Obfuscating Network Traffic to Circumvent Censorship

Awn Umar

MEng Mathematics and Computer Science

University of Bristol

May 21, 2021

Abstract

This report explores some techniques used to censor information in transit on the Internet. The attack surface of the Internet is explained as well as how this affects the potential for abuse by powerful nation state adversaries. Previous research and existing traffic fingerprinting and obfuscation techniques are discussed. Rosen, a modular and flexible proxy tunnel, is introduced and evaluated against state-of-the-art DPI engines, as are some existing censorship circumvention tools. Rosen is also successfully tested from behind the Great Firewall in China.

Contents

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."
— Isaac Asimov

1 Contextual Background

1.1 Information on the Internet

Information is one of the world's most valuable resources. Everything we do relies on it: from the content that we produce and consume to all the decisions we make. Every scientific and technological breakthrough throughout history has depended on gathering, processing, and disseminating information, and making it persistently accessible for future advancements to build upon. Isaac Newton famously said, "If I have seen further it is by standing on the shoulders of Giants".

The invention of computer networks and the subsequent development of the Internet caused an explosion in the volume of information easily available and the speed with which it can be distributed to many people across large distances. This has facilitated vast creation of wealth [1] and many of the world's most valuable companies have an existential reliance on it. Google and Facebook make the majority of their profits from digital advertising, which uses harvested user data to target advertisements [2]. Digital data streaming services like Netflix and Spotify exploit low costs and high reach to profit from products that physical distribution centres cannot feasibly monetise [3].

As infrastructure develops and new technologies are introduced, the amount of information stored and exchanged over the Internet will increase as more people will have access to faster connections and more devices and servers will be connected. Satellite Internet Service Providers (ISPs) like Starlink could provide Internet access to underserved and hard to reach areas of the world. 5G massively increases the bandwidth available to cellular clients and could make the wide-scale deployment of "smart" Internet of Things (IoT) devices feasible[1].

1.2 Security of Internet Protocols

The exchange of information on the Internet is facilitated by agreed upon standards. These were designed primarily with scalability and reliability in mind with little attention being given to security. As such, most of these protocols are unencrypted and unauthenticated which has led to mass exploitation in the form of surveillance and censorship.

The Domain Name System (DNS) [5] protocol, which is used to translate domain names to Internet Protocol (IP) [6] addresses, is susceptible to attacks where an on-path adversary can inspect and modify traffic, compromising data integrity, authenticity, and confidentiality. ISPs exploit these weaknesses to implement censorship [7] and harvest user data, sometimes leading to security issues [8]. DNS Security Extensions (DNSSEC)—an extension to DNS that attempts to add cryptographic authenticity—is criticized as being unsafe [9] and is not widely deployed [10].

Transport Layer Security (TLS) [11] is widely used to negotiate a channel that implements confidentiality, authenticity, and integrity for communications over computer networks. Using TLS, a server can prove ownership of a domain by providing clients with a public key that has been signed by a trusted Certificate Authority (CA), or peers can maintain a list of trusted keys that can be used to authenticate other peers. Delegating trust to a CA may be undesirable so DNS-based Authentication of Named Entities (DANE) was proposed to authenticate TLS peers without a CA using DNSSEC. However it raises its own problems and

[1] IoT devices, which are usually embedded within trusted networks, have suffered from many security vulnerabilities due to a lack of incentive for manufacturers to design secure systems and provide security updates [4]. However these issues are orthogonal to the topic of this report and so won't be discussed further.

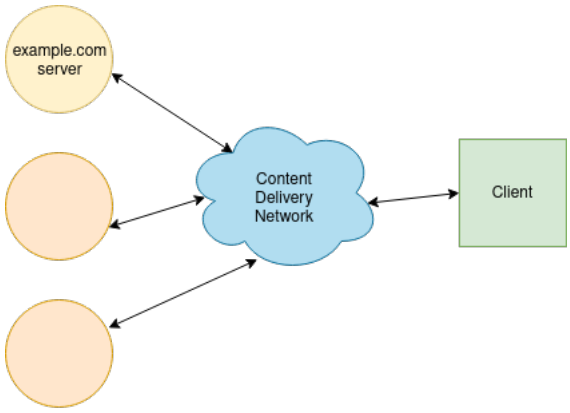

Figure 1: A client connecting to `example.com` through a Content Delivery Network (CDN) that hosts many websites.

did not gain traction [12].

Most websites on the World Wide Web have shifted from serving content using Hypertext Transfer Protocol (HTTP) [13] to HTTP Secure (HTTPS)—which typically uses TLS—so clients can be reasonably confident that they are communicating with the correct server, that the link is confidential, and that messages were not modified in transit. Similarly, DNS over TLS (DoT) and DNS over HTTPS (DoH) may be used to communicate with DNS resolvers over an encrypted channel, but they do not provide end-to-end security as the route between resolvers and authoritative name servers remains unencrypted, and they provide no guarantees about the correctness of responses other than that the client received what the resolver sent.

If a client initiates a TLS connection to a server that does not own the queried domain, it is unlikely that the server will supply a valid certificate for that domain. Therefore configuring clients to always use HTTPS[2] mitigates the risk of malicious DNS responses. This strategy only prevents situations where a malicious server is attempting to masquerade as a legitimate service. In particular, it does does not prevent DNS-based Denial-of-Service (DoS) attacks.

When clients initiate a TLS connection by sending a `Client Hello` message, the server responds with a `Server Hello` containing a certificate proving its ownership of the domain. Many websites today are hosted behind CDNs (fig. 1) which must respond with a certificate that is valid for the domain that the client requests. This information usually resides within the HTTP `Host` header that is sent *after* the TLS handshake is completed. One possible solution would be for the CDN to respond with a certificate that includes all the domains it is able to serve, but the certificate would then have to be revoked and re-issued every time this list changes, and the compromise of the private key would result in the authenticity of many websites being broken instead of just one. Server Name Indication (SNI) tackles this issue by including the domain name within the `Client Hello` message, which the server can then use to select a suitably scoped certificate to respond with. However, this information is not encrypted so an on-path adversary can exploit this vulnerability to implement surveillance and censorship. Encrypted SNI (ESNI) was an experimental attempt to fix this security flaw by encrypting the SNI payload, but it has been superseded by Encrypted Client Hello (ECH) [14] which encrypts the entire `Client Hello` message[3].

TLS is not a perfect solution. The Border Gateway Protocol (BGP) [15]—which is responsible for routing internet packets—lacks authentication meaning that someone with sufficient access can publish a false claim

[2]Servers can use HTTP Strict Transport Security (HSTS) to instruct clients to refuse to connect to their service over an unencrypted channel, but the first request remains vulnerable as the client is not yet aware of the server's policy. This is called Trust On First Use (TOFU). Preloading HSTS policies into browsers solves this issue, and there exist Top-Level Domains (TLDs) that preload all their domains.

[3]A side effect of ECH is making it much harder to perform a TLS client fingerprinting attack.

about IP ownership resulting in packets being routed to a host that does not own the associated IP address. There have been attempts to introduce BGP Security (BGPsec), an authenticated extension of BGP, but it is not widely deployed [16]. The assumption that packets are routed correctly underpins the authenticity of TLS certificates issued by CAs, but new issuances show up in certificate transparency logs which would raise a warning about such an attack. Other mitigations include using multiple servers around the world to verify domain ownership and using DNS Certification Authority Authorization (CAA) to whitelist specific CAs.

1.3 Exploitation of the Internet by Nation States

The large attack surface of Internet infrastructure is especially worrying considering that powerful entities seek to exploit it. In popular culture the combination of strict surveillance and censorship programmes is commonly associated with totalitarianism [17], [18].

1.3.1 Mass Surveillance

Many nation states and other well-positioned entities seek to collect information about people and their activities on the Internet. Mass surveillance—which is when an entire or large subset of a population is broadly monitored—is often cited as necessary in order to prevent terrorism, crime, social unrest, and to protect national security interests. Conversely, mass surveillance has equally often been criticised for violating privacy rights, being illegal, and giving governments the power to develop a "surveillance state" where civil liberties are infringed and political dissent is undermined.

Such criticisms are not without precedent. For example, COINTELPRO was a repressive and illegal domestic counterintelligence programme run from 1956 to 1971 by the Federal Bureau of Investigation (FBI) aimed at surveilling, infiltrating, discrediting, and disrupting domestic political organisations, including feminist groups, communist parties, anti-Vietnam War organisers, civil rights activists and environmental and animal rights groups [19]. "The FBI unilaterally determined whether any domestic political organizations posed a threat to national security and if so whether to authorize operations." [20]

In 2013, ex-National Security Agency (NSA) contractor Edward Snowden stole more than a million confidential documents—of which thousands have been published [21]—that reveal numerous global surveillance programs, many run by the NSA and the Five Eyes (FVEY) intelligence alliance with cooperation from telecommunication companies and European governments [22], [23]. Documents show that the NSA and GCHQ worked together to break encryption schemes and secure network protocols, control the design of new security standards, insert vulnerabilities into commercial software, gain privileged access to providers of network infrastructure, and monitor vast amounts of Internet traffic [24]–[26].

Mass surveillance does not have to be directly implemented by governments. In China, Content Service Providers (CSPs) and social media platforms are legally compelled to implement mass surveillance and censorship programmes [27], and in the United Kingdom ISPs are legally obligated to store users' Internet browsing history for 12 months [28].

Cryptography can provide some defence. For example, end-to-end encryption ensures that only participants of a conversation are able to read messages, but often this leaves valuable metadata unprotected. Even if Eve cannot read the contents of a conversation between Alice and Bob, she may be able to ascertain that they *are* communicating, as well as the time, size, or frequency of messages. In 2014 the former director of both the CIA and NSA famously proclaimed, "we kill people based on metadata" [29]. Also, cryptography cannot protect information posted to social media websites or other publicly accessible locations.

The lack of privacy on the Internet could be mitigated by anonymity [30], as then collected data cannot be associated with a specific individual or group. Unfortunately anonymity is an extremely difficult security property to implement, it involves many tradeoffs, and it may be undesirable or infeasible for many users or use cases. Tor is one example of software that aims for anonymity, but it suffers from high overhead, low adoption, and is commonly blocked because of its association with criminal activity.

Figure 2: A comment left by the developer of Shadowsocks on the project's GitHub page, informing everyone about the discontinuation of their work on the project due to pressure by the Chinese authorities. The comment has since been removed but is available at https://web.archive.org/web/20150822042959/https://github.com/shadowsocks/shadowsocks-iOS/issues/124/#issuecomment-133630294

Preventing mass surveillance is not a trivial task. It will require the cooperation of governments and legislation, the dilution of power, and a capable judicial system that will prioritise the protection of civil liberties and political expression. Research is needed into methods that are used to implement mass surveillance and technological countermeasures should be developed and incorporated into protocols and standards.

1.3.2 Censorship

Governments and other well-resourced actors around the world also seek to employ censorship—which is often dependent on surveillance—in order to assert control over the flow of information on the Internet. Motivation for this may include that such information is politically or socially sensitive, or because it is inconvenient for those with power.

For example, in China, references to the 1989 Tiananmen Square Massacre are prohibited, and access to international communication platforms is routinely blocked [31]. A white paper published in 2010 by the Chinese government prevents people on the Internet from "endangering state security, divulging state secrets, subverting state power and jeopardizing national unification" and "disrupting social order and stability" [32]. Also, beginning in 1996 the Chinese authorities systematically attacked, with the intent to eradicate, the spiritual Falun Gong movement through a multifaceted propaganda campaign involving enforced ideological conversion and re-education and reportedly a variety of coercive measures such as arrests, forced labour and physical torture, sometimes resulting in death [33]. References to the movement are prohibited and actively censored to this day. The "Great Firewall" of China is perhaps the best known example of Internet censorship by a nation-state [34], but repressive measures are implemented in many places including the United Kingdom, Turkey, Syria, Iran, and Pakistan [7], [35], [36]. It is clear that free access to information and the ability to freely redistribute it is a necessary (but not sufficient) condition for holding those with power accountable.

Legally-mandated censorship of material on the Internet is also common. For example, in the United Kingdom ISPs are ordered to block access to copyrighted materials [7]. A popular proxy tunnel used in China to bypass censorship was removed from GitHub after police came to the developer's house and told them to delete the code and stop work on the project (fig. 2).

Various methods are employed to censor information on the Internet, and various countermeasures and counter-countermeasures have been developed in the war between censorship and freedom of information. In order to break this deadlock, significant work needs to be done in understanding existing censorship systems and countermeasures in terms of their functionality, effectiveness and efficiency. This is difficult since censorship programs are often highly secretive and users within repressive countries may risk consequences for

using circumvention tools. These obstacles need to be overcome, clear adversarial models must be defined so that a systematic approach is possible, and generic circumvention frameworks must be developed that can adapt to changing adversarial conditions.

In general, a widely used, decentralised protocol would be the ideal anti-censorship tool. The intuition for this is clear: if some information is available from many different sources and the protocol for accessing it is so widely used that blocking it would involve significant collateral damage, then it becomes much harder to prevent access to such information than if it was stored on a single server. For example, BitTorrent is a decentralised file sharing protocol that is widely used for piracy, and yet has withstood significant legal and technological pressure from governments and copyright holders for a long time. However, a decentralised model represents a significantly different architecture to the highly centralised nature of the Internet so it is extremely difficult for such a protocol to displace existing ones today.

1.4 Ethic of Responsibility

Cryptography, like many other technical fields, has an intrinsically political dimension. This stems from the powerful nature of the tools it provides. Arranging a number of mathematical constructions a certain way allows someone to *configure* who can do what, how, and to what. Much of the political and societal influence of cryptographic work is *implicit*: by influencing power relations as a byproduct of technical work, as opposed to *overt*: through the mechanisms of activism and participatory democracy. The development of nuclear weapons is a powerful example of implicit politics: they are ultimately *purely* political, *despite* requiring extraordinary technical innovation.

The *ethic of responsibility* contends that scientists and engineers have an obligation to select work that promotes the social good, or, at the very least, to refrain from work that damages mankind or the environment [37]. This is justified by three basic truths: that the work of scientists and engineers transforms society, that such transformations can be for the better or for the worse, and that technical work is sometimes arcane enough that people who do such work bring an essential perspective to society. But doesn't the mathematical nature of cryptographic work exclude it from ethical and moral questions or obligations? This is the view of *technological optimists* who believe that technological innovation is intrinsically a force for good and works to improve society. If you are a technological optimist, a rosy future flows from the work you do and so all work is justifiable. After all, if every innovation you contribute to will eventually lead to a net positive outcome for society or humanity then there is no need to consider ethics. This view is exemplified by Stanley Fish, a well-known literary theorist and professor, who wrote "don't confuse your academic obligations with the obligation to save the world; that's not your job as an academic" [38]. Perhaps he is able to justify this belief to himself because literary theory is relatively harmless and in fact the right to write anything you want is a valuable and protected treasure. However in other fields this viewpoint seems myopic and can be harmful. The implicit transformative power of technical innovations *necessitates* ethical considerations. Just as physicists developed nuclear weapons, it is computer scientists, cryptographers, technologists and other academics who create the technology used around the world today to implement mass surveillance, censorship, and to suppress human rights and freedoms. It must also be technologists and academics who create the tools to fight back [39]. As such this project is not searching for an *opportunity*. Instead, it is an ethical *responsibility*.

2 Project Definition

2.1 Aim and Objectives

The aim of this project is to expand on existing research in censorship resistance by designing and implementing a modular circumvention tool that is comparable with current state of the art censorship resistance systems.

1. Develop an adversarial model for censorship systems.
2. Research and explore techniques used by existing censorship and anti-censorship systems.

Term	Definition
Sensitive flow	Network traffic targeted for being offensive or inconvenient, or for using circumvention tools.
Censor, adversary	The party that controls the communication network and is attempting to identify and restrict sensitive flows of information.
Circumvention tool	Software designed to prevent censors from identifying or blocking communications.
Background traffic	Non-sensitive flows potentially coexisting with sensitive flows on the network.
Collateral damage	Background traffic adversely affected by a censor's efforts to control sensitive flows.
Cover protocol	Target network protocol that a circumvention tool may disguise its traffic as to maximise the collateral damage of being blocked.
Proxy server	Party that will forward traffic to the broader Internet on behalf of a censored client; external to censor-controlled network.

Figure 3: Summary of often-used technical terms. Some of these are taken or adapted from [40].

3. Design and develop a modular framework for implementing obfuscating proxy tunnels, implement at least one cover protocol with it, and evaluate its effectiveness with respect to existing software.

2.2 Scope

This project focuses solely on censorship resistance for information in-transit over computer networks. We will explore the landscape of existing censorship and anti-censorship systems, and present and evaluate a new tool that fits within this. This is a large problem space and so there is a lot that is not in scope. In particular, detailed discussion and optimisation of proxy performance or evaluation of circumvention tools against real-world censorship systems is not a goal. Some points that are not in scope may be mentioned in the context of possible future work.

2.3 Vocabulary

Throughout this report we will use terms like "adversary", "censor", and "sensitive flow" repeatedly. To aid understanding, a summary of important terms is provided in fig. 3.

3 Adversarial Model

Suppose there is a user connected to a network and an adversarial actor who wants to prevent them from accessing any content that is deemed "sensitive".

The adversary could censor sensitive traffic at its source using a court order or by exerting other influence that results in the content being removed or becoming inaccessible. However, they may not have sufficient power to do so. For example, the content may be hosted by an unsympathetic entity in a different country that is outside their influence. On the other hand if they do have the ability to censor sensitive material in this way, the only countermeasure is to have a copy saved and served from elsewhere. Therefore, we will only consider censorship that happens to *information in transit*, where there is an *on-path adversary* (fig. 4) that is able to inspect, modify, remove and create messages as they travel between network peers[4]. This requires

[4]It may be possible to circumvent an adversary's ability to disrupt a connection, as demonstrated in [41]. The adversary in response may adapt their capabilities or—since anonymity [30] is not assumed—employ an unmodelled technique such as cutting off the user's access entirely or deploy legal, physical, or psychological force.

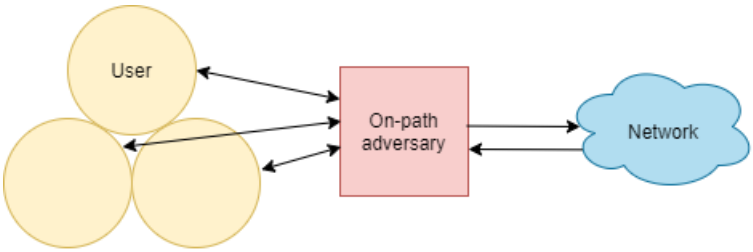

Figure 4: An on-path adversary sits between a number of clients and another network.

that the adversary have privileged access to network infrastructure. Governments typically have this via their influence over domestic ISPs, but the operator of a private network has similar powers.

Suppose there is a set of users that connect to a network through a channel C that is controlled by a passive or active adversary. A user u communicates by sending messages $\overrightarrow{m} \longrightarrow C_u$ and receiving responses $\overleftarrow{m} \longleftarrow C_u$, where $C_u \subseteq C$.

1. A **passive adversary** $\mathbb{A}_p$ has an oracle $\mathcal{O}$ that returns a copy of every message that passes through C, and has some finite memory M in which it is able to store a set of messages and their timestamps.
2. An **active adversary** $\mathbb{A}_a$ is at least as powerful as $\mathbb{A}_p$ but is also able to
   (a) Create messages $\overleftarrow{m}$ or $\overrightarrow{m}$ and send them through the channel to any reachable network host.
   (b) Remove any message, preventing it from being routed to its intended destination.
   (c) Modify a message in transit, which is equivalent to removing a message and replacing it.

The goal of either adversary is to infer something about the communication including perhaps the participants or the network protocol. An active adversary has the ability to induce an edge case that could reveal extra information. For example actively probing a network host by sending it crafted messages and seeing its responses can reveal the type of service that it hosts [42], [43]. We assume reasonable computational time and space bounds on all adversaries. For example, M can't be too large, and an adversary trying to categorise some traffic may only have a few hundred milliseconds to make a decision.

There are many possible security properties that can be constructed over these adversaries but we will consider the following:

1. **Confidentiality**: An adversary is not able to infer the data contents of any communications. This property, and other desirable security properties, are fulfilled by many encrypted network protocols such as TLS 1.3 [11].
2. **Authenticity**: Communicating parties are able to distinguish between messages sent by the other party and forged messages inserted by an adversary.
3. **Integrity**: Communicating parties are able to detect if a message changed in transit, or if one was removed.
4. **P–Indistinguishability**: An adversary is not able to distinguish between network traffic that is disguised as a cover protocol P and background traffic using protocol P.

It is clear that it is possible to achieve perfect P–indistinguishability of protocol messages since cryptographic ciphertext is indistinguishable from random noise and therefore random sequences are indistinguishable from each other [44], [45]. So you can take a protocol message and replace all cryptographic ciphertext with your own resulting in a message that is indistinguishable from the original. This suggests that the optimal strategy is to select a P that is encrypted to both provide confidentiality, authenticity and integrity, and to maximise the number of random bits in exchanged messages. Using a real implementation of the selected protocol is

called tunnelling, as opposed to randomisation [46] (where traffic is scrambled so that it is indistinguishable from random) or mimicry [47] (where packets are encoded such that they appear to belong to a given protocol), and has been shown to be more effective than alternatives [48], [49].

However, this strategy ignores other characteristics of protocol traffic such as the time, size, and frequency of messages. For example, website traffic would typically be dominated by downloads as requests are usually much smaller than responses, and there would be periods of low activity separated by bursts of traffic as the user navigates between pages. On the other hand Virtual Private Networks (VPNs) can manifest a wide range of traffic patterns depending on, for example, whether the user is browsing websites or participating in a Peer-to-Peer (P2P) file sharing network. Also, different instances of the same protocol can vary between themselves due to implementation details. We need to understand to what extent real-world censors can and do use this type of traffic metadata, typical traffic patterns for some set of target protocols, the variability of these patterns with respect to time and location, and the trade-offs between indistinguishability and protocol efficiency inherent in conforming to typical traffic patterns. One option that might allow disguised traffic to blend in better is to masquerade as a protocol that is more bandwidth-symmetric, for example by using WebSocket Secure (WSS) [50].

Perfect indistinguishability is desirable but not necessary. An important metric to consider is the false-positive rate of an adversary's P–detection algorithm, corresponding to the proportion of positive detections of P that are incorrect. A false-positive rate of 10% means that 1 out of 10 detections correspond to innocent users that were wrongfully blocked. A high false-positive rate can cause unwanted disruption. Source [48] suggests "A false-positive rate of 0.2% is likely to still be considered prohibitively high for some censors". An adversary must simultaneously balance the false-positive rate with the sensitivity (true-positive rate). A low sensitivity results in a high proportion of unwanted traffic to be mistakenly allowed through. Thus we can see that these metrics serve to somewhat constrain adversaries, and suggests that the optimal strategy is to maximise the false-positive rate, blending in with background traffic, so that it is difficult for an adversary to ascertain with certainty whether a traffic stream is disguised. This is an intuitive result.

The other attack vector is fingerprinting the endpoints of a network communication. If an adversary sees a sequence of disguised messages being exchanged between some user and a known proxy server, it can block the communication without considering any characteristics of the traffic itself. However an adversary has to first be able to discover this information about the endpoint. Tor, a popular anti-censorship and anonymity tool, faces a similar issue in maintaining and distributing a set of hidden servers ("bridges") to which users can connect without censors being able to know that they are using Tor. It has been shown that discovering bridges is simple and efficient [51].

Distributing lists of available servers to users without censors discovering them is a challenging problem and deserves attention, but it is potentially useless if a censor can discover them in other ways. So suppose a user sets up their own server and tells no one else about it. An adversary subsequently sees traffic belonging to protocol P being exchanged with this server and so attempts to open a series of connections with it. The server must respond in a way that is typical of most servers that speak P, while simultaneously proxying arbitrary traffic for the legitimate user. To achieve this we can rely on Kerckhoffs' principle [52] which states that the security of a system must depend not on keeping secret the algorithm, but only the key. If the user and the server maintain some shared key material, the server could respond with some decoy response when the shared key is not included in a request. For example, HTTPT [43], a probe-resistant proxy tool, does exactly this by including a key in HTTPS-encrypted Uniform Resource Locator (URL) paths. Indistinguishability now relies also on the confidentiality of this key and on securely distributing it.

We still need to understand typical response patterns for servers that speak a set of protocols so that decoy responses can emulate them. The proxy server may be able to simply use an existing implementation of the target protocol and redirect authenticated requests to a separate handler that can proxy connections for the user. Unauthenticated requests could be responded to with any number of decoy actions, for example a HTTPS server could return a generic error code, serve a decoy website, or redirect/reverse-proxy to a different web address.

4 Traffic Identification Methods

4.1 Endpoint-Based Filtering

The most basic form of traffic filtering is to make decisions based on the *participants* of communications. If a user is exchanging packets with a server or service that is *known* to host sensitive content, the censor can safely stop the communication without analysing the type or content of the conversation. This kind of surface-level analysis of network packets, where information from the application layer of the Open Systems Interconnection (OSI) model is not considered, is sometimes called Shallow Packet Inspection (SPI). A benefit of endpoint-based filtering is that the probability of collateral damage is strictly bounded: if the endpoint in question is categorised correctly by the censor then it will not be falsely blocked.

The main piece of identifying information within network packets is the destination IP address. When a packet is sent from A to B, it takes a "near"-optimal route through some number of network nodes that forward it according to the IP address it contains. This address is always unencrypted since every on-path network node *requires* access to it[5]. For example, if `sci-hub.se` is the target of censorship attempts, the censor can ping the domain repeatedly in order to enumerate a list of IP addresses and subsequently refuse to route any traffic to them. A commonly used countermeasure to IP address blocking is for either the user or server to setup a proxy that is unknown to the adversary that will route traffic between them. For example a user could use a VPN so that the destination of their packets is the VPN server instead of the blocked IP, or the website host can setup a mirror[6] so that users can connect to that instead of the original server (until the censor discovers and blocks the mirror as well).

IP address blocking is made easier by the fact that IPs are included in every network packet, allowing for low level decisions to be made about individual packets, with little overhead. Domain names on the other hand are ordinarily confined to application-layer DNS protocol messages, but domain-based filtering of network traffic is still possible using DNS Hijacking [36] which works by intercepting DNS responses and replacing the IPs inside them with erroneous ones. The client assumes the returned IP is valid and attempts to open a connection to it instead of the real one, resulting in either a connection failure or a *blocked page* notification being shown[7]. Requiring valid DNSSEC signatures on responses could detect an attack but won't prevent it, and may cause issues with other services due to DNSSEC's low adoption rate. Performing DNS queries over an encrypted tunnel, for example using DoT or DoH, is another option and has similar security properties to any other TLS connection. However at the moment, while these encrypted versions of DNS are popularly used to secure the connection between users and DNS resolvers, the connection between DNS resolvers and nameservers remains insecure (see fig. 5). Fortunately, it is easy to route DNS queries through a proxy server as well.

DNS Hijacking is not the only way to filter connections based on domain names. As mentioned in section 1.2, when connecting to a service that is hosted inside a CDN, browsers will include the hostname of the service inside the SNI payload. Since SNI on its own is unencrypted, it can be used to implement network filtering and censorship [53]. To combat this problem TLS 1.3 introduced ESNI which encrypts the SNI payload with the CDN's public key, but recently ESNI has been superseded by ECH [14] which encrypts the entire TLS `Client Hello` message. A side effect of ECH is that it makes it much harder to perform TLS client fingerprinting attacks. However at the moment TLS 1.3 is not universally supported, and some censors block ESNI[8] so the same may happen to ECH. The hostname of the service may also be leaked inside the server's TLS certificate, but TLS 1.3 encrypts certificates with an ephemeral key preventing Man In The Middle (MITM) attacks even when a malicious root certificate is trusted by the client. This has caused headaches for some network security systems [54].

[5] *Authenticating* IP addresses is probably infeasible as it requires that every sender has a keypair associated with their identity and that every possible on-path network node has access to their public key. Simpler challenges have remained unsolved, for example the effort to authenticate BGP has had little success.

[6] Sci-Hub itself hosts a number of mirrors to circumvent censorship (https://www.reddit.com/r/scihub/wiki/index). There is also a Telegram bot (https://telegram.me/scihubot) that searches and returns research papers as attachments: blocking it would supposedly require blocking Telegram entirely (unless Telegram themselves block the bot specifically).

[7] A phishing page could instead be shown, but using a properly pre-loaded HSTS policy should prevent this.

[8] https://mailarchive.ietf.org/arch/msg/tls/Dae-cukKMqfzmTT4Ksh1Bzlx7ws/

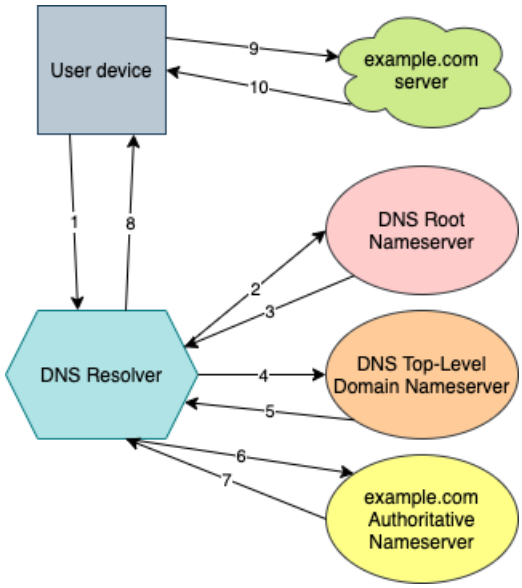

Figure 5: A DNS resolver makes recursive queries to the different types of DNS nameservers to find the result of a query involving `example.com`. Adapted from https://www.cloudflare.com/en-gb/learning/dns/dns-server-types/

Domain fronting. The existence of SNI makes it possible to implement a popular anti-censorship technique called *domain fronting*. Suppose there is a CDN that hosts two domains, `sci-hub.se` and `wikileaks.org`, and suppose a client makes a request to the CDN with `sci-hub.se` in the SNI payload and `wikileaks.org` in the HTTP `Host` header. To an on-path eavesdropper, it will look like the client is attempting to access `sci-hub.se`. The CDN will respond with a certificate that is valid for `sci-hub.se`, the two will perform a TLS handshake, and then the CDN will see that the `Host` header points to `wikileaks.org` and may subsequently forward the connection there. The `wikileaks.org` server can then act as a HTTP proxy for arbitrary traffic and thus allow the client to covertly connect to any Internet service. However, this mechanism violates the protocol specification and some popular cloud service providers now actively block this technique by verifying that the host specified in the SNI payload matches the one in the HTTP header. In a variation of this technique called *domainless fronting*, the SNI header is left empty and so the fronting may still work provided that empty SNI fields are ignored by the CDN. Domain fronting is also made slightly less impactful by the existence of ECH within TLS 1.3, which encrypts the entire `TLS Client Hello` message including the SNI field, but the IP address of the server still appears to be the CDN for any watching adversaries.

Domain shadowing is a novel and very recent development that attempts to achieve the same thing that domain fronting does except it doesn't rely on an undocumented and in many cases unsupported feature of CDNs, is supported by most CDNs, can be used to forward traffic to any domain including ones that are not hosted on a CDN, and can be completely configured by the user without assistance from either the censored website or any third party [55]. It works by exploiting the fact that CDNs allow users to claim arbitrary domains as the back-end origin and so by setting the front-end to an allowed domain it's possible to access resources of the back-end domain with all "indicators", including the connecting URL, the SNI of the TLS connection, and the `Host` header of the HTTPS request, appear to belong to the allowed domain [56]. The researchers also describe a technique that combines domain fronting with domain shadowing (*DfDs*), and another that allows setting arbitrary unclaimed domains as the front-end (*DfDs++*) [56].

Both IP address and domain name filtering require a pre-compiled blocklist of "sensitive" targets, so users who forward their traffic through a proxy server have to avoid the IP address or hostname of the proxy server ending up on blocklists. This implies the proxy server should make it difficult to discover that it is acting as a proxy server, instead it should disguise itself as some innocuous service by obfuscating its traffic and

implementing resistance to *active probing* attacks. Consideration of the proxy service's traffic patterns also plays a part, as will be discussed in section 4.2.

4.2 Traffic Fingerprinting

The success of proxy-based circumvention has forced censors to turn to Deep Packet Inspection (DPI) techniques which can make decisions based on deeper analysis of in-transit network traffic that considers information inferred from the application layer of the OSI model, including perhaps "stateful" analysis where information is compared across multiple packets. For example application-layer content in unencrypted network packets can reveal user-generated keywords like search terms. China's Great Firewall blocks traffic containing blacklisted keywords [40]. Once a host has been flagged as a proxy server, its IP or domain may be blocked so the cost of proxy server redeployment has to be kept low while also maximising the uncertainty of classifications.

Modern circumvention tools use encryption to avoid revealing data so the goal of DPI engines has also been to detect traffic from circumvention tools themselves. This includes obvious protocols for tunnelling traffic such as VPNs which will typically have unique protocol signatures that they do not attempt to hide, but it may also be possible to detect disguised or obfuscated traffic. As discussed in section 3, this is possible because of leaked information such as message time, size, and frequency. Meek, a pluggable transport for Tor that implements domain fronting over HTTPS [57], can be reliably detected with a machine learning model that uses protocol features including message entropy and timing patterns [48]. Implementation-specific protocol quirks can also be revealing, for example it's possible to detect specific implementations of TLS with a TLS client fingerprinting attack [58]. Tor has been successfully targeted by this technique [40], [59]. A possible solution to this problem is to use an implementation that is ubiquitous (for example by programmatically using a headless browser[9]) or try to randomise or mimic characteristics of another implementation[10], but these countermeasures do not guarantee perfect P–indistinguishability.

Perfect P–indistinguishability is not necessarily required (yet). The vast majority of current DPI engines use string matching or regular expression based inference techniques instead of more advanced, more complex, and more difficult to implement techniques like machine learning [60]–[63]. As such current DPI systems will need significant upgrades in order to be able to detect more advanced obfuscation techniques, but it is only a matter of time before this is feasible for censors to implement. A notable constraint on DPI engines is that they must be able to function in real-time, i.e. analysis must be faster than the traffic rate to be monitored as otherwise it would result in packet drops. A workaround is to perform *faster* validation in real-time and store traffic data to be analysed more thoroughly later. This is what the Great Firewall of China does, for example.

A nation state censor is able to use information gathered from an entire population's traffic patterns in order to make deductions. If a proxy server that is disguised as a website usually gets very few users who each exchange only a little data with it relatively infrequently, then a small set of users who exchange disproportionately large volumes of traffic with it will be suspicious. One possible workaround is to have a "decoy" service that reverse-proxies to some high-volume web service, or host such a service directly and build a community of legitimate users. Decoy services are discussed further in section 5.1.1. Nowadays cloud-hosted CDNs are popular meaning that it is quite common for large amounts of traffic to be exchanged with a relatively small set of hosts belonging to major cloud providers, so a proxy service hosted on a CDNs may find it easier to blend in. In this case the domain name of the service could still potentially be a uniquely trackable identifier for traffic to the proxy service specifically, so either TLS 1.3 or domain fronting/shadowing are necessary to hide it (as discussed in section 4.1).

[9] Selenium is an umbrella project encapsulating a variety of tools and libraries enabling web browser automation: https://github.com/SeleniumHQ/selenium

[10] uTLS is a library for substituting TLS `Client Hello` messages: https://github.com/refraction-networking/utls

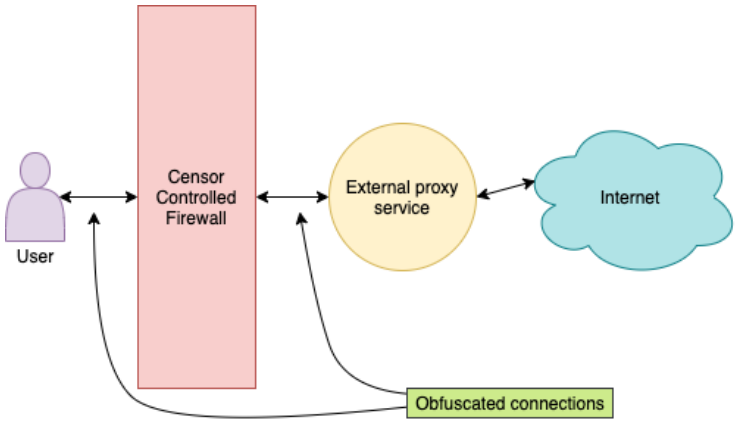

Figure 6: A proxy server acts as an innocent looking messenger for sensitive traffic that could be censored. For example, the proxy may appear to be a HTTPS service.

5 Network Traffic Obfuscation

Censorship circumvention techniques rely (roughly) on *proxying* and *obfuscation* to defeat *endpoint-based* and *in-transit* censorship methods, respectively. Proxying will present an innocuous or unknown network destination to an adversary, whereas obfuscation refers to any measure that hides the nature of the traffic itself, including hiding the nature of the service itself from passive and active attackers. Figure 6 shows how these techniques work together.

5.1 Proxying

Proxying traffic refers to any kind of situation where a defined intermediary or set of intermediaries handle the forwarding of traffic for a user or set of users. In the context of censorship circumvention, there are a number of commonly used solutions.

Virtual Private Networks are any kind of technology that can encapsulate and forward network traffic data over another network. Typically there is a VPN *client's* traffic being forwarded to the same network as a VPN *server*. Commonly used VPN solutions include OpenVPN, IPsec, and more recently WireGuard, all of which are easily identifiable by DPI engines. They are usually encrypted and serve as a relatively simple way to open secure, persistent network tunnels between networks. VPNs have legitimate uses, for example to allow employees of a company to securely connect to the company intranet from an outside network, but they are also commonly used to bypass regional restrictions and evade censorship.

The Onion Router is a proxy tool specifically focused on anonymity. It achieves this by routing traffic through the *Tor network*, a 3-layer deep mesh of network nodes composed of an entry/guard relay, a middle relay, and an exit relay, through to the Internet. This is a *Tor circuit*. Each relay knows only the identity of neighbouring nodes, so for example the entry relay knows only the identity of the user and the middle relay, and Internet services know only the identity of the exit relay. Also, data is encrypted three times, once for each node in the route, giving the scheme its name: *onion routing*. In this way, assuming that the different layers of the network are not collaborating, it should not be possible for any individual participant (except the user) to ascertain which users are connecting to which services [64], [65]. There are many published attacks against the Tor network targeting different aspects of its security model [59], [66]–[68]. Tor is detectable by DPI engines, but it comes with built in obfuscation modules called *pluggable transports* that can be used to potentially side-step local or national firewalls that target Tor traffic. *Onion services* are Tor web services that are only accessible through the Tor network, increasing the depth of a Tor circuit to 6, and providing anonymity to both the client and server. Since you are unable to ascertain the public IP address of the server it is much harder to take down and censor an onion service than it is a clear-net website.

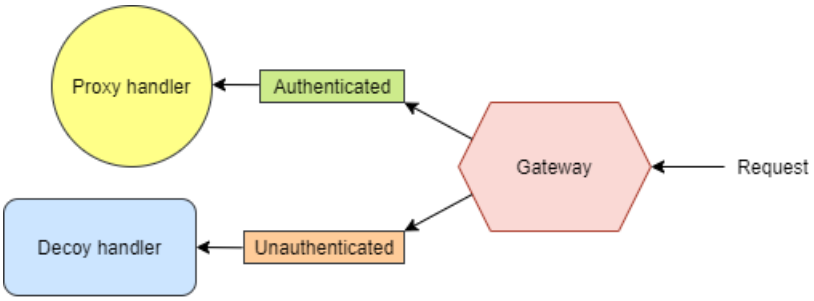

Figure 7: Authenticated requests are forwarded to a proxy handler and everything else is served by a decoy.

V2Ray is a framework for developing and deploying proxy modules[11]. It is composed of a set of proxy protocols, network transport protocols, and an obfuscation layer[12]. It is designed to be extremely modular and flexible, includes support for many protocols, and allows users to compose, chain, and deploy multiple instances of them. However, it also includes support for weak and insecure configurations, and there are a number of published attacks against it[13].

5.1.1 Active Probing Resistance

An active adversary is able to directly probe and open connections with a proxy server to gain more information about services running on it. For example if you attempt to speak the Tor protocol to a server and it responds in a way that indicates that it understands, you can be confident that the server is acting as a relay. For this reason a proxy server that wants resistance to active probing attacks must appear to be an innocuous *decoy service* to any attacker that attempts to probe it. This can be achieved by relying on Kerckhoffs's principle [52], as mentioned in section 3. Essentially, if a client is not able to prove knowledge of a shared secret then the server will not behave like a proxy. This is shown in fig. 7.

But what kind of decoy response is acceptable or ideal? There are many possibilities and they vary depending on the cover protocol that is being used. A HTTPS server could respond with a HTTP redirect, an error, or a genuine website that is hosted from the same server or reverse proxied from elsewhere. On the other hand a raw TLS server could behave in a vast number of different ways depending on the service it runs, so it's recommended to simply *read infinitely*, with an unlimited timeout, on any unauthenticated connection in order to avoid an adversary being able to trigger a protocol edge case, and to prevent fingerprinting based on specific timeout values [69].

HTTPT is a proxy that is designed to hide behind HTTPS servers to resist active probing attacks [43]. It acts as a reverse proxy with multiple backends where users authenticate by specifying a pre-shared password in the URL of the original HTTP request. Unauthenticated requests are forwarded to some conventional webserver like Apache or Nginx that serves a decoy, and authenticated requests are forwarded to the proxy handler. In their report they suggest the following options for a decoy:

1. Deploy the proxy alongside an existing webservice and forward unauthenticated requests there.
2. Reverse proxy unauthenticated traffic to some other external webservice.
3. Return an error response. They say, "As of June 2020 over 21% of the servers probed responded with `400 Bad Request`, 11.19% responded with `403 Forbidden`, 8.62% with `404 Not Found`, and 2.91% with `401 Unauthorized`."
4. Content could be copied from other websites. This would avoid latency issues with reverse proxies but could run into copyright problems.

[11] https://github.com/v2fly/v2ray-core
[12] https://github.com/net4people/bbs/issues/36
[13] https://github.com/net4people/bbs/issues/36#issuecomment-644929739

5. Restrict access by requiring bogus authentication, for example by requiring HTTP authorisation and not accepting any passwords as valid.

Note that these are mostly applicable to proxies disguised as HTTPS services.

5.2 Obfuscation

There are many different techniques for obfuscating network traffic in terms of its appearance in-transit, but they can broadly be split into three categories.

5.2.1 Randomisation

Randomising obfuscators work by scrambling messages to make transmitted bits indistinguishable from random. The technique is sometimes described as trying to make network flows "look like nothing" by totally removing their signature. This is often achieved by applying a stream cipher to every byte, but since stream ciphers on their own don't provide strong authenticity and confidentiality guarantees, they're typically applied on top of moden encryption methods that do. Examples of randomising obfuscators are Dust and ScrambleSuit.

Randomisers are reliably detectable by measuring the entropy content $H(X)$ of the set of bytes in the traffic flow, where

$$H(X) = -\sum_{x \in X} P(x) \log_2(P(x))$$

and $P(x)$ is the *measured* probability of seeing the byte x in the sequence. Since randomisers will have a uniform probability distribution, their entropy will be maximised and so

$$H(X) \approxeq \log_2(256)|X| = 8|X|$$

which should be consistently higher than any other protocol, especially in the first packet payload where many protocols put headers and other structured, unencrypted data. Randomisers are also sometimes detectable by testing simple heuristics like message length [48]. In addition, the lack of a fingerprint is itself a fingerprint and so this technique fails against whitelist-based adversaries who block any unrecognised traffic flow [40].

5.2.2 Mimicry

To address whitelisting censors, mimicry-based methods attempt to evade censorship by making packet payloads look like they belong to a whitelisted protocol instead of a random stream. It can be roughly thought of as a form of steganography against a DPI adversary. A common example is to make payloads looks like HTTP, which is rarely blocked due to its ubiquity. Protocol mimicry in the context of censorship evasion originates with the Tor project prepending HTTP headers to Tor packets. Unfortunately mimicry-based obfuscators have prohibitively poor performance. [40]

A simple but flexible approach to mimicry is Format-Transforming Encryption (FTE), a type of symmetric-key cryptography where the user specifies a *regular-expression* that the output ciphertext will conform to [47], [70]. Since DPI also use regular expressions, it can sometimes be possible to *precisely* force misclassification. However, since mimicry-based obfuscators try to mimic a cover protocol without actually using a valid implementation of it, the syntax and semantics of messages originating from mimicry systems can deviate substantially from that of messages conforming to the protocol. For example a proxy server that uses a mimicry obfuscator may produce a message that looks like a `HTTP GET` after having just received a `HTTP GET`, thereby totally violating the HTTP specification [40]. It has been shown that basic entropy-based tests as well as tests on protocol semantics are able to reliably detect FTE obfuscators [48]. Also, [49] has shown that mimicry-based obfuscation tools "completely fail to achieve unobservability" since it is infeasible to mimic all aspects of the target protocol's implementations.

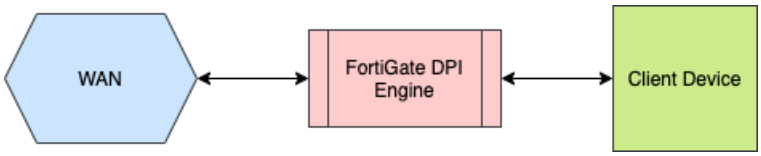

Figure 8: A FortiGate DPI engine acts as a router, sitting in-between a client device and a wider network. It is configured to block access to proxy tools and any unrecognised protocols.

5.2.3 Tunnelling

Tunnelling takes mimicry a step further by using an actual implementation of the cover protocol to create messages. Tunnelling on its own is not a new concept, many existing protocols are simply instances of one protocol being encapsulated by another. For example, HTTPS is HTTP over TLS, DoH is DNS over HTTPS, and anything over a VPN is technically being tunnelled. The difference in our case is that the tunnelling is specifically used to hide the fact that a circumvention tool is being used.

There are challenges in implementing tunnelling obfuscators in a way that preserves P–indistinguishability. For example, HTTPS is a ubiquitous protocol that is commonly used for high downstream bandwidth and low upstream bandwidth situations, whereas someone using Tor may be torrenting or uploading a large file. Also a carrier protocol may not support all the features required to allow tunnelling some other protocol. Functionality has to be achieved while ensuring suitably high performance. There could also be unintended side channels, for example it's sometimes possible to identify a specific implementation of TLS using a client fingerprinting attack. Obfuscation that implement tunnelling have to match the behaviour of common implementations *and* usage of the cover protocol [40]. Despite these challenges, tunnelling is by far the most powerful obfuscation technique out of those we have discussed [49].

5.3 Evaluation

In this experiment we have a very simple setup (fig. 8) where a client device has access to a wider network through a FortiGate router with a commercial DPI engine running on it. The DPI engine is configured to block access to proxy tools and any unrecognised protocols.

Traffic Source	**Result**	**Detection**
Tor	Blocked	`Tor`
obfs4	Blocked	`Tor`
Meek	Failed	`HTTPS, CDN`
OpenVPN (UDP)	Blocked	`OpenVPN`
WireGuard	Blocked	`WireGuard`
V2Ray	Blocked	`Unknown`

Figure 9: Detection summary for some commonly used proxies and network traffic obfuscation tools.

Tor itself without any obfuscators was blocked as expected, whereas obfs4 (randomiser) was detected *as Tor*, not *unknown*. Meek failed to connect to the Tor network but there were no explicit blocking events in the DPI logs suggesting that none of the tried CDNs support domain fronting any more. This theory is supported by the fact that the DPI logs showed successful connections to Amazon S3, Microsoft Azure and some other CDNs, but with only a few kB of data transferred. OpenVPN and WireGuard were detected correctly and blocked, as expected. Our V2Ray setup used the recommended configuration options[14] and was blocked as

[14] https://www.v2ray.com/en/welcome/start.html

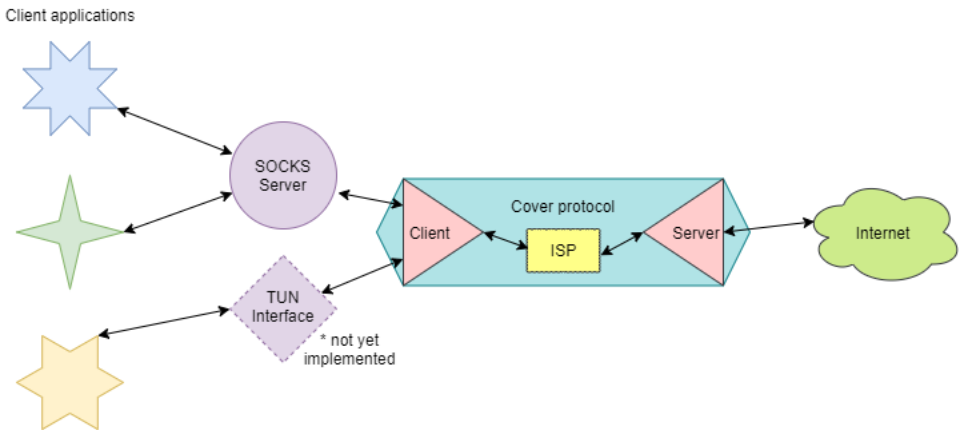

Figure 10: High-level overview of Rosen's architecture. Applications interface with a local proxy client that tunnels traffic through a cover protocol to the server.

the DPI engine did not recognise the used protocol. It's possible that a different configuration may have allowed the traffic to pass through.

6 Rosen Proxy Tunnel

Rosen[15] is my attempt at designing and implementing a censorship circumvention proxy tool[16]. An outline of its architecture is expressed in fig. 10.

6.1 Design Goals

1. **P–Indistinguishability**, **Confidentiality**, **Authenticity** and **Integrity**, as defined in section 3.
2. **Flexibility**. It should be relatively easy to add support for another cover protocol or configure the behaviour of an existing protocol to adapt to changing adversarial conditions. Censorship techniques and countermeasures are constantly evolving and so this flexibility is necessary in order to maintain functionality.
3. **Compatibility**. It should be possible to route most application traffic through the proxy.
4. **Simplicity**. The protocol should be conceptually simple and the software should be easy to use. This allows users to deploy proxy servers easily and use the software in a safe manner wile avoiding pitfalls.
5. **Performance**. There should be a low space and time overhead compared to not using the proxy.
6. **Stability**. The program should be run stably for sustained periods of time under all normal operating conditions.

6.2 Cover Protocols

Based on our design goals and discussion in previous sections, it's clear that good choices for cover protocols are those that are encrypted and ubiquitous. An encrypted protocol will typically be mostly made up of ciphertext which, due to its indistinguishability from random, can be safely replaced with our own ciphertext. In addition, many modern encrypted protocols inherently provide properties 1 to 3 from section 3. Ubiquity is desirable in order to maximise collateral damage from either an adversary attempting to block the cover protocol or falsely categorising a proportion of background traffic. Some candidate protocols include:

[15]Source code is available at https://github.com/awnumar/rosen

[16]Rosen is licensed under the BSD Zero Clause License, an unconditional, public-domain-equivalent license. The license text can be found here: https://github.com/awnumar/rosen/blob/main/LICENSE

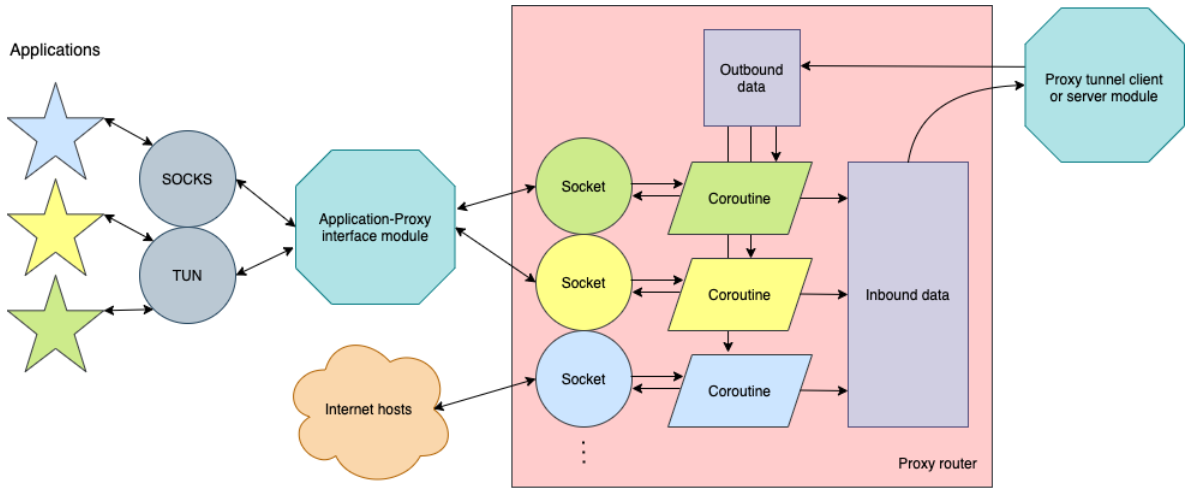

Figure 11: Rosen's message multiplexer/demultiplexer (red) handles the communication between application sockets and a proxy client module, or between Internet hosts and a proxy server module, depending on whether Rosen is acting as a proxy client or server.

1. TLS
2. HTTPS
3. WSS
4. Secure Shell (SSH)

At the time of writing only HTTPS is supported. It was chosen mainly for its ubiquity.

6.3 Architecture and Implementation

Rosen's architecture is constrained by the design goals. In particular, flexibility and compatibility necessitate modularity. A proxy client instance needs to communicate with various client applications and forward their traffic to a server instance. On the other hand server instances need to coordinate conversations between the client application and various Internet hosts. So, on either side we have data streams being multiplexed between a single channel and an arbitrary number of sockets. The *multiplexer/demultiplexer* (fig. 11) is the core component that facilitates this on both the client and the server. It aggregates messages from many external sockets into a single binary message blob, and can separate such blobs, routing each message to the correct destination. This scheme facilitates the highly-threaded architecture of Rosen where each socket has two coroutines attached to it, one for reading and another for writing.

The application-proxy interface (which handles communication between client programs and the proxy) and the proxy client and server endpoint modules (which transport the multiplexed message blobs between each other over the cover protocol) use Go's interface primitives and a standard message format to facilitate modularity. It will be easier to explain how Rosen works with an example. Suppose we have a setup where the client side starts a Socket Secure (SOCKS) server to interface with user-space applications, and the proxy endpoint modules are using HTTPS as the cover protocol. See fig. 10 for a visual guide.

Configuration. Rosen includes an easy-to use configuration wizard that guides users through the different options. An example is shown in fig. 12. The quiz is configured by the developer of the cover protocol implementation and is fully specified using code. Once a configuration file has been created, it is copied to both the client and server devices. It also contains a secret 256 bit key that the client can use to authenticate with the server.

```
$ ./rosen -configure

Which protocol do you want to use?
Choose from {https}
> https

Enter the address that the client will use to connect to the proxy server.
It must start with https://
> https://example.com

Enter the public hostname that your server will be accessible from.
This will be used for TLS certificate provisioning.
> example.com

Enter an email for LetsEncrypt registration.
This will be used when provisioning a TLS certificate.
> anon@example.com

Should the LetsEncrypt CA root certificate be pinned by the client? (yes/no)
> yes

Set the maximum TLS version that should be used, 1.2 or 1.3
> 1.3

Config file path: /mnt/c/Users/Awn/src/rosen/7PrXt6N1.json
```

Figure 12: Rosen implements an easy-to-use wizard for generating a configuration file.

The configuration wizard also asks users whether the client should pin LetsEncrypt's root certificate. There are many CAs in existence and they all exist in the exact same trust pool, meaning every single CA in a client's trust pool *must* not be malicious or else authenticity and thus confidentiality is broken. Rosen implements automatic TLS certificate issuance and renewal with LetsEncrypt so it makes sense to limit the client's trust pool to only this CA.

Application-Proxy Interface. A client opens a connection to a remote host through the proxy by connecting to the SOCKS server and passing it an address and connection type. Rosen will associate a unique identifier with this socket and attach two coroutines to it, one perpetually reads and the other writes. At this point the connection is "registered" on the client side and we are ready to inform the server about it.

Server Connection Registration. A message containing instructions to open a new connection is forwarded to the server. The message format looks like:

```
type Packet struct {
    // ID of the connection that this packet belongs to.
    ID string

    // Network address of the remote Internet host.
    Dest Endpoint

    // This packet either:
    // is an instruction to open a new connection OR
    // contains data for a connection OR
    // is an instruction to close the connection.
    Type PacketType

    Data []byte
}
```

and so an open connection instruction would be something like:

```
Packet{
    ID: "connection_identifier",
    Dest: Endpoint{
        Network: "tcp",
        Address: "2.3.5.7:443",
    },
    Type: Open,
}
```

When the server receives this message it opens a new socket and registers it in its local multiplexer with the same connection identifier. This way both client and server agree on connection mappings.

Proxying. Suppose there are a set of sockets on the client side corresponding to open connections. Whenever data arrives on any of them, it's packaged into a `Packet` object and forwarded to the multiplexer. The aggregate queue of these messages is repeatedly serialised and forwarded to client-side proxy endpoint which handles the forwarding of these messages to the server, which then demultiplexes and forwards each individual message to the right socket. The same thing happens in reverse on the server side. When a connection is closed any resources related to it are cleaned up and a notification is sent to the other proxy endpoint which then does the same. Any subsequent messages referring to non-existent connections are ignored.

With regards to the HTTPS cover protocol implementation specifically, the main challenge is handling the fact that HTTP is an asymmetric protocol where the client is able to make requests at any point in order to send data, but the server can only send data to the client inside the HTTP response. This necessitates regular polling by the client to allow the server to send back any waiting data. The current implementation waits a short, random interval of time between polls, unlike Meek which uses a regular polling pattern with a linear decay of the polling frequency if no data is transferred.

Authentication. The HTTPS cover protocol implementation uses a simple authentication mechanism: the key is included as a HTTP header inside each request. Care is taken to limit the impact of timing side channels by performing the validation of the key in constant time. This was the obvious and straightforward authentication mechanism and is similar to what other proxy tools such as HTTPT do. However perhaps a more flexible and secure method would be to use a fast authenticated symmetric-key cipher on the payload before transmitting it. This way, the server will only be able to decrypt the payload if the client encrypted and authenticated it with the correct key. Benefits of this approach include that the key material does not have to be sent alongside each message payload, there is two-way authentication instead of just client-authentication[17], and the scheme is protocol-agnostic and so it's easily expanded to other cover protocol implementations. This alternative authentication mechanism will likely be added in a later version. The key that the client and server share is automatically placed into the configuration file by the configuration wizard and so distributing this key is as simple as sharing this file.

Decoy. Currently Rosen allows a configurable decoy handler to be substituted into the proxy. This is made possible by Go's `http.Handler` that allows any function that looks like

```
func hander(w ResponseWriter, r *Request) {}
```

to handle incoming HTTP requests. Here `r` is a pointer to the full request data and `w` allows the function to write back its response. The default at the moment is a simple static file server that serves some files that are embedded into the binary at compile time. Eventually though support for a set of configurable decoy modules is planned, similar to those mentioned in section 5.1.1.

Rosen is implemented using the Go programming language[18] for a number of reasons:

- Go is relatively easy to read and write, resulting in a fast development cycle and reducing the barrier of entry for contributors.
- It has an extensive standard library that includes popular production-ready implementations of needed cover protocols such as HTTPS[19], TLS[20], SSH[21], and there exist good libraries for others such as WSS[22].

[17] Currently the only way for the client to authenticate the server is using the server's TLS certificate.
[18] https://golang.org/doc/
[19] https://golang.org/pkg/net/http
[20] https://golang.org/pkg/crypto/tls
[21] https://pkg.go.dev/golang.org/x/crypto/ssh
[22] https://github.com/gorilla/websocket

- It multiplexes many lightweight coroutines over a number of system threads, and uses channels to *share memory by communicating*[23]. This model makes it possible to have potentially thousands of independent routines that are easily synchronised.
- Many different operating systems and architectures are natively supported. Cross compilation is hassle-free and the compiler produces static binaries, making it easy to quickly deploy applications without worrying about environment dependencies.

6.4 Evaluation

P–Indistinguishability. Figure 13 shows the results of running Rosen against a set of state-of-the-art commercial and open source DPI engines. It was also tested behind the Great Firewall in China by an anonymous person who volunteered after seeing my post[24] introducing Rosen. Of course only HTTPS is evaluated here since it is currently the only supported cover protocol.

DPI Engine	Result	Detection
nDPI (Open Source)	Allowed	`HTTPS`
Palo Alto Networks	Allowed	`HTTPS`
Fortinet FortiGate	Allowed	`HTTPS`
Great Firewall (China)	Allowed	`N/A`

Figure 13: Detection summary for Rosen traffic by various different DPI engines.

Rosen has an "unfair" advantage in these tests since it is totally new and unknown to any adversaries. Ideally it should be attacked specifically by an adversary that has access to the source code. In terms of distinguishing attack vectors against Rosen, a few come to mind:

- The HTTPS cover protocol implementation has very frequent pings to the server to poll for waiting data. When there is no data this results in many short requests and responses, or otherwise they can be very large. HTTP typically has infrequent messages with bursts of activity, and download sizes are much larger than uploads. This may not at all hold true for arbitrary traffic being routed through the proxy. An alternative cover protocol such as WSS would blend in better in this regard, and using some kind of randomised padding scheme would reduce the information leaked.
- Content served on the decoy handler may not align with observed traffic patterns for users of the proxy service. For example if the decoy is a basic static site with not much data, and users connecting to the proxy are exchanging disproportionately large volumes of data with the server, this may cause suspicion. For this reason the decoy handler should be (ideally) a legitimate, high-traffic volume service with many users, so that large volumes of exchanged proxy data blend in.
- TLS client fingerprinting attacks may be able to specifically identify Rosen based on the contents of the `Client Hello` message, assuming ECH is not used. However, the TLS implementation used is that of the Go standard library and care is taken to stick to defaults wherever possible. For this reason it seems likely that an adversary would only be able to narrow the implementation down to Go, but not to Rosen specifically, using this method. This may be good enough against some censors but not others, depending on their tolerance for collateral damage. For example, Iran is known to care very little about false positives. There exist tools to extract TLS fingerprints from traffic dumps[25], and there are libraries that can allow us to mimic and randomise these fingerprints[26].

[23] https://golang.org/doc/codewalk/sharemem
[24] https://spacetime.dev/rosen-censorship-resistant-proxy-tunnel
[25] https://tlsfingerprint.io/pcap
[26] https://github.com/refraction-networking/utls

It's not currently clear how many of these can currently be feasibly targeted or how expensive such attacks would be. It's also not clear what tolerances real-world censors have with respect to collateral damage et al. Targeted academic testing of Rosen would be a good start to better understand its resistance to distinguishing attacks. In particular, finding out the false-positive rate of the best attack is extremely valuable information.

Confidentiality, Authenticity and Integrity. Rosen's implementation of the HTTPS cover protocol relies on TLS to provide these security properties. The main attack vector here is in validating the authenticity of the server. If an attacker manages to get access to a valid certificate for the domain being used they will be able to perform a MITM attack. This is unlikely in general and can be made even less likely if the user enables the pinning of LetsEncrypt's root certificate. However, supposing an attack does happen, the client will send the adversary a request containing the authentication key that the client and server share, and then the attacker will be able to make their own request to the server with this key to confirm that it's acting as a proxy, which breaks P–indistinguishability. In addition, if the adversary simply monitors the proxy session, they will be able to see the client's traffic data and maybe modify it if it's not itself encrypted.

As discussed in section 6.3, a partial solution is to use an authenticated symmetric encryption scheme to protect every payload that's sent between the proxy endpoints. This provides strong authenticity guarantees for *both* the client and server and avoids sending the key over the network. If an MITM attack happens, confidentiality and authenticity are preserved, although integrity may not be due to replay attacks. P–indistinguishability may also be broken in this case.

Flexibility. While Rosen's design was heavily guided by this requirement, it's difficult to evaluate at the moment whether it succeeded at implementing modularity *well* since there is only a single cover protocol implemented. There is an implementation of WSS in progress so it will be easier to answer this question once that is complete.

Compatibility. Rosen implements a SOCKS5 server to allow client applications access to the proxy tunnel. SOCKS is a very popular protocol for this purpose and is supported by many applications. TUN support is planned for the future to allow users to easily route *all* application traffic through the proxy server, including traffic from applications that do not support SOCKS. There are tools that allow the creation of a virtual network interface that forwards all traffic through a SOCKS server[27].

Simplicity. Rosen has been designed to be very simple and easy to configure, use, deploy, and develop. A few design choices exemplify this:

- Go produces static binaries that eliminate dependencies on any system libraries or other environment specifics except the kernel and architecture, and cross-compilation is trivial. This allows extremely fast and simple deployment. The HTTPS cover protocol's decoy handler uses Go's `embed` directive to embed the static site content *into the binary itself*, further simplifying the deployment process.
- Configuration, including configuring the configuration wizard itself, is very easy. The wizard explains options and what they mean, checks the inputted values for errors, generates a cryptographically-secure 32 byte key, and creates the configuration file.
- Commands to run Rosen are very simple. There are only a few flags, one to launch the configuration wizard, a *mode* flag that launches Rosen in either client or server mode, a *config* flag that specifies the configuration file, and an option to change the port that the SOCKS server runs on.
- Go is a simple language with a limited set of primitives that compose well. It is easy to read and write, simplifying the process for a new contributor to understand and expand the codebase.
- Care is taken to make relatively complex tasks as painless as possible. For example, issuance and renewal of TLS certs is *completely automated*, as long as the server is accessible from the domain name specified in the configuration file.

[27] https://github.com/ambrop72/badvpn/wiki/Tun2socks

Performance. Figure 14 shows Rosen's HTTPS cover protocol implementation's bandwidth and latency characteristics versus a direct connection.

Proxy	**Avg. Latency**	**Avg. Download Speed**	**Avg. Upload Speed**
None	`19.7 ms`	`367.6 Mbps`	`36.73 Mbps`
Rosen HTTPS	`47.7 ms`	`196.6 Mbps`	`20.5 Mbps`
Difference (%)	`242.1` %	`53.5` %	`55.8` %

Figure 14: Summary of testing Rosen's HTTPS cover protocol implementation's bandwidth and latency characteristics versus a direct connection. Results are the average of three tests, collected using `speedtest.net`

Overall these results aren't terrible but I am not totally happy with them. Work is needed to understand what bottlenecks there are, where they are, and how they can be reduced. Other cover protocols may perform significantly better due to HTTPS's inherent drawbacks in terms of its asymmetric nature. There is also sometimes an issue with testing upload speeds where the results are erratic, vary wildly, and sometimes are `0 Mbps`. This may be an artefact of the way speed testing websites implement tests, or an issue with Rosen. The issue appears less frequently on `speedtest.net` if you set it to test using only a single stream instead of multiple.

Stability. At the time of writing there is currently an intermittent bug in Rosen's HTTPS cover protocol implementation that causes the server to crash. This is a critical issue and I am working to diagnose the root cause. This issue has been reported by an anonymous tester, and I have managed to reproduce it myself in some circumstances.

7 Future Work

- TUN support will allow users to route *all* traffic through the proxy tunnel, including traffic belonging to applications that do not support SOCKS.
- Implement the other cover protocols mentioned in section 6.2, starting with WSS as it is closely related to HTTPS and has much more appealing typical traffic and performance characteristics.
- Testing of bottlenecks affecting Rosen's latency and bandwidth will help identify how to improve these features.
- Find and fix the root cause of server crashes.
- Testing of Rosen by many people over a longer time is essential to understanding its effectiveness. Also, academic analysis of distinguishing attacks against it are necessary, with a focus on the false-positive rate of any attacks.
- Better understanding of censors, their capabilities, and tolerances for collateral damage is necessary to inform optimal trade-offs decisions between P–indistinguishability and usability.
- Support multiple clients per server. Potentially support multiple protocols per server.
- Implement the alternative authentication mechanism discussed in sections 6.3 and 6.4.
- Develop a set of configurable, modular, and reusable decoy handlers.

8 Conclusion

The goal of this project was to model adversarial conditions relevant to anti-censorship systems, explore the existing body of work in this field, and use this knowledge to design, develop, test and evaluate a modular proxy framework.

We started off by introducing the importance of information, how the Internet works and the protocols that it relies on, vulnerabilities in these protocols, how they can be exploited, and some ways that these exploits can be mitigated. We go over some historical and modern-day examples of Internet exploitation by powerful actors that use it to suppress human rights and implement totalitarian measures like mass surveillance and censorship.

A formal but not rigorous adversarial model is introduced and used to reason about adversarial capabilities and trade-offs. Some security properties are also introduced. In particular we define P–indistinguishability with respect to different adversarial capabilities. Various methods of identifying and fingerprinting traffic are explained, as well as different techniques to obfuscate this information. A number of existing tools are explored and then tested against a state-of-the-art commercial DPI engine. None of them were able to bypass the DPI engine and successfully open a connection.

Rosen is introduced and explained, including its main goals, key features, design choices, and it's then tested in terms of its P–indistinguishability and performance, and informally evaluated with respect to its other goals. It successfully manages to bypass every DPI engine it was tested against, and successfully opened a connection from behind the Great Firewall in China. Finally we discuss possible future work.

9 Acknowledgements

This project would not exist in its current form without the help of my project supervisor, Dr Daniel Page[28], whose experience and guidance I relied on when researching, planning, and writing this report. Also, the knowledge and technical experience that I received from James Webb[29] was incredibly helpful, and his support made possible the testing of Rosen and other censorship circumvention tools against state-of-the-art commercial DPI engines. I'm very grateful for their time and effort.

Special thanks goes to the anonymous Chinese user who graciously offered to test Rosen despite the potential danger to themselves. I am very grateful for their help.

[28] Senior Lecturer in Computer Science at the University of Bristol; https://github.com/danpage
[29] CEO at Ultra Horizon & Research Associate at the University of Bristol; https://github.com/jwsi

10 Appendix

10.1 Acronyms

BGP Border Gateway Protocol

BGPsec BGP Security

CA Certificate Authority

CAA DNS Certification Authority Authorization

CDN Content Delivery Network

CSP Content Service Provider

DANE DNS-based Authentication of Named Entities

DNS Domain Name System

DNSSEC DNS Security Extensions

DoH DNS over HTTPS

DoS Denial-of-Service

DoT DNS over TLS

DPI Deep Packet Inspection

ECH Encrypted Client Hello

ESNI Encrypted SNI

FTE Format-Transforming Encryption

HSTS HTTP Strict Transport Security

HTTP Hypertext Transfer Protocol

HTTPS HTTP Secure

IoT Internet of Things

ICMP Internet Control Message Protocol

IP Internet Protocol

ISP Internet Service Provider

MITM Man In The Middle

OSI Open Systems Interconnection

P2P Peer-to-Peer

SNI Server Name Indication

SOCKS Socket Secure

SPI Shallow Packet Inspection

SSH Secure Shell

TCP Transmission Control Protocol

TLD Top-Level Domain

TLS Transport Layer Security

TOFU Trust On First Use

URL Uniform Resource Locator

VPN Virtual Private Network

WSS WebSocket Secure

References

[1] McKinsey Global Institute. (2011). "The great transformer: The impact of the Internet on economic growth and prosperity," [Online]. Available: https://web.archive.org/web/20200925101447/https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-great-transformer.

[2] Electronic Frontier Foundation. (2019). "Behind the One-Way Mirror: A Deep Dive Into the Technology of Corporate Surveillance," [Online]. Available: https://web.archive.org/web/20191229031106/https://www.eff.org/files/2019/12/11/behind_the_one-way_mirror-a_deep_dive_into_the_technology_of_corporate_surveillance.pdf.

[3] C. Anderson, *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion, 2006, ISBN: 1401302378.

[4] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019. DOI: 10.1109/COMST.2019.2910750.

[5] P. Mockapetris, "Domain Names - Implementation And Specification," Information Sciences Institute, RFC 1035, 1987. [Online]. Available: https://tools.ietf.org/html/rfc1035.

[6] Information Sciences Institute, "Internet Protocol," University of Southern California, RFC 791, 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.

[7] Y. Xu. (2016). "Internet Censorship Around the World," [Online]. Available: https://web.archive.org/web/20201112021544/https://blog.thousandeyes.com/internet-censorship-around-the-world/.

[8] Wired. (2008). "ISPs' Error Page Ads Let Hackers Hijack Entire Web, Researcher Discloses," [Online]. Available: https://web.archive.org/web/20201112011927/https://www.wired.com/2008/04/isps-error-page/.

[9] T. Ptacek. (2015). "Against DNSSEC," [Online]. Available: https://web.archive.org/web/20201111232023/https://sockpuppet.org/blog/2015/01/15/against-dnssec/.

[10] R. White. (2018). "A Look at the Current State of DNSSEC in the Wild," [Online]. Available: https://web.archive.org/web/20201109031933/http://www.circleid.com/posts/20180906_a_look_at_current_state_of_dnssec_in_the_wild/.

[11] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Mozilla, RFC 8446, 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446.

[12] A. Langley. (2015). "Why not DANE in browsers," [Online]. Available: https://web.archive.org/web/20201118011206/https://www.imperialviolet.org/2015/01/17/notdane.html.

[13] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, "Hypertext Transfer Protocol – HTTP/1.0," Massachusetts Institute of Technology and University of California Irvine, RFC 1945, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1945.

[14] E. Rescorla, K. Oku, N. Sullivan, and C. Wood, "TLS Encrypted Client Hello," RTFM, Inc., Fastly, and Cloudflare, RFC, 2020. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-esni-07.

[15] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol (BGP)," Cisco Systems and IBM Corp., RFC 1105, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1105.

[16] The Washington Post. (2015). "Quick fix for an early Internet problem lives on a quarter-century later," [Online]. Available: https://web.archive.org/web/20201116095219/https://www.washingtonpost.com/sf/business/2015/05/31/net-of-insecurity-part-2/.

[17] G. Orwell, *Nineteen Eighty-Four*. Secker & Warburg, 1949, ISBN: 0436350076.

[18] H. A. Giroux, "Totalitarian Paranoia in the Post-Orwellian Surveillance State," *Cultural Studies*, vol. 29, no. 2, pp. 108–140, 2015. DOI: 10.1080/09502386.2014.917118.

[19] W. Churchill and J. V. Wall, "The COINTELPRO Papers," *Boston: South End*, 1990. [Online]. Available: https://web.archive.org/web/20190319071513/http://chinhnghia.com/Cointelpro_Papers.pdf.

[20] P. Caban, "Cointelpro," 2005. [Online]. Available: https://web.archive.org/web/20190816204815/https://scholarsarchive.library.albany.edu/cgi/viewcontent.cgi?article=1017&context=lacs_fac_scholar.

[21] (2013). "Snowden Digital Surveillance Archive," [Online]. Available: https://snowdenarchive.cjfe.org/greenstone/cgi-bin/library.cgi.

[22] The Guardian. (2013). "NSA Files: Decoded," [Online]. Available: https://web.archive.org/web/20201124172600if_/https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded.

[23] Business Insider. (2016). "This is everything Edward Snowden revealed in one year of unprecedented top-secret leaks," [Online]. Available: https://web.archive.org/web/20171101204723/http://www.businessinsider.com/snowden-leaks-timeline-2016-9.

[24] The Guardian. (2013). "Revealed: how US and UK spy agencies defeat internet privacy and security," [Online]. Available: https://web.archive.org/web/20201201113028/https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security.

[25] The Guardian. (2013). "GCHQ taps fibre-optic cables for secret access to world's communications," [Online]. Available: https://web.archive.org/web/20201201113036/https://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa.

[26] The Guardian. (2013). "GCHQ and European spy agencies worked together on mass surveillance," [Online]. Available: https://web.archive.org/web/20201112000122/https://www.theguardian.com/uk-news/2013/nov/01/gchq-europe-spy-agencies-mass-surveillance-snowden.

[27] X. Pan, "Hunt by the Crowd: an exploratory qualitative analysis on cyber surveillance in China," *Global Media Journal*, vol. 9, no. 16, pp. 1–19, 2010.

[28] UK Government. (2016). "Investigatory Powers Act 2016," [Online]. Available: https://www.legislation.gov.uk/ukpga/2016/25/contents.

[29] C. Grothoff and J. Porup, "The NSA's SKYNET program may be killing thousands of innocent people," *Ars Technica*, 2016. [Online]. Available: https://hal.inria.fr/hal-01278193.

[30] R. Clayton, "Anonymity and traceability in cyberspace," 2005. [Online]. Available: https://web.archive.org/web/20170808091542/http://www.cl.cam.ac.uk/~rnc1/thesis.pdf.

[31] OpenNet Initiative. (2012). "China," [Online]. Available: https://web.archive.org/web/20170713134307/http://access.opennet.net/wp-content/uploads/2011/12/accesscontested-china.pdf.

[32] The Chinese Government. (2010). "The Internet in China," [Online]. Available: https://web.archive.org/web/20130408094606/http://english.gov.cn/2010-06/08/content_1622956_7.htm.

[33] Amnesty International. (2000). "China: The crackdown on falun gong and other so-called "heretical organizations"," [Online]. Available: https://www.refworld.org/docid/3b83b6e00.html.

[34] X. Xu, Z. M. Mao, and J. A. Halderman, "Internet censorship in china: Where does the filtering occur?" In *Passive and Active Measurement*, N. Spring and G. F. Riley, Eds., Springer Berlin Heidelberg, 2011, pp. 133–142, ISBN: 978-3-642-19260-9.

[35] OpenNet Initiative. (2013). "Country Profiles," [Online]. Available: https://web.archive.org/web/20200803220315/https://opennet.net/country-profiles.

[36] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in iran: A first look," in *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*, Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/aryan.

[37] P. Rogaway, *The moral character of cryptographic work*, Cryptology ePrint Archive, Report 2015/1162, https://eprint.iacr.org/2015/1162, 2015.

[38] S. Fish *et al.*, *Save the world on your own time*. OUP USA, 2008.

[39] K. Hafner and J. Markoff, *Cyberpunk: outlaws and hackers on the computer frontier, revised*. Simon and Schuster, 1995.

[40] L. Dixon, T. Ristenpart, and T. Shrimpton, "Network traffic obfuscation and automated internet censorship," *IEEE Security & Privacy*, vol. 14, no. 6, pp. 43–53, 2016. DOI: 10.1109/MSP.2016.121.

[41] R. Clayton, S. J. Murdoch, and R. N. M. Watson, "Ignoring the great firewall of china," in *Privacy Enhancing Technologies, Lecture Notes in Computer Science*, vol. 4258, Springer, 2006, pp. 20–35.

[42] J. Cheng, Y. Li, C. Huang, A. Yu, and T. Zhang, "Acer: Detecting shadowsocks server based on active probe technology," *Journal of Computer Virology and Hacking Techniques*, vol. 16, Sep. 2020. DOI: 10.1007/s11416-020-00353-z.

[43] S. Frolov and E. Wustrow, "HTTPT: A probe-resistant proxy," in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/foci20/presentation/frolov.

[44] U. Maurer, "Indistinguishability of random systems," in *Advances in Cryptology — EUROCRYPT 2002*, L. R. Knudsen, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 110–132, ISBN: 978-3-540-46035-0.

[45] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: Deniable encryption, and more," in *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '14, New York, New York: Association for Computing Machinery, 2014, 475–484, ISBN: 9781450327107. DOI: 10.1145/2591796.2591825. [Online]. Available: https://doi.org/10.1145/2591796.2591825.

[46] Y. He, L. Hu, and R. Gao, "Detection of tor traffic hiding under obfs4 protocol based on two-level filtering," in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, 2019, pp. 195–200. DOI: 10.1109/ICDIS.2019.00036.

[47] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," ser. CCS '13, Association for Computing Machinery, 2013, 61–72, ISBN: 9781450324779. DOI: 10.1145/2508859.2516657.

[48] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, Denver, Colorado, USA: Association for Computing Machinery, 2015, 57–69, ISBN: 9781450338325. DOI: 10.1145/2810103.2813715. [Online]. Available: https://doi.org/10.1145/2810103.2813715.

[49] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," *IEEE Symposium on Security and Privacy*, 2013.

[50] I. Fette and A. Melnikov, "The WebSocket Protocol," Google Inc. and Isode Ltd., RFC 6455, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455.

[51] Z. Ling, J. Luo, W. Yu, M. Yang, and X. Fu, "Extensive analysis and large-scale empirical evaluation of tor bridge discovery," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 2381–2389. DOI: 10.1109/INFCOM.2012.6195627.

[52] S. Mrdovic and B. Perunicic, "Kerckhoffs' principle for intrusion detection," in *Networks 2008 - The 13th International Telecommunications Network Strategy and Planning Symposium*, vol. Supplement, 2008, pp. 1–8. DOI: 10.1109/NETWKS.2008.6231360.

[53] Z. Chai, A. Ghafari, and A. Houmansadr, "On the importance of encrypted-sni (ESNI) to censorship circumvention," in *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: https://www.usenix.org/conference/foci19/presentation/chai.

[54] F. Andreasen, N. Cam-Winget, and E. Wang, "TLS 1.3 Impact on Network-Based Security," Cisco Systems, Internet-Draft, 2017. [Online]. Available: https://tools.ietf.org/id/draft-camwinget-tls-use-cases-00.html.

[55] Tor Project Blog. (2021). "Domain shadowing: Leveraging cdns for robust blocking-resistant communications," [Online]. Available: https://blog.torproject.org/anti-censorship-domain-shadowing.

[56] M. Wei, "Domain shadowing: Leveraging content delivery networks for robust blocking-resistant communications," in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/wei.

[57] S. Sheffey and F. Aderholdt, "Improving meek with adversarial techniques," in *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: https://www.usenix.org/conference/foci19/presentation/sheffey.

[58] M. Husák, M. Cermák, T. Jirsík, and P. Celeda, "Network-based https client identification using ssl/tls fingerprinting," in *2015 10th International Conference on Availability, Reliability and Security*, 2015, pp. 389–396. DOI: 10.1109/ARES.2015.35.

[59] "How the great firewall of china is blocking tor," in *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*, Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/foci12/workshop-program/presentation/Winter.

[60] R. T. El-Maghraby, N. M. Abd Elazim, and A. M. Bahaa-Eldin, "A survey on deep packet inspection," in *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, 2017, pp. 188–197. DOI: 10.1109/ICCES.2017.8275301.

[61] P.-C. Lin, Y.-D. Lin, Y.-C. Lai, and T.-H. Lee, "Using string matching for deep packet inspection," *Computer*, vol. 41, no. 4, pp. 23–28, 2008. DOI: 10.1109/MC.2008.138.

[62] R. Bendrath and M. Mueller, "The end of the net as we know it? deep packet inspection and internet governance," *New Media & Society*, vol. 13, no. 7, pp. 1142–1160, 2011. DOI: 10.1177/1461444811398031.

[63] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "Ndpi: Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2014, pp. 617–622. DOI: 10.1109/IWCMC.2014.6906427.

[64] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," 2004. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA465464.

[65] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Commun. ACM*, vol. 42, no. 2, 39–41, Feb. 1999, ISSN: 0001-0782. DOI: 10.1145/293411.293443. [Online]. Available: https://doi.org/10.1145/293411.293443.

[66] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network," 2014. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA599695.

[67] Y. Shi and K. Matsuura, "Fingerprinting attack on the tor anonymity system," in *Information and Communications Security*, S. Qing, C. J. Mitchell, and G. Wang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 425–438, ISBN: 978-3-642-11145-7.

[68] D. Arp, F. Yamaguchi, and K. Rieck, "Torben: A practical side-channel attack for deanonymizing tor communication," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '15, Singapore, Republic of Singapore: Association for Computing Machinery, 2015, 597–602, ISBN: 9781450332453. DOI: 10.1145/2714576.2714627. [Online]. Available: https://doi.org/10.1145/2714576.2714627.

[69] S. Frolov, J. Wampler, and E. Wustrow, "Detecting probe-resistant proxies," Jan. 2020. DOI: 10.14722/ndss.2020.23087.

[70] D. Luchaup, K. P. Dyer, S. Jha, T. Ristenpart, and T. Shrimpton, "Libfte: A toolkit for constructing practical, format-abiding encryption schemes," in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA: USENIX Association, Aug. 2014, pp. 877–891, ISBN: 978-1-931971-15-7. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/luchaup.