscation if another pluggable transport could be used in conjunction with DNSCatProxy.

3.1. Design Goals

We have two major design goals that we want to achieve with the implementation of DNSCatProxy.

1. **Traffic based solely on DNS queries:** As discussed previously, we want DNSCatProxy to be as different as possible from existing pluggable transports to create orthogonal blocking resistance. Pluggable transports implemented so far are all TCP-based and must pass through

firewalls. If TCP connections were suddenly all unusable for some reason, we want to make sure DNSCatProxy would still be able to function as a last-resort by not relying on any other data transport besides DNS queries. This is the main focus of this paper.

2. **Usability:** We want DNSCatProxy to be accessible to everyone, and function reasonably well enough for people to actually use. DNSCatProxy's implementation should consider mimicking DNS traffic enough to evade censorship, but provide enough bandwidth to access the Internet at a reasonable speed. Usability also involves making sure the implementation works on all operating systems, and having ways to easily scale bridges running DNSCatProxy depending on its demand.

We also have two secondary goals that are desired, but may be out of the scope of this paper.

1. **Active Probing:** A bridge running DNSCatProxy should be disguised and function as any authoritative DNS server unless it is talking to a specific Tor user. Given an IP address of a bridge running DNSCatProxy, an adversary should not be able to establish a Tor connection with that bridge. The bridge should not exhibit any behavior to that adversary that defers from that of a true DNS server.
2. **Flexibility:** On its own, the DNSCatProxy and other pluggable transports may be vulnerable to specific attacks targeting each specifically. However, allowing another pluggable transport to be combined with DNSCatProxy may be useful to further increase circumvention. For instance, the FTE pluggable transport may be used to transform data to arbitrary file formats, which can then be sent over DNS using DNSCatProxy.

3.2. Threat Model

Our threat model consists of four major players. First, we have a **client** whose desire is to access the Internet in an area where there is mass censorship. The client is assumed to be able to install arbitrary programs on his computer and at minimum, be able to access an installation of Tor and its pluggable transports, including DNSCatProxy, from either the Tor Project website directly or through other means. Second, we have a **bridge**, an unlisted Tor entry relay that is correctly configured to run DNSCatProxy and is outside of the censored region. The bridge also includes other tools that don't directly interact with Tor, but may help with circumvention, such as a legitimate DNS server.

Our circumvention tool is **DNSCatProxy**, a pluggable transport that relays information back and forth from a client to a bridge, and includes software installed on both the client and bridge. Our **adversary** is a nation-wide censor whose goal is to identify users that are attempting to access Tor and block all such access. The adversary primarily does so by identifying and blocking Tor entry relays that the client can connect to.

Passive Monitoring and Active Probing: We assume that the adversary has already blocked off access to the publically listed Tor relays, making the use of bridges necessary. Furthermore, we will assume that the adversary has similar capabilities censorship tools with that of the Great Firewall of China. Specifically, we assume that the adversary is able to use deep packet inspection *on all TCP traffic, though not on DNS*. Furthermore, the adversary is capable of active probing: scanning arbitrary IP addresses and ports, and sending them queries to test whether they are bridges.

Limited TCP Connectivity: We assume that an adversary is throttling all TCP connections, or in the worst case, hijacking all HTTP and HTTPS traffic, and blocking all other TCP services. As mentioned earlier, nation-wide censors and ISPs have throttled encrypted TCP connections in the past, though UDP-based traffic has been largely unaffected. Blocking all TCP services and hijacking HTTP is also a common practice in captive portals employed by airports and hotels.

Limited DNS Tunneling Detection: Detection of DNS tunneling is a serious issue as it serves as the main transport of DNSCatProxy. Such tools for detection and possible mitigation techniques will be explored towards the end of the paper. However, preventing an adversary from detecting a DNS tunnel is beyond the scope of this paper. We will assume that the adversary is able to look at DNS traffic but have not invested in the tools yet to detect DNS tunneling with certainty. An adversary suspicious of a particular bridge will not block it unless it can initiate a Tor connection with it.

Note that our threat model has several limitations because of this last assumption. Highly-censored countries are currently not suspected of performing any deep packet inspection analysis on UDP-based DNS traffic, or have any nation-wide tools deployed for DNS tunnel detection. DNSCatProxy may likely have high levels of success in the beginning. However, with increased use of DNSCatProxy, censors will surely deploy more sophisticated tools. Dealing with these attacks require a more sophisticated threat model, requiring dedicated research. Our contribution in this paper is thus DNSCatProxy, which handles the other parts of a threat model for a DNS-based

pluggable transport, leaving DNS tunneling detection mitigation, a separate topic itself, to future research.

4. DNSCatProxy: Setup

A few setup steps are needed for both the client and bridge for any implementation of DNS tunnels and pluggable transports, even those not specific to DNSCatProxy. An adversary may block these steps, though we will show that the client and Tor network is still likely be able to work around the censor.

4.1. Client

A client looking to access the Tor network for the first time needs a working copy of all relevant software. We had earlier assumed that the client is able to download Tor and the DNSCatProxy client software. This particular assumption is reasonable due to the various ways our client could find such software. Even if the Tor Project website is blocked, the client could find links to download the Tor bundle using GetTor [32], an email-based distribution system for Tor using cloud-based download links. There are perhaps other, more unconventional ways to acquire the software as well such as peer-to-peer software distribution. The adversary may tamper with the copies of Tor downloaded by the client, though if they are modified, signatures are available online for clients to verify its integrity [32].

All bridges need to communicate at least some information to the client before a connection can be established. Typically, pluggable transports require the bridge to send the client its IP address. DNSCatProxy too needs to send the client some data, specifically the domain name of which the bridge is an authoritative DNS server. Clients may be able to generate and access this data directly through the Tor Project's BridgeDB program [31]. Alternatively, clients can acquire locations of new bridges by sending an email to an automated Tor bot. This information is simply a domain name and clients can employ numerous methods of obtaining this information.

4.2. Bridge

Bridges simply need to have an active copy of Tor, properly configured Tor configuration files, and the relevant pluggable transport software. However, for a DNS-based pluggable transport, such

bridges need to be properly configured to act as an authoritative DNS server. This requires bridge operators to acquire a domain, point its nameservers to the bridge's IP address, and allow the bridge to accept UDP traffic on port 53, the port used for DNS traffic.

5. DNSCatProxy: Connecting to the Tor Network

5.1. Pluggable Transport Protocol

Pluggable transports have a specific specification that must be implemented. In particular, a typical pluggable transport requires the establishment of a pluggable transport client/server pair in addition to the Tor server/client pair. The Tor client communicates to the PT client through a local SOCKS connection. The PT client then uses the established pluggable transport protocol to send obfuscated traffic to the PT server; this step in particular is one in which an adversary is able to passively listen or actively step in. Finally, the PT server communicates to the Tor Bridge using an Extended ORPort connection, which allows for communication of the data sent by the client as well as metadata such as the type of PT used. The full details of the pluggable transport protocol is covered in its specifications [1].

In addition to the features outlined by the pluggable transport specifications, there are two other desired features of DNSCatProxy in regards to its distribution as a pluggable transport.

- **Support for Windows, Linux and Darwin:** Most Tor users use these operating systems, and the implementations of DNSCatProxy should certainly support them at minimum. Implementations should also consider parts of the codebase that are natively supported in these operating systems; for instance, some DNS tunneling libraries require TUN/TAP functionality, which does not have native support in Windows. Support for Android is highly desirable as well.
- **Language capable of deterministic builds:** Deterministic builds are packages that are "byte-to-byte identical no matter who actually builds them, or what hardware they use." [24]. Deterministic build processes help protect against targeted attacks, namely "water hole" attacks which use malware to distribute malicious code through the build process. Common languages that are capable of deterministic builds include C, Go and Python.

5.2. DNS Tunneling

The PT client and server are simply equivalent to DNS tunneling client and server pairs. The design of the DNS tunnel must have four primary characteristics:

- **Maintain connections**

  DNS queries are UDP-based, which is connectionless. DNS tunnels by design must be initiated by the client with a DNS query, to which a server responds. Some types of DNS tunnels however may not require a bidirectional transfer of data, such as a compromised computer reporting its status to a botnet. Others may work fine knowing all data transfer needs will be initiated by the client and can be contained in one server response. However, our DNS tunnel requires the server be able to initiate transmitting data back to the client, and the ability to send data in chunks across multiple replies. This may require some sort of periodic pinging.

- **Congestion Control**

  DNS queries are sent using the UDP protocol, meaning congestion control is not built in. Congestion control *must* be implemented due to the higher levels of DNS traffic used in DNS tunneling. The threat of collapse of DNS servers due to high levels of DNSCatProxy traffic is a serious issue.

- **Minimal Overhead**

  DNS tunneling as covert channels are unique in that they solely rely on DNS queries, consisting of a single UDP request from a client and a single UDP reply from the server. Other channels should not be used at all for DNS tunnels to be effective. The amount of data needed from outside channels in a DNS tunnel should be minimized. In particular, the data should be restricted to a domain name used to connect to the bridge and perhaps an asymmetric key.

- **Reliability Layer**

  The pluggable transports protocol does not enforce a global reliability layer, and therefore, DNSCatProxy must implement its own reliability layer [1]. Reliability is not built into the UDP-based DNS queries itself, and such features must be built within the handling of packets for DNSCatProxy.

  There are some desired secondary features as well:

- **Multiple sessions**

  DNSCatProxy should be able to support multiple users connecting to the bridge at once through the same domain name. Furthermore, DNSCatProxy should be able to act as the authoritative DNS server for multiple domain names. This allows one deployed bridge to be beneficial to a larger number of clients.

- **Shared Secret**

  To combat active probing, a cryptographic key must be given to both the server and client before a DNS tunnel is initiated. An adversary that tries to initiate a Tor connection to a DNS server should be unable to without the shared secret.

- **Forwarding DNS Traffic**

  Related to shared secrets, an adversary that actively probes that fails to guess the shared secret should not suspect that the bridge isn't a real DNS server. DNSCatProxy should implement a feature that allows unauthenticated DNS queries to be forwarded to a real, upstream DNS server.

5.3. Implementation Suggestions

While most pluggable transports are purposely designed for use in Tor, we suggest that the DNS tunneling portion of DNSCatProxy be as decoupled from Tor as possible. It should operate as a stand-alone DNS tunneling software for several reasons. First, such a DNS tunnel may be useful in areas in which circumvention is desired, but the overhead of Tor's anonymity features is not. For instance, this DNS tunnel may find widespread use among users behind captive portals in cafes or airports. An open-source project with widespread use would likely have more contributors and bug reports, which overall improve the health of the entire DNSCatProxy project. Second, a simpler and more abstract interface for the DNS tunnel portion of DNSCatProxy allows easier composition of pluggable transports. Another pluggable transport may transform the data in some way, but use DNSCatProxy's DNS tunnel implementation as their main transport.

When implementing DNSCatProxy, note that efforts should prioritize creating a robust DNS tunnel first. Relative to other pluggable transports, DNSCatProxy by nature relies less on data obfuscation and more on using covert channels for data transport. Most of DNSCatProxy's effectiveness will rely on how well the underlying DNS tunnel is able to transmit and disguise data.

Tor developers are largely language-agnostic beyond the requirement that the language have

deterministic builds. However, there are certainly some languages that are easier to work with than others for developing DNSCatProxy. Pluggable transport libraries have already been implemented that are written in Go [28], and Python [29] [33]. Pluggable transports are also suggested to be written in memory-safe languages; this doesn't rule out C or C++ specifically, but requires more auditing from Tor developers [16].

6. Proof of Concept Design Decisions

We had built an implementation of DNSCatProxy as a proof of concept with most of our desired features. We left out reliability and congestion control, features that are hard to code from scratch and are beyond the scope of a 1-semester independent work project. However, these features were in mind during implementation, and could be easily added onto the existing project.

Following these pluggable transport specifications, we designed and deployed a working implementation of DNSCatProxy based off of the goptlib and dnscat2 frameworks on Amazon's EC2 servers.

6.1. Pluggable Transport Protocol

Early on in the design process, we were faced with choosing between the greater flexibility in implementing the pluggable transport portion from scratch, and an existing pluggable transport framework. We ultimately decided to go with the latter option. There are a few frameworks for pluggable transports available that do the bulk work of negotiating a SOCKS proxy and creating an Extended ORPort connection. We examined four particular frameworks: goptlib [28] and obfs4proxy [2] (both written in Go), and pyptlib [29] and obfsproxy [33] (both written in Python). After a bitch of research, we chose Go over Python because of its powerful concurrency primitives, though this decision could have gone either way. We ultimately chose to work with goptlib over obfs4proxy because obfs4proxy was created to specifically deal with TCP-based protocols whereas goptlib, on the other hand, was easier to modify to create a UDP-based transport.

6.2. Building a DNS Tunnel

Our long-term goals are to implement a DNS tunneling library written in Golang, since no such library is available at this time. Such a library would work a lot better with goptlib. However, for a

proof-of-concept, we decided to use an existing DNS tunneling library to handle the traffic, using Golang's os/exec package to spawn a process that runs the tunnel.

Numerous DNS tunneling libraries exist, though only a handful were actively maintained and suitable for this project. Two libraries are mentioned in the previous related work section. We ultimately chose dnscat2 for several reasons. First and most importantly, dnscat2 ran over all operating systems. Unlike other DNS tunneling libraries like iodine, dnscat2 does not require any binary kernel installations or require native TUN/TAP functionality, a feature that Windows lacks. Second, dnscat2 offered a simple, easy to understand DNS tunneling protocol that made it easy to reason about. Third, the author of the dnscat2 project had documented much of the code extensively, meaning hacking it to work with Tor would be much more pleasant. Finally, in the spirit of open-source work, dnscat2 had only recently released its beta version in March 2015. Choosing to use dnscat2 for DNSCatProxy would have helped the project with feature expansion and bug fixes (note: our efforts in implementing DNSCatProxy ultimately did lead to the discovery of software-breaking bugs [13], which were later merged into the main dnscat project).

DNSCat2 offered a few features that would have been particularly useful for development. DNSCat2 features the most common DNS message types such as TXT, MX, CNAME, A and AAAA. It offers a passthrough mode, allowing unhandled requests to be passed to an upstream, real DNS server. DNSCat2 also offers two modes of operations: recursive or direct connection. Users can run the DNS tunnel through the DNS hierarchy, but can also choose to connect directly to a server by IP address or port, a useful tool for debugging. In a direct connection, packets are still disguised as DNS traffic, though they are easier to detect.

What DNSCat2 lacks however is an efficient way to tunnel traffic between a client and server. DNSCat2 is primarily intended to be used as a command and control server, a low-bandwidth way for operators of botnets to run simple commands on compromised computers. Much of the work done in DNSCat2 focuses on building an interface for its command protocol, and does not immediately support the transfer of data.

6.3. DNS Tunneling Protocol

DNS traffic comes in pairs: a client sends a DNS query and a server gives it a response. Thus, a protocol was designed to deal with these communication constraints. The transport protocol based

on DNS tunneling used for DNSCatProxy is modeled after that of dnscat2 [5]. The DNSCatProxy protocol creates a connection that works for any polling methods such as ICMP Ping in addition to repeatedly sending DNS queries. All data mentioned in this protocol are hex-encoding strings, and periods in the domain name of the DNS query are ignored.

Three types of packets are defined: SYN, MSG, and FIN. The client sends over a SYN packet to initiate a connection, to which the server replies with another SYN packet to establish the connection. The client then periodically polls the server by sending it a MSG packet, possibly with data, to which the server may send a MSG packet back containing more data. Because a server can only reply to DNS queries, the server sending large amounts of data will need to send it in parts over each MSG packet to arrive. A client on the other hand may send over data as soon as it's readily available. To close a connection the client can send a FIN packet, to which the server replies with another FIN packet. The server can close a connection by responding to a MSG packet with a FIN packet. Multiple connections are handled using an identifier passed in each packet. The full details of this protocol can be viewed on the dnscat2 docs page [5].

DNSCatProxy's protocol differs in a few ways from the dnscat2 protocol for security measures. When an invalid MSG is received, dnscat2 closes a connection with a FIN packet, a potential security vulnerability for an adversary that sends invalid packets to a bridge. DNSCatProxy's protocol dictates that for such events, the connection should not be dropped and rather, the MSG packet should be sent upstream to a DNS server. dnscat2 also defines a connection-less PING packet to help identify servers running dnscat2. The DNSCatProxy protocol removes this functionality since it exposes a bridge to adversaries conducting active probing.

In dnscat2's current implementation, anyone with the proper domain name can send a valid SYN or FIN packet, making it vulnerable to adversaries terminating existing connections or initiating new ones. We can avoid this by distributing asymmetric keys to both the bridge and client. Note that SYN and FIN's current implementation in dnscat2 give it ample space to hold a signature. Thus, when a client or bridge would like to initiate or end a connection, they would send within the SYN or FIN packet a signature of a pre-determined string unique to each SYN or FIN packet and known by both the client and server (perhaps an integer counter incremented for every SYN or FIN packet sent or received). If the client or server cannot verify the signature, then that particular SYN or FIN packet is ignored, and handled similarly to an invalid MSG. Note too that this signature can be

employed on MSG packets at an expense of a lower data payload.

7. Implementing DNSCatProxy Proof of Concept

The DNSCatProxy bridge was created using cloned versions of the goptlib and dnscat2 repositories, the source code of which is published on GitHub [44] [43]. The goptlib library was first modified to run a dummy pluggable transport with its Tor logs piped to a file, and deployed on Amazon EC2.

dnscat2 was also first deployed to Amazon EC2 as well. The deployed instance acted as an authoritative DNS server for an owned domain, simpleapp.me. Simple commands were sent to the server in both direct and recursive mode to ensure the library itself was working correctly.

dnscat2, though it did not have a tunneling feature, had a similar mode of operation named console, a mode that a client can initiate to transfer ASCII-type messages between the client and server. That became the starting point of the development. Among other things, there were seven main changes to the console mode of dnscat2 to allow it to pass Tor data:

- **Disable responses to PING packets:** Such packets were simply forwarded to an upstream DNS server.
- **Prevent closure of connection for invalid messages:** Instead of responding with a FIN packet, invalid messages were passed to an upstream DNS server. No further actions were taken and the connection remained open.
- **Verify Signatures in SYN and FIN packets:** Before a connection was initiated or ended, the contents of SYN and FIN packets were compared to a value containing the "expected signature". If the signature verification failed, the client ignored the packet or the server forwarded it to an upstream DNS server. Methods related to cryptographic signatures were stubbed out with pre-determined values.
- **Allowing UTF-8 characters:** Due to an encoding bug [13] in the dnscat2 repository, dnscat2 only allowed for ASCII characters. Such a bug was not uncovered earlier since it only appeared for small input. Handling the Tor data however made the entire DNS tunnel collapse. The bug was fixed within the main dnscat2 repository as soon as it was found.
- **Reading server stdin byte by byte:** The server uses a Ruby library called ReadLine to parse input. Among other things, ReadLine also had specific actions for certain characters, such as displaying the previous command when the up arrow was pressed. Most stdin functions in Ruby

require a newline to be seen before that incoming data is available; thus, stdin was read byte by byte using Ruby's Stdin.read(1).

- **Disabling of server stdin parsing:** Similarly, all data received by the server was passed through a parser to handle commands. This was an unnecessary overhead and was subsequently removed.
- **Force output to stderr in server:** The server had an interface that displayed many status messages in its stdout. Rather than capturing all of these stdout print calls, the data from the client was passed to the stderr port of the server. Error logging was disabled as well.

With these changes to the codebase, dnscat2 was run within goptlib using the Golang's os/exec package in a separate concurrent process (named a goroutine in Go). The dnscat2 client was run in console mode and was passed the domain of the DNSCatProxy bridge. The dnscat2 server was run in auto-attach mode, which allowed servers to automatically start transmitting data once a connection was made.

The entire project is deployed on an Amazon EC2 instance with the IP address 52.5.198.6 acting as the authoritative DNS server for simpleapp.me. The EC2 instance is a micro instance running Linux. A working local version was used for development and testing as well.

8. Discussion

8.1. Deployment

DNSCatProxy only requires a limited amount of information: the domain name of the bridge and a cryptographic key used for digital signature creation and verification. This information can be distributed in various ways. As mentioned earlier, bridge information can be securely obtained from the TorProject BridgeDB project page [31] or even through email. After this initial information is acquired, DNSCatProxy solely uses UDP-based DNS queries, satisfying a primary goal.

8.2. Blocking Resistance

The ability to easily disseminate bridge information using BridgeDB grants DNSCatProxy address blocking resistance given there is a large pool of available bridges. This will remain true as long as the rate at which the number of bridges running DNSCatProxy grows is larger than the rate at which a censor can block them. This form of blocking resistance is used by the obfs3 pluggable transport.

A single DNS server can act as the authoritative DNS server for multiple domains. DNSCatProxy can thus be running on a DNS server of an important service that the censor can't block, even if they know the bridge's IP address. For instance, requests unrelated to the DNSCatProxy protocol will be forwarded to the real DNS server. This technique grants DNSCatProxy bridges major blocking resistance if they were deployed in CDNs for instance, and is similar to the blocking resistance used by the meek pluggable transport.

Though we do not have access to a BridgeDB or a popular authoritative DNS server, DNSCatProxy was built in a way to easily handle integration with them.

8.3. Active Probing

Our adversary will attempt to establish a Tor connection with suspected bridges. DNSCatProxy however has taken some basic measures to mitigate this risk. First, note that DNSCatProxy acts as an authoritative DNS server for a particular domain name; any DNS queries to a different domain will not be interacted with. Thus, an adversary cannot conduct Internet-wide scanning of IP addresses to discover bridges without first obtaining a domain name.

Adversaries that do have domain names need to know the cryptographic keys of the client. To initiate and close a Tor connection with DNSCatProxy, the client and server must both verify the signatures of the SYN and FIN packets, information that the adversary does not know. Invalid packets that an adversary sends are forwarded to a real DNS server, meaning an adversary that sends arbitrary DNS queries to the server will get legitimate DNS replies.

Our current implementation of DNSCatProxy stubs out the cryptographic signatures with predetermined values, as active probing is only a secondary objective. Future work may involve expanding this out and experimenting with adding digital signatures to MSG packets as well.

9. Analysis

9.1. Performance

Tor and DNS tunneling both slow the browsing speed of users. We examined the impact DNSCatProxy may have on browsing speeds in Tor. We examined two types of data using our EC2 instance, first looking at the time it took to establish a Tor connection, and then the browsing speeds of three websites: Reddit (www.reddit.com), Google (www.google.com) and TorProject

(www.torproject.org). We tested each of these over HTTPS, and used the same EC2 instance deployed at 5.25.198.6. On the EC2 instance, we deployed three pluggable transports: DNSCatProxy, the obfs3 PT, and a dummy pluggable transport that simply forwards information received from a client to its server over TCP. Each experiment was performed five times and the average result is shown. Because the Tor circuits change and may impact the times, we restarted both the client and bridge for each trial. Note that due to connectivity issues, we ran the experiments using DNSCatProxy in only direct connection mode.

Pluggable Transport	Time
Dummy	4.24
DNSCatProxy	143.43

Table 1: Establishing a Tor Connection (seconds)

9.1.1. Tor Connection: Establishing a connection with the Tor Network took much longer using DNSCatProxy. The overhead in starting up a dnscat2 client and server, and connecting the two with a SYN is minimal, taking less than 3 seconds on average to complete. Most of the bottleneck is, as expected, the limited bandwidth from using just DNS queries.

Pluggable Transport	Reddit	Google	TorProject
Dummy	11.01	4.55	5.74
obfs3	14.38	4.63	4.86
DNSCatProxy	263.21	61.30	47.08

Table 2: Page Load Times of Websites in Tor Browser (seconds)

9.1.2. Page Loads: We introduced obfs3 in this measurement because of its leading popularity as a PT. Page load times were measured from the moment the url was submitted to the time when all HTML, images, javascript and other static files were loaded.

DNSCatProxy performed significantly worse than the two other pluggable transports for the same reasons as before. Most notably, Reddit took by far the longest across all pluggable transports due to the larger number of images on the frontpage. Though TorProject has far more images on its homepage, it largely outperformed Google in its Page Load times since a large number of Javascript requests were made by Google.

From a usability standpoint, is DNSCatProxy even worth it? Page load times can be a bit deceptive; in a standard page load, the HTML and CSS are loaded first, and are done quite quickly.

DNSCatProxy loaded a fully functional site in a few seconds, since much of the waiting was due to loading images and Javascript calls. DNSCatProxy thus works well enough for browsing websites in which users don't care about waiting for all images to load. DNSCatProxy may therefore have some practical usage in activities like checking email or quickly searching up information on Google. Consider for instance reaching a webpage with no Javascript, little CSS and small-sized images:

Pluggable Transport	Time
Dummy	3.53
DNSCatProxy	8.61

Table 3: Connecting to Tor Mailing List (seconds)

We examined the loading times of the Tor mailing list located at https://lists.torproject.org/cgi-bin/mailman/listinfo/tor-dev, which offers a simple form, some descriptive text, a few small logos and some minimum CSS. In this use case, DNSCatProxy offered reasonable speeds that wasn't too far off from that of a dummy proxy. In general, we have found that spending 30 minutes using DNSCatProxy through the Tor Browser was quite manageable. We were able to read Wikipedia entries and browse news articles with little impact on user experience.

10. DNS Tunneling Limitations

Since our pluggable transport relies so much on the effectiveness of DNS tunneling, it is worth having a discussion on its possible limits.

10.1. Theoretical Bound on Data Transfer

Note that DNS queries consist of a single UDP request from the client followed by a single UDP reply from a server. The client is limited by the length of the domain name, as each DNS label can contain up to 63 characters meaning each DNS query can, in theory, hold up to 255 bytes. A DNS tunnel that only uses UDP requests restricts the server's response to only 512 bytes, as larger messages are required to be truncated and sent using TCP [17]. Because DNS queries must be sent with enough time between them to look like normal, these theoretical bounds restrict the bandwidth of any protocol using DNS tunneling.

10.2. Security Concerns

Addressing all security concerns with DNS tunneling in general are beyond the scope of this paper, and throughout it, we have assumed that the adversary had extremely limited capabilities in analyzing DNS tunnels. We acknowledge that this threat model in practice may be unlikely as there are increasingly more tools being researched and deployed to stop DNS tunneling. Though most censored countries have not deployed these tools yet, if DNSCatProxy were to be more widely used, such options would be available and hard to stop without more research done.

Because of the low-bandwidth, high-accessibility nature of DNS tunnels, it has been a favorite tool of operators of botnets and command and control (C&C) servers [10]. DNSCat2 operates in this fashion; botnet operators are able to control their bots with a few commands that do not take a lot of data to transmit. Thus, the arms race between covert users of DNS tunnels and those looking to expose them are heavily one-sided, as much academic literature has been produced to look into various forms of detecting DNS tunneling.

Here is a brief summary of various proposed techniques to detect DNS tunneling

- **Flow-Based Detection**[9]

  DNS tunnels can be detected using a novel method of analyzing network flow. Network flow information is captured by monitoring DNS traffic data, and statistic tests on relevant flow-derived variables reveal any DNS tunneling traffic. Researchers have extensively tested this detection in practice with high success rates.
- **Character Frequency Analysis**[4]

  Typical DNS queries and responses in character frequencies follow Zipf's law similar to natural languages. However, DNS tunneling does not and follows a much more uniform distribution. DNS tunnels has thus been able to be detected by analyzing character frequencies of DNS traffic data.
- **Artificial Neural Nets**[11]

  With recent advances in artificial intelligence, researchers have been applying them to detecting DNS tunnel. One such proposal is using artificial neural networks to analyze domain names and point out irregularities that might suggest that a DNS tunnel is being used.

10.3. Mitigation against DNS Tunneling Countermeasures

Countermeasures against these may hinder the performance of the already low-bandwidth connection. For instance, DNSCatProxy sends a DNS query every second, but may be configured to ping the server more irregularly and less frequently, thereby making it harder to detect at the expense of lower bandwidth. Different ways of encoding messages in DNS queries, such as adding a certain amount of padding [4], can also be worth looking into to mitigate character frequency analysis, but may introduce some overhead. Similarly, artificial neural networks can be defeated by not using lowest level domains (LDDs) or using multiple requests to multiple domains [11].

However, as mentioned before, recent literature is currently finding more and more ways to deal with DNS tunneling for some very noble reasons. Thus, the use of DNS tunneling as a pluggable transport will surely increase uses of DNS tunneling. This may be both a blessing and a curse for researchers. Those who research new ways of blocking DNS tunnels help defeat botnets and cybercriminals, but allow censors more powerful tools to stop circumvention. For researchers looking into detection mitigation, the opposite is true. Widespread DNS tunneling use for circumvention and censor attempts to block it advances the arms race. It becomes a double-edged sword; better DNS tunnels help circumvention at the expense of helping cybercriminals. Thus, there are ethical implications of releasing a DNS-tunneling based pluggable transport that should ultimately be considered.

11. Future Work

There are many places to go from here. A more robust DNSCatProxy implementation could be made to handle features such as congestion control and a reliability layer. If this route was to be taken, then more research could be done into looking at ways to mitigate DNS tunneling detection, such as the few mentioned above. The dnscat2 project could be expanded as well as a result.

Since DNSCatProxy has some extreme weaknesses (low-bandwidth, easy detection with high traffic) balanced by some extreme strengths (UDP-based, uses DNS tunneling), more research can be done into investigating the parts that make DNSCatProxy effective.

Since the Tor Project largely revolves around TCP, a UDP-based pluggable transport creates a completely orthogonal method of connecting to a bridge. Encrypted TCP connections have fallen

victim to being throttled in the past, making UDP-based pluggable transports more effective in those situations. There are many options for UDP traffic that can be spoofed, and likely offer much better bandwidth that DNS tunneling.

DNS tunneling as a covert channel can be helpful for activities requiring low bandwidth and latency since DNS tunneling is much harder to detect when smaller numbers of queries are sent over a larger period of time. Relevant to the Tor Project, DNS tunneling may be used as a covert channel for distributing bridge addresses.

12. Conclusion

DNSCatProxy is a prototype for a UDP-based protocol based on DNS tunneling, the first of its kind. Because DNS traffic is rarely blocked, DNSCatProxy is potent in extreme situations when TCP connections are unusable. Due to its low-bandwidth however, DNSCatProxy should always be used as a last resort as there are many better pluggable transports out there. We have shown however with our DNSCatProxy implementation that in practice, this lower-bandwidth is manageable for certain types of online activity. Our implementation of DNSCatProxy as a pluggable transport offers some desirable features, such as high-availability, basic resistance to active probing, blocking resistance, and ability to be easily used for purposes outside of Tor.

Due to the increasing literature around DNS tunnel detection, the effectiveness of DNSCatProxy will likely decrease over time. We suggest that other UDP-based pluggable transports should be investigated, which might deliver some of DNSCatProxy's greatest strengths while solving its issues with bandwidth and reliability. DNS tunneling's usefulness can also be explored in other areas of Tor, such as bridge distribution.

13. Acknowledgements

A huge thank you to Professor Nick Feamster for advising me throughout this semester for my independent work. I want to also thank Yawning Angel, David Fifield, George Kadianakis, David Stainton, and Philipp Winter of the Tor community who had helped advise and answer questions, in addition to putting forward the efforts in developing Tor and PTs to make this project available in the first place. Thank you to Ron Bowes, the author of dnscat2, for closely working with me in hacking dnscat2, and for generally being incredibly helpful in answering questions.

References

[1] torspec. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt

[2] Y. Angel, "Yawning/obfs4/bofs4proxy," 2015. Available: https://github.com/Yawning/obfs4/tree/master/obfs4proxy

[3] M. J. Barratt, "Silk road: Ebay for drugs," *Addiction*, vol. 107, no. 3, pp. 683–683, 2012.

[4] K. Born and D. Gustafson, "Detecting dns tunnels using character frequency analysis," *arXiv preprint arXiv:1004.4358*, 2010.

[5] R. Bowes, "Dns transport protocol," 2015. Available: https://github.com/iagox86/dnscat2/blob/master/doc/protocol.md

[6] R. Bowes, "iagox86/dnscat2," 2015. Available: https://github.com/iagox86/dnscat2

[7] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[8] K. P. Dyer *et al.*, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 61–72.

[9] W. Ellens *et al.*, "Flow-based detection of dns tunnels," in *Emerging Management Mechanisms for the Future Internet*. Springer, 2013, pp. 124–135.

[10] A. Fruz, "Dns tunneling," 2014. Available: http://resources.infosecinstitute.com/dns-tunnelling/

[11] J. Hind, "Catching dns tunnels with ai," *Proceedings of DefCon*, vol. 17, 2009.

[12] iodine, "iodine," 2015. Available: http://code.kryo.se/iodine/

[13] izhan, "Github issue: Certain input for server in console mode throws error," 2015. Available: https://github.com/iagox86/dnscat2/issues/43

[14] L. Jorgenson, "Where could i find info on active probing technology to monitor network environments?" 2015. Available: http://searchnetworking.techtarget.com/answer/Where-could-I-find-info-on-active-probing-technology-to-monitor-network-environments

[15] W. S. Journal, "Iran's web spying aided by western technology," 2009. Available: https://metrics.torproject.org/userstats-relay-country.html

[16] G. Kadianakis, "[tor-dev] how the pluggable transports factory works, or: How to deploy your very own pluggable transport," 2013. Available: https://lists.torproject.org/pipermail/tor-dev/2013-August/005231.html

[17] P. Mockapetris, "Rfc 1035—domain names—implementation and specification, november 1987," *URL http://www. ietf. org/rfc/rfc1035. txt*, 1987.

[18] H. Mohajeri Moghaddam *et al.*, "Skypemorph: Protocol obfuscation for tor bridges," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 97–108.

[19] S. J. Murdoch, "Comparison of tor datagram designs," Technical report, Nov. 2011. www. cl. cam. ac. uk/~ sjm217/papers/tor11datagramcomparison. pdf, Tech. Rep., 2011.

[20] Opennet.net, "Filtering data | opennet initiative," 2015. Available: https://opennet.net/research/data

[21] Opennet.net, "Tor metrics," 2015. Available: https://metrics.torproject.org/userstats-relay-country.html

[22] T. T. Project, "Tor project: Ethiopia introduces deep packet inspection," 2012. Available: https://blog.torproject.org/blog/ethiopia-introduces-deep-packet-inspection

[23] T. T. Project, "Tor project: Kazakhstan upgrades censorship to deep packet inspection," 2012. Available: https://blog.torproject.org/blog/kazakhstan-upgrades-censorship-deep-packet-inspection

[24] T. T. Project, "Deterministic builds part one: Cyberwar and global compromise," 2013. Available: https://blog.torproject.org/blog/deterministic-builds-part-one-cyberwar-and-global-compromise

[25] T. T. Project, "meek," 2015. Available: https://trac.torproject.org/projects/tor/wiki/doc/meek

[26] T. T. Project, "obfs2-protocol-spec.txt," 2015. Available: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt

[27] T. T. Project, "obfs3-protocol-spec.txt," 2015. Available: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt

[28] T. T. Project, "pluggable-transports/goptlib," 2015. Available: https://gitweb.torproject.org/pluggable-transports/goptlib.git

[29] T. T. Project, "pluggable-transports/pyptlib," 2015. Available: https://gitweb.torproject.org/pluggable-transports/pyptlib.git/

[30] T. T. Project, "Say hi to the new gettor," 2015. Available: https://blog.torproject.org/blog/say-hi-new-gettor

[31] T. T. Project, "Tor project: Bridges," 2015. Available: https://www.torproject.org/docs/bridges.html.en

[32] T. T. Project, "Tor project: Gettor robot," 2015. Available: https://www.torproject.org/projects/gettor.html

[33] T. T. Project, "Tor project: obfsproxy," 2015. Available: https://www.torproject.org/projects/obfsproxy.html.en

[34] T. T. Project, "Tor project: Torcloud," 2015. Available: https://cloud.torproject.org/

[35] Reflets, "Bluecoat's role in syrian censorship and nationwide monitoring system," 2011. Available: https://reflets.info/bluecoats-role-in-syrian-censorship-and-nationwide-monitoring-system/
[36] tor dev, "Dns tunneling transport," 2015. Available: https://trac.torproject.org/projects/tor/ticket/15213
[37] tor dev, "obfsproxy dns transport," 2015. Available: https://lists.torproject.org/pipermail/tor-dev/2014-February/006250.html
[38] TorrentFreak, "Rogers fights bittorrent by throttling all encrypted transfers," 2007. Available: https://torrentfreak.com/rogers-fighting-bittorrent-by-throttling-all-encrypted-transfers/
[39] S. University, "Flash proxies," 2015. Available: https://crypto.stanford.edu/flashproxy/
[40] T. Wilde, "Tor bug 4185 testing and report," 2011. Available: https://gist.github.com/twilde/da3c7a9af01d74cd7de7
[41] P. Winter and S. Lindskog, "How the great firewall of china is blocking tor."
[42] P. Winter, T. Pulls, and J. Fuss, "Scramblesuit: A polymorphic network protocol to circumvent censorship," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 2013, pp. 213–224.
[43] I. Zhan, "dnscat2-temp," 2015. Available: https://github.com/izhan/dnscat2_temp
[44] I. Zhan, "dnstun-pt," 2015. Available: https://github.com/izhan/dnstun_pt