Fall 2019

CovertNet: Circumventing Web Surveillance Using Covert Channels

Will C. Burghard
Bard College, wb6289@bard.edu

COVERTNET: CIRCUMVENTING WEB CENSORSHIP USING COVERT CHANNELS

Senior Project submitted to

The Division of Science, Mathematics and Computing

by

Will Burghard

Annandale-on-Hudson, New York

December 2019

ACKNOWLEDGMENTS

Thanks to all professors in the Computer Science department, including my advisor Robert McGrail, for making this project possible and for an enriching experience at Bard.

TABLE OF CONTENTS

Introduction

Web censorship is a persistent problem in many countries across the world. Because the Internet has been perhaps the most groundbreaking tool invented by man, facilitating the exchange of knowledge and communication on an unprecedented scale, many people, myself included, feel that access to this tool should be a basic human right. Because it has changed the lives of so many, it should be accessible to every human being throughout the world. Unfortunately, in places like the Middle East and China, the Internet is heavily censored, and major websites such as Google and Wikipedia are not accessible. In particular, the program of censorship in China is perhaps the most stringent in the world, and has been dubbed the "Great Firewall of China."

There have been many techniques developed to circumvent web censorship and surveillance, such as proxies and VPN's. In this paper, I propose a new method of web censorship circumvention which has several benefits over other methods such as VPN's and proxies. The method is based on a concept called a covert channel. A covert channel is essentially a means by which data can be transmitted using a mechanism not intended for the transfer of that particular data. The protocol I propose utilizes a covert channel based on the inner workings of network protocols. This covert channel was developed specifically to operate under the Great Firewall of China. In my protocol, data is encrypted and sent in the body of network packets to a server which has access to censored content. The data is disassembled and decrypted by the server, and the request is made for the censored content. When the response is received, the same covert channel is used to bundle the data back into network packets and is sent back to the user, where it is decrypted and parsed to reveal censored content.

I was able to implement this protocol with a good degree of Internet functionality. Although not all websites are accessible yet, the functionality reached so far is promising, and provides several websites that are censored in the Great Firewall. This indicates that the future of covert channel use in circumventing web censorship is bright and may offer a new, more developed mode of web censorship circumvention.

WEB CENSORSHIP

Over the course of the Internet's near-30 year history, it has changed the world perhaps more quickly and on a greater scale than any other technology in human history. Never before has information been as readily available as it is today, nor has communication been easier. However, the Internet is not as accessible for many across the world as it is for those of us in the United States. Web surveillance has been around since the Internet's infancy in the 1990s. In the summer of 2012, the Internet Society conducted a survey of more than 10,000 internet users across the world regarding their opinions on the Internet and Internet censorship [14]. Of those surveyed 83% agreed that "the internet should be a basic human right," and 86% agreed that "Freedom of expression should be guaranteed on the internet." However, 82% agreed that "The Internet should be governed in some form to protect the community from harm", and 71% agreed that "Censorship should exist in some form on the internet."

Faris and Villeneuve [13] provide a comprehensive list obtained by the OpenNet initiative of states that are suspected to be involved in some of internet filtering, as well as states suspected of internet filtering and those who are proven not to filter. Of these states, the overwhelming majority are in Asia, with a few in Africa as well. The Open Initiative also found in 2010 [14] that the Middle East was extensively using Western technologies such as McAfee SmartFilter for their state-wide filtering systems. Especially filtered in the Middle East are sites pertaining to sexual content such as pornography, nudity, and LGBT content. The ONI has also measured states based on their level of content filtering in three different areas: Social, political, and conflict/security.[15] 13 states had high levels of filtering in one or more areas, while 34 states had moderate filtering in one or more areas. China was the only state to have high levels of filtering in all 3 areas.

Riley [17] discusses the history of Internet censorship and its evolution as a "necessary evil." He talks about how in the early days of the Internet, it operated very smoothly without the need for the intervention of the law to govern it. The originators and early adopters of the Internet were effective governors of the content that flowed throughout the web. However, as the Internet grew in scope both as a medium of communication and disseminating information and as the largest global marketplace in history, it outgrew the capabilities of its originators to effectively govern. In a sense, the Internet became such a powerful and far-reaching technology that the people who built it were not prepared to regulate it. The elements that made the Internet so useful also made it exploitable. Hackers became more insidious and grew in number, as people became more familiar with the technology, and the increase in Internet users gave insidious adversaries more opportunities to exploit the masses. In addition to hackers, the Internet began to be adopted by criminals as a means of covertly conducting operations. A large marketplace for illegal products such as drugs and weapons, as well as outlawed content such as child pornography (as well as ordinary pornography which remains illegal in many places across the world) emerged, taking advantage of technologies that allowed criminals to hide communications from authorities.

Simultaneously, the technologies used by entities such as governments and ISP's have developed and become more widely used. In the days of the Internet's infancy, censorship efforts were seen as futile and easily circumvented. Internet pioneer John Gilmore famously said, "The Net interprets censorship as damage and routes around it."[18] Since then, censorship technologies

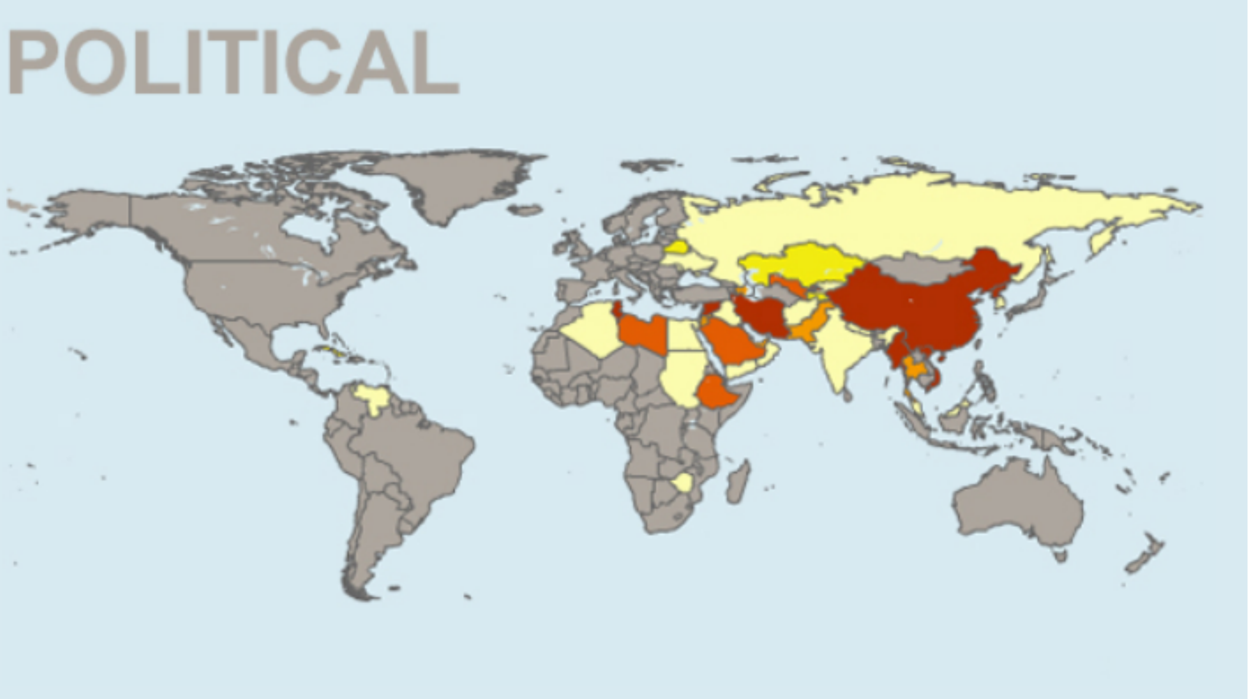


Figure 1 - Political censorship on the Internet across the world, from the OpenNet Initiative [14]. Darker countries indicate higher levels of censorship.

have become more and more advanced. Deep packet inspection technologies have grown in popularity, and other methodologies such as IP address blocking and DNS spoofing have become more sophisticated[19].

Riley discusses these topics in the context of Internet policy. Internet policy refers to the governance practices which have directed the Internet since the downfall of its self-governance, although the people who use the Internet have had as much to do with the enforcement of Internet policy as governments have. Internet policy has several goals in mind: freedom of speech, privacy, security, economic growth, and social order. The development of Internet policy has varied across the globe as a balancing act between these goals, and different nations have had different ideas about the relative importance of each of these goals. The extent to which these goals are mutually exclusive is also up for debate.

Freedom of speech and privacy are the most intimately linked of the goals. Privacy and anonymity are what attracts many to the Internet who seek a place to express their views without fear of identification or persecution. Censorship appears diametrically opposed to the first goal, which is freedom of speech. However, it is intrinsically linked to the fifth goal, which is social order. This might seem to be a contradiction, but in reality is more of a balancing act. Most people would agree that some form of censorship is necessary to uphold social order, especially when harm to others is involved, such as is the case with child pornography for example. However, the idea of social order in the minds of many governments goes beyond regulating illicit content, and into the questionable arena of upholding "moral order." In some places, particularly the Middle East, this has more religious implications, and in other, such as China, the motivations are more political in nature.

In the United States, Internet censorship is a highly contested subject. Today, the Internet in the U.S. is regulated largely by legislation, with very little technical interference.[21] The first calls for the implementation of heavier censorship on the web came in the 1990's, when explicit sexual content first began to appear in large quantities, and people became concerned about the ease of access that minors had to a plethora of adult content. Since then, a great deal of legislation has been passed attempting to limit the degree to which sexually explicit material can be disseminated to minors on the Internet, with limited success. Where legislation has had more success is in regulating the use of intellectual property on the Internet. Content which interferes with copyright laws is heavily filtered. One of the most controversial areas of Internet policy in the United States has been surveillance. Since the early 2000's, government wiretaps are reported to include major Internet connection points, and the Bush administration pushed for legislation which would force Internet providers to provide the government with wiretap access to communication networks. Since then, the evidence only suggests that surveillance in the U.S. has increased, particularly with increasingly sophisticated smartphone technology. In 2013, Edward Snowden was responsible for leaking NSA information which indicated that several mass surveillance programs were in place, some in cooperation with telecommunications companies. This subsequently sparked nationwide backlash and an ongoing debate about the ethics of internet surveillance. Despite heavy surveillance, technical censorship of the Internet is low in the U.S., and legal efforts to censor the Internet have been met with considerable backlash.

On the other hand, countries in the Middle East, such as Saudi Arabia, have some of the highest levels of Internet censorship in the world. Saudi Arabia has been aggressively censoring content since 2001. All of its content is directed to a proxy, which filters content of a number of different categories. The OpenNet Initiative published a report on its findings of Internet censorship in 2004[20], and have published updates on the status of censorship since then, which has only become more pervasive. The most heavily censored content pertained to pornography, drug use, gambling, and muslim conversion. Interestingly enough, topics that were less censored included LGBT topics, alcohol use, and most major religions. Saudi Arabia is considerably more open about its use of censorship than other countries with such pervasive censorship, and shares information regarding censored content on the website of the ISU (Internet Services Unit).

THE GREAT FIREWALL OF CHINA

Although the Middle East is heavily censored, the largest censorship system in the world can be found further East, in China. The Chinese system of Internet censorship is so vast that it has been dubbed the "Great Firewall of China." The aforementioned OpenNet Initiative found that China was the only nation in the world that had the highest levels of surveillance in all 3 categories - social, political, and conflict/security. China's censorship program targets a vast portion of the Internet, including popular social media sites such as Facebook and Twitter, knowledge-sharing websites such as Google and Wikipedia, and websites covering a wide range of topics that the Chinese government deems harmful.[21] These topics include religious movements such as Falun Gong and Buddhism, police brutality, freedom of speech and expression, and criticisms of the Chinese government. One of the most heavily filtered topics is the 1989 Tiananmen Square Protests.[22] Little to no information on the protests is available on the Chinese Internet, and on the 20th and 30th anniversaries of the protest, in 2009 and 2019 respectively, large portions of the Internet were taken down.

Back in 2002, Zittrain and Edelman [22] found that out of 204,012 distinctly requested websites, more than 18,000 websites were inaccessible from at least two proxy servers in China on two distinct days (a measure of intentional blockage vs. inaccessibility due to glitches or internet access). Since then, the level of censorship in China has only increased, although their methods of censorship have changed significantly. Internet censorship has particularly increased under the rule of president Xi Jinping, who took office in 2012.[24] In 2015, the Great Firewall received a significant upgrade, which decreased the availability of VPN protocols used to circumvent Internet censorship.

TECHNIQUES

China has a wide variety of different tools at its disposal for surveilling and censoring its Internet traffic. The usage of each of these methods has fluctuated over time, particularly within the last 10 years. This is largely due to the monumental rise in Internet users in the past 20 years. Internet usage in China increase from 1.8% in 2000, to 34.4% in 2010, and up to 54.3% in 2017[25]. Today, China has more Internet users than any country in the world, with over 748 million users according to the World Atlas [26]. With these figures in mind, filtering Internet traffic has become an immense task, and the Great Firewall has had to abandon certain techniques in favor of others to keep up with the meteoric rise in traffic.

Because China does not publish any information about the techniques it uses for censoring the Internet, most of the information we have about the subject comes from experimentation and conjecture. In general, the basis for the Great Firewall is conjectured to be the use of an IDS system which uses intermediary devices scattered throughout the network, most likely connected to preexisting routers in the network.[30] This is in contrast to many other firewalls which direct all traffic through a proxy server to be inspected. This method would be practically impossible considering the amount of Internet traffic coming from China on a daily basis. In fact, legislation that required ISP's to perform filtering of network traffic for child pornography via the use of proxies was struct down in 2004 as being "unconstitutional."[31] Rather than the law violating the ideals of our founding fathers, the reality was that the system was simply too expensive to implement, and affected the speed of benign network traffic. You can imagine that if the system was too hard to implement in a state with nearly 10 million Internet users[31], it would be downright impossible in a nation with almost 80 times that number of users. Instead, the IDS system that the Great Firewall uses most likely consists of a number of different machines hooked up to different routers across the network, which can read crossing traffic, but do not interfere with the flow itself. These machines inspect all traffic for certain banned keywords. If a packet inspection device detects a banned keyword, it uses one of a number of different methods to kill the connection it belongs to.

To inspect packets traveling through a network, China uses deep packet inspection. [35] A packet traveling through a network will generally be an amalgamation of the packets of the three main protocols in the IP protocol suite, IP, TCP, and HTTP. Deep packet inspection can look at the entire amalgamated packet and look for suspicious fields in any of the protocols. This includes banned IP address, TCP ports, and keywords in HTTP headers. It appears that in recent years, China generally only inspects the first HTTP GET request in a given connection, and does not inspect the requests that come after (such as POST or PUT), nor does it generally inspect HTTP responses from servers. [28] In fact, in recent years, the level of overall HTTP packet inspection by the GFW has decreased dramatically, with the advent of HTTPS which encrypts HTTP headers, making them unreadable to a censor.[30]

One of the most heavily used techniques that is used to bar Chinese citizens from accessing certain website is network black-holing.[27] In this technique, packets with IP addresses that fall within a certain IP range are dropped by the network. This means that the packets, upon reaching a router that performs filtering, are not forwarded to their intended address, and the source is not informed that the packets have not reached their destination. This process is known as null routing, and is the same behavior as one would expect from sending packets to an IP address that is not assigned to any machine or is assigned to a machine that is no longer in use. The Great Firewall performs null routing within the Border Gateway Protocol (BGW), which is an application-layer protocol that allows the purveyors of a network to assign routing information and determine routing paths at given points on a network. Importantly, this technique blocks only outbound traffic leaving China, and not inbound traffic coming into China; however, this is enough to stop most connections from starting in the first place. This is one of the easiest and most lightweight of the techniques deployed by the Great Firewall. It is implemented at a low cost of implementation, with little involvement from ISP's (Internet Service Providers), and with little effect to traffic speeds. However, the fact that it does not block incoming traffic means that there are workarounds. First and foremost, an IP address can switch its IP address while retaining the same domain name, although it must do so constantly as the GFW catches up to its new addresses. In addition, the fact that no incoming traffic is blocked means that one can easily access censored sites from a proxy outside of China - which is precisely the service CovertNet sets out to offer.

Another method which is used today by the Great Firewall is DNS spoofing. To understand DNS spoofing, one must first understand the DNS (Domain Name System) protocol. The DNS protocol is essentially a translation system for assigning human-readable URL's to IP addresses. A DNS server translates a human-readable URL, such as https://www.google.com, into an IP address. DNS spoofing is the technique of injecting fake IP address into the DNS system in response to a sensitive URL. For example, a Chinese censor will see https://www.wikipedia.org coming from a user and deliver a fake IP address to the user in response. In reality, Chinese DNS spoofing does not discard the real response of a DNS server, but the fake DNS response will typically get to the user before the real one does.[29]

TCP RST is a technique that was once heavily used by the Great Firewall, but has seen a decline in use in recent years.[27], [30] The TCP protocol is entirely connection-based, meaning that communication is dependent upon the maintenance of a connection between the user and a host.

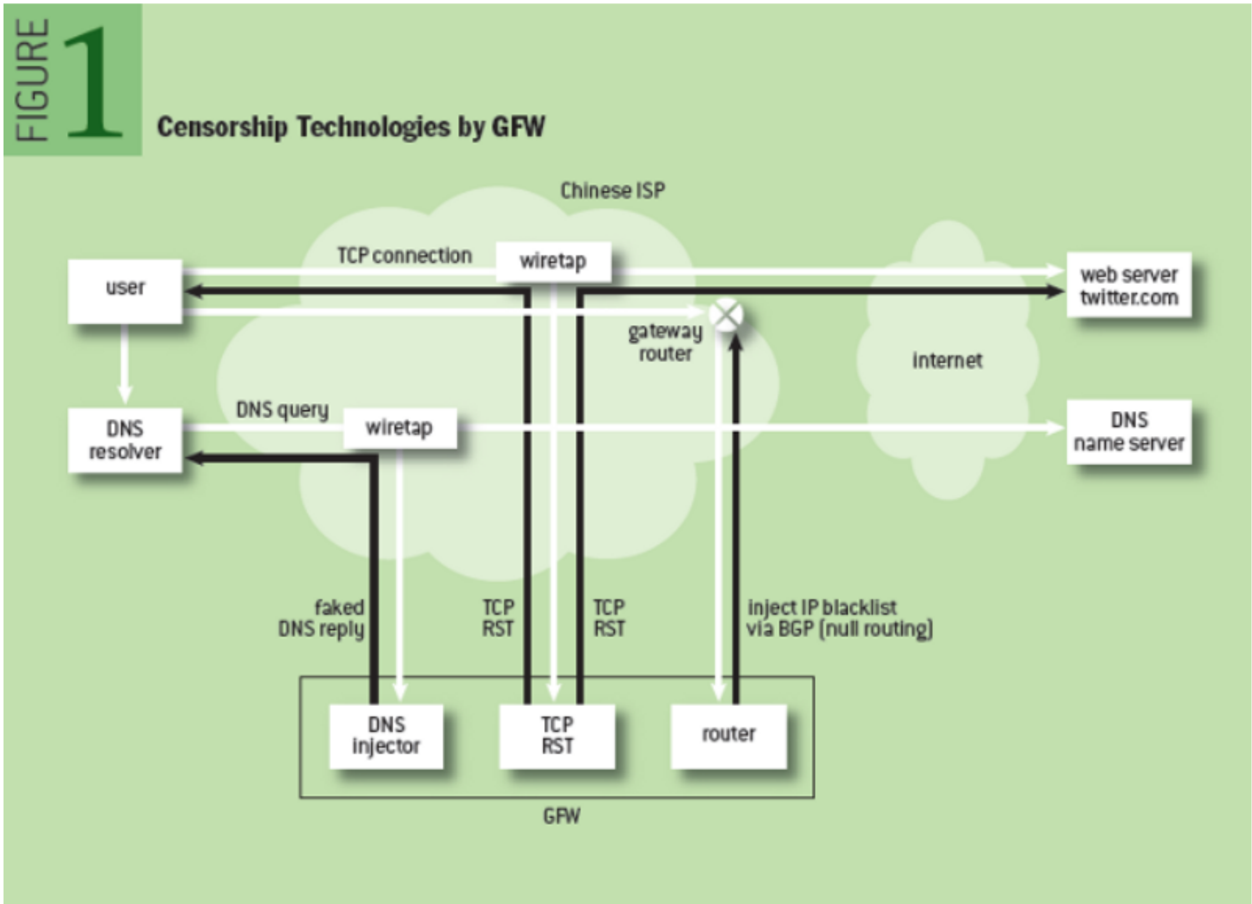


Figure 2: Technologies used by the Great Firewall. [27]

TCP RST packets are essentially a kill-switch for TCP connections. If either side of a TCP connection receives a RST packet, it immediately kills the connection - no questions asked. The Great Firewall searches for banned keywords at various points throughout the network. Upon detection of a banned keyword, the GFW sends a TCP RST packet to both the client and host, spoofing the source IP address and TCP port of both packets as to make them look like they came from the client or host respectively. Much like network black-holing, this process is lightweight and does not affect packets without banned keywords. Despite its effectiveness, TCP RST injections have decreased in recent years. The primary reason for this is that the vast majority of websites have begun to support the HTTPS protocol instead of plain old HTTP. The S in HTTPS stands for secure, and that's because HTTPS is essentially the HTTP protocol wrapped in a layer of encryption. HTTPS encrypts HTTP packets in an encryption protocol called TLS (formerly SSL), which stands for Transport Layer Security. This accomplishes two things: increased privacy across a network, so that things like credit card numbers can be securely transmitted across a network, and more secure authentication, meaning the server that you are talking to can be verified to be the one which you requested. This has been, understandably, of great dismay to the architects of the Great Firewall. When two parties exchange packets over HTTPS, they are the only two people who can encrypt or decrypt the messages sent in the packets. This makes it impossible for the IDS systems of the GFW to inspect HTTPS packets at the HTTP level. Martin Johnson from GreatFire.org, a website that tracks the Great Firewall and offers circumvention tools, had this to say about TCP RST: Martin

"Keyword resets matter much less now with most big websites using HTTPS and so many major ones being blocked wholesale anyway. I just tested a couple of sensitive keywords and the connection was not reset, so perhaps the GFW is using it much less now than it used to."[30] However, HTTPS does not keep the IDS systems of the Great Firewall from viewing IP addresses of packets, nor does it interfere with DNS spoofing. It is unknown whether the Great Firewall continues to use TCP RST injections in connections with normal HTTP servers.

In recent years, the Great Firewall has begun to use more advanced machine learning techniques in order to find and filter out sensitive content.[31] In the past, the GFW had to hire people to wade through swaths of Internet content in order to find websites containing sexual content or content otherwise deemed inappropriate. Nowadays, AI has the capability to recognize such content at a much higher speed. In 2017, the Internet took note after China's machine learning censors began banning all mentions of Winnie the Pooh from blogging platforms, after internet memes began surfacing containing photographs comparing Chinese president Xi Jinping to the cartoon bear. In addition to using AI to recognize banned content, the GFW can also use AI to detect suspicious connections and slow down packets in those connections. This is one of the main counterattacks which the GFW is known to employ against VPNs and other tunneling protocols. The GFW detects to what degree a connection is suspicious (i.e. might be using tunneling protocols) using AI, and slows down the packet rate to that connection, to such a degree that the connection is unusable.[32] The algorithms that it uses to do this are hard to determine. This is one of a number of reasons why VPN's are becoming increasingly difficult to use in China.

COLLATERAL DAMAGE

One of the problems with the Great Firewall is that a censorship program so large is bound to cause unintended damage. This occurs often as the blocking of benign websites in China. One of the reasons for this is that multiple websites are often hosted on the same IP address. If one contains a banned keyword, any website on the address is blocked. The GFW also does not limit its network black-holing to single IP addresses; entire ranges of addresses are blocked, to mitigate the fact that many big-name websites have their servers on multiple IP addresses. It may be a case that a benign website falls within the IP address range of a banned one. This fact has been taken advantage of in the past by those protesting the Great Firewall. In 2008, the website www.falundafa.org, pertaining to the outlawed religious movement Falun Gong, resolved its IP address to the same address as www.mit.edu. This was a problem for people in China attempting to access MIT's OpenCourseWare program. The backlash was so great that the GFW had no choice but to lift the ban on www.falundafa.org.

In recent years, the Great Firewall has had problems regarding HTTPS, the encrypted version of HTTP mentioned earlier. The past decade has seen the majority of the Internet outside of China switching to HTTPS for security and privacy. Since the GFW cannot read HTTP headers encrypted with SSL/TLS, it often does not take kindly to connections using fully encrypted packets. Even though HTTPS/SSL is fully legal in China, and many websites do in fact have SSL certificates (a key element of HTTPS which allows users to authenticate the identity of web servers), they rarely use them. Most of China's major websites, such as QQ and Baidu, do not fully encrypt their traffic, allowing the GFW to perform packet inspection. Websites that do use HTTPS do so at the risk of being marked suspicious, and having their packet loss increased, causing connections to slow down to the point of unusability.

The effects of the Great Firewall are not limited to China. Because the Internet is such a massive, global-scale technology with many different interconnected parts, and because China is such a huge part of it, the Great Firewall has the capacity to inadvertently affect Internet traffic around the world. For one thing, the Great Firewall has made it increasingly difficult for foreigners to do business in China by virtue of the fact that they cannot make contact with many sites that they use on a regular basis. This would not be so big a problem if China were not the industrial center of the world.

One of the ways in which the Great Firewall causes collateral damage is through its DNS spoofing techniques. To understand why this often causes issues even for users outside of China, it is important to understand how the DNS protocol works. DNS, in a nutshell, translates human-readable domain names into IP addresses. When a user types in an address like www.google.com, the request is sent to a DNS resolver, typically the user's ISP. The DNS resolver will then send the query to a number of different servers, which all specify a range of IP addresses in that domain. Domain name servers are structured in a tree, with domains like .com, .net and .org being higher up in the tree, and names like "google" acting as leaves, which carry specific IP addresses (a big-name website like google will typically have multiple IP addresses for its multiple servers). Queries sent from a DNS resolver descend this tree in order to find an IP address, and the very first (highest) level of the tree that they access is called the root server. The root server contains can direct a user toward any of the Top-Level Domains (TLD's), which include .com, .org, and .net. Several of these root servers are in China, and many ISP's in neighboring countries will direct queries to those root servers, whether intentionally or unintentionally. In many cases, if the query is for a banned DNS name, the user will receive a spoofed address. This can happen even if traffic is not going through the root server. If a query passes through the Great Firewall at any point in its descension through the tree, it may end up spoofed. This happens often in countries surrounding China such as those in East Asia and the Middle East, where they might rely on the services of Chinese ISP's.[27] A study measuring 43,000 DNS resolvers in 173 countries outside of China found that as much as 26% of these resolvers in 109 countries were DNS polluted, i.e. they received spoofed IP addresses instead of real ones. In South Korea, almost 80% of DNS resolvers tested were affected. In addition to causing DNS queries to resolve incorrectly, the DNS tampering performed by the GFW sometimes causes huge volumes of traffic to be routed to innocent IP addresses that are accidentally generated by their tampering system. Craig Hockenberry, a network engineer, writes in his blog [33] about how his small web server went down after being bombarded with thousands of requests from China. His server was hit with 13,000 requests per second, which is roughly one third of Google's global search traffic. Obviously this caused his server to completely shut down.

CIRCUMVENTION

For these reasons, it is obvious that the Great Firewall has been subject to many circumvention attempts. One of the primary ways which users attempt to circumvent the GFW is through the use of VPN's. A VPN stands for Virtual Private Network, and it works by tunneling traffic through another server. Essentially, when a user attempts to access a website through VPN software, the VPN software will first encrypt the data to be sent, and then send it through to a dedicated VPN server. The VPN server acts as a proxy, sending the data received from the user to its destination, fetching the response, and sending the data back to the user. This tunneling protocol has the effect of allowing the user to act in a "private network" while having access to public addresses and the Internet. When using a VPN, user traffic not only appears to come from the address of the VPN server, but it is also encrypted, meaning it cannot be accessed or tampered with by outside sources.

Unfortunately, the Chinese government has lately been fiercely cracking down on VPN usage. According to [33], all VPN services are now required to be registered with the Chinese government. Registered VPN services must comply with government regulations, which includes restricting access to banned websites. Those that do not comply or either shut down, or their packet rate is dramatically reduced, making connections impossible. This means that VPN services within China are ineffective at circumventing the Great Firewall. In addition, the Great Firewall has become adept at identifying connections using the OpenVPN protocol, which is the standard protocol upon which most VPN's are built today. However, certain services such as NordVPN [34] operate outside of China, and thus are not under the watchful eye of the Great Firewall. NordVPN also utilized a technique called Obfuscated Servers, which hides traffic as normal HTTPS traffic. The caveat to this is that the Great Firewall does not take very kindly to fully encrypted traffic, and will not hesitate to kill connections using traffic encrypted without a Chinese SSL certificate.

Another way that some users have managed to circumvent Chinese surveillance is using the Tor browser. The Tor browser uses onion routing, which is a process that encapsulates network traffic in a series of layers of encryption, rendering it invisible to a network warden. Many have had issues with using the Tor browser in China to circumvent web surveillance. It was widely used until around 2012, when the Great Firewall first officially blocked the Tor website [35]. Since then, Tor has set up a number of different mirror sites in a bid to stay ahead of the curve.

Clayton et. al[36] describe a technique which ignores TCP RST injections. Essentially, it instructs the computer to discard any packets which arrive with the TCP RST flag set. They found that at the time of the experiment, this was effective at circumventing the GFW. However, there admit a few problems with utilizing this method. First of all, both user and client have to ignore TCP RST packets in order for this strategy to work. Many people in China may not wish to go through the trouble of installing such software. In addition, ignoring TCP RSTs might be risky business, as though rarely used, they perform an important duty in the TCP protocol (a “kill switch”). Also, this experiment was done in 2006, when TCP RSTs were still widely used. A paper from 2016 referenced earlier [30] suggests that TCP RSTs have been phased out of the GFW, and thus this circumvention strategy may no longer be sufficient. As an aside, they also suggest a fun strategy using TCP RSTs which is essentially a DOS (Denial-of-Service) attack. Essentially, when the GFW kills a connection between a user and host, those two cannot communicate for some time afterward, regardless of the content of the connection. By searching for a banned keyword but forging the source IP address to point to that of a government server, one could theoretically cause government servers to lose access to the host server, leaving it unwatched. The authors do not attempt this experiment, but calculate that it could theoretically be useful, if issued on a wide enough scale.

In the search for strategies of circumventing surveillance on the web, one might turn to more sophisticated methods of hiding information. Two of these methods are steganography and covert channels. The next section will be dedicated to a thorough description of covert channels, as they are an integral part of CovertNet. Steganography is essentially the hiding of information in bits of image data. The binary data of an image such that it contains the bits of a hidden message without corrupting the image such that it looks tampered. This technique requires that both client and host have access to software that can insert bits into image data as well as extract bits from it. Both steganography and covert channels are used in the Infranet [37], which is a protocol that will be discussed after a detour into the world of covert channels.

COVERT CHANNELS

A covert channel is essentially a mechanism for transmitting information using a medium not intended for the transmission of said information, or the type of information transmitted. Data is hidden through the nature of its placement or method of transmission. The term “covert channel” was first coined by Butler Lampson in his 1973 paper “A Note on the Confinement Problem,” where he describes covert channels as “those not intended for information transfer at all, such as the service program’s effect on the system load.”[1] For our definition, we expand on this by including channels that are intended for information transfer but can be used to covertly transmit information, typically by virtue of the fact that the data normally transmitted through the channel is no longer of great value to the receiver, such as is the case with the covert channels in various network protocols. The covert channel used in CovertNet will fall into this category. A covert channel is generally a two-way interaction in which a sender and a receiver agree on a channel and both participate in the transmission of information. Thus covert channels differ from other forms of data leakage and corruption in the sense that they require two parties to interact.

Covert channels are a recognized threat to computer security systems, and are even outlined as a criteria in the Department of Defense Trusted Computer System Evaluation Criteria[2], published in August 1983. From the perspective of an organization, they pose threats via the leakage of information both within the organization and out of the organization. From within the organization, data can be leaked down levels of security clearance. Out of the organization, confidential data can be leaked by someone with the right security clearance. Covert channels are often used by Trojan horse viruses to communicate information using the victim's operating system.

As an example of a covert channel, consider a typical function of an operating system, such as opening and closing a file. The operating system is constantly doing such an operation, for non-nefarious purposes, obviously. However, if two users were intent on setting up a covert channel using this mechanism, provided they could both see whether or not the file was opened, the sender of the data could synchronize the opening and closing of the file such that it transmits binary of the data (i.e. open the file to transmit a 0, close it to transmit a 1). Such a channel could be used by an entity with a higher security clearance than another to transmit data only accessible at that level of clearance. Note that in this particular channel the receiver of the data doesn't even need to have the ability to open or close a file, only the ability to see if it has been opened or

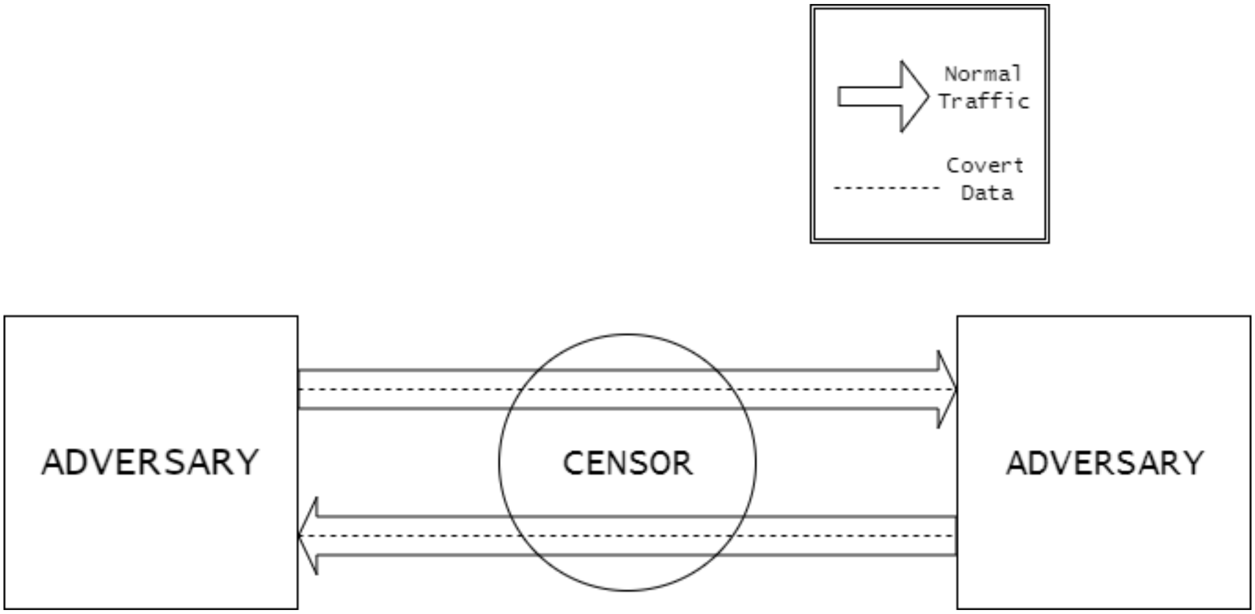


Figure 3: Covert channel structure.

closed. In that sense covert channels can be used to bypass security clearance simply by choosing a channel that is readable by multiple levels of clearance.

CLASSIFICATION

There have been many different systems proposed for classifying covert channels. One such classification which appears to be universal is the division into storage and timing channels. A storage channel is one that transmits information by storing it in a place not intended for the transmission of such data. Such covert channels are easy to implement and offer a high bit rate relative to other channels. One downside to these kinds of channels is that they are more easily detectable than other channels.

Timing channels are slightly more complex, and involve transmitting data by modulating the timing of some other operation. For example, the channel discussed earlier by way of opening and closing a file can be made into a timing channel by varying the amount of time between operations, i.e. short intervals equal 0, long intervals equal 1. Timing channels are generally more covert than storage channels because they can be built in such a way that the behavior of the elements involved appears no different from their normal operation. Also, it is costly and generally impossible for a warden to monitor the timing of every event that takes place in a system. The advantage of covertness comes at the cost of very low bit rate, as only one bit can be transmitted per timing interval.

Brown et. Al[3] introduce a new categorization of covert channels known as behavioral channels. Behavioral channels operate by transmitting data based on the assignment of different behavioral events to pieces of data. For example, the channel discussed earlier, i.e. the opening and closing of a file, might be considered a behavioral channel because it transmits information based on an operation done by the sender. Another example of a behavioral channel might be the navigation of web pages.

Another classification that arises when discussing covert channels is throughput, or the amount of data that can be transmitted through the channel over time[3]. Throughput is generally measured in b/s (bits per second) or kb/s. Throughput is generally inversely related to the covertness of a channel, i.e. the more data a channel can transmit over a given period of time, the higher the risk of the channel being discovered. This is because the channels that are most covert are timing and behavioral channels which generally can only transmit one bit of information per timing interval or operation.

This leads to the next classification of covert channels, which is detectability.[3] Detectability is essentially the ease by which a warden or censor might be able to detect a channel. Brown et al[3] measure detectability as high, medium, or low. As stated previously, detectability is generally at an inverse relationship with the throughput of a channel. The ideal covert channel is one with a low enough detectability to operate without serious threat of interference, but a high enough throughput to transmit the amount of data intended. Straddling this boundary is a key task in choosing a covert channel, and is largely a matter of context.

Okhravi et al[4] introduce more classifications in their paper about covert channel attacks. In addition to storage/timing, they also classify channels into network/OS/hardware and value/transition based channels. Network channels operate in network protocols, and are abundant in the Internet Protocol Suite. OS based channels use some mechanism of the operating system to transmit information. Cioranesco et al.[5] describe a number of different covert channels that can occur in the Process Table, which is an OS data structure that handles RAM. Hardware based channels utilize some aspect of the hardware, and are harder to come by. Value based covert channels actually transmit the information intended somewhere in the mechanism of action, whereas transition based channels transmit information by modulating the value of some element in the system. Storage based channels fall into the value-based category, and timing and behavioral channels are always transition-based.

Smeets and Koot[6] also introduce several more unique classifications of covert channels. One of the most significant is active vs. passive. Active covert channels generate their own traffic, i.e. they generate a mode of transmission and read from it. Passive covert channels piggyback information on top of other transmission protocols. Most network-based covert channels are passive, because they essentially insert information into various places in the protocol. The file-based channel discussed earlier would be classified as active because the traffic is being generated by the channel itself. Timing channels might be considered a mix between the two as the traffic may already exist but is modulated by the channel.

Covert channels can also be classified by the path that the information takes from sender to receiver.[6] The most common method is directly from sender to receiver. All channels discussed so far have been direct. Here the sender transmits information which the receiver directly reads off. Indirect covert channels utilize an intermediate hop with forwards or bounces the traffic to the receiver. Brown et al.[3] describe an indirect network-based covert channel which utilizes the fact that client-side cookies can be made visible to any server the client wishes, meaning a client can be used as an intermediary hop for a server to transmit information to another server. Jaskolka and Khedri[7] describe another indirect channel which uses the 3-way handshake that initiates any connection in the TCP (Transmission Control Protocol). Data can be sent covertly to an intended recipient by spoofing the source IP address in a TCP connection so that it points to the intended recipient. In general, indirect channels are stealthier than direct channels. This is partly because even if a channel is discovered, it is harder to determine the source or sender of the data. Spread covert channels are those which utilize multiple intermediate hops to transmit data to the receiver. These are the stealthiest of all channels.

Finally, both Smeets and Koot[6] and Brown et Al[3] classify covert channels by their robustness. Robustness is simply the reliability by which data can be transmitted across the channel. Particularly in a network, covert channels are likely to encounter a number of different obstacles, including firewalls, proxy servers, and store-and-forward devices. Robustness refers to a channel's ability to survive in its environment, and may be categorized as high, medium, or low.

Jaskolka and Khedri[7] build a graphical representation of the nature of covert channel transmission. The idea is essentially that the data is situated in a data structure, which is

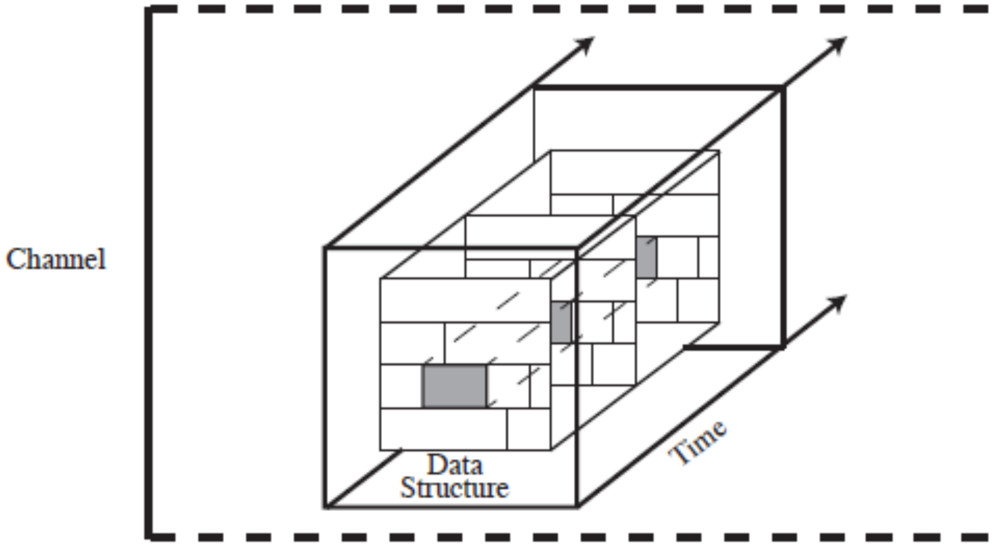


Figure 3: Graphical representation of covert channel transmission.

modulated and transmitted over time in the context of a channel. A covert channel thus can be thought of as the confluence of these layers. For example, the data structure used in our file-based channel from earlier would be the opening and closing of a file, and the channel would be interpreting that mechanism as binary data.

IN NETWORKS

Covert channels are particularly abundant in network protocols, such as those used in the Internet Protocol Suite, commonly known as TCP/IP. The reason for this is largely the nature by which data is transmitted through each protocol. Data is transmitted by passing packets from one protocol to another and appending the packets of each individual protocol in the process. The main protocols in the Internet Protocol Suite, which are IP, TCP, and HTTP, all pass packets containing headers which identify and describe the data contained therein. Many of these headers were introduced into their respective protocols with a purpose in mind that was never as pressing as the creators of said protocol expected, and are thus largely ignored or even completely unused by the protocol they belong to.

The Internet Protocol Suite works as a stack of protocols that each represent a different stage in the transmission of data over a network. When a client wants to make a connection with a host, the client sends data which descends the stack, starting from the highest-level protocol down to the physical layer (fiber optic cables, etc). The data travels, ascending and descending the stack at various points along the way, until it eventually reaches the host and ascends a final time up the stack, where the host receives the data. The highest-level protocols in TCP/IP are the application-layer protocols. These protocols handle the transmission of data at the user-level, and directly correspond to the user's actions. The most heavily used protocols in the application layer are HTTP, used for the navigation of web pages and communication via hypertext, and FTP, which handles the transfer of files across a network. Directly below the application layer is the transport layer, which establishes connections between users across a network and ensures the reliability of data sent across a network.The most heavily used protocols in this layer are TCP and UDP. TCP (Transmission Control Protocol) provides connection-oriented communication, meaning a connection between users is established by way of a three-way "handshake" before data is sent. UDP (User Datagram Protocol), on the other hand, performs connectionless communication, meaning data is simply sent out to a host without first establishing a connection. The layer directly below transport is the network layer. The network layer handles the navigation of packets across a network. The most widely used protocol in this layer is the Internet Protocol (IP). IP navigates packets throughout a network using numbers called IP addresses, which are

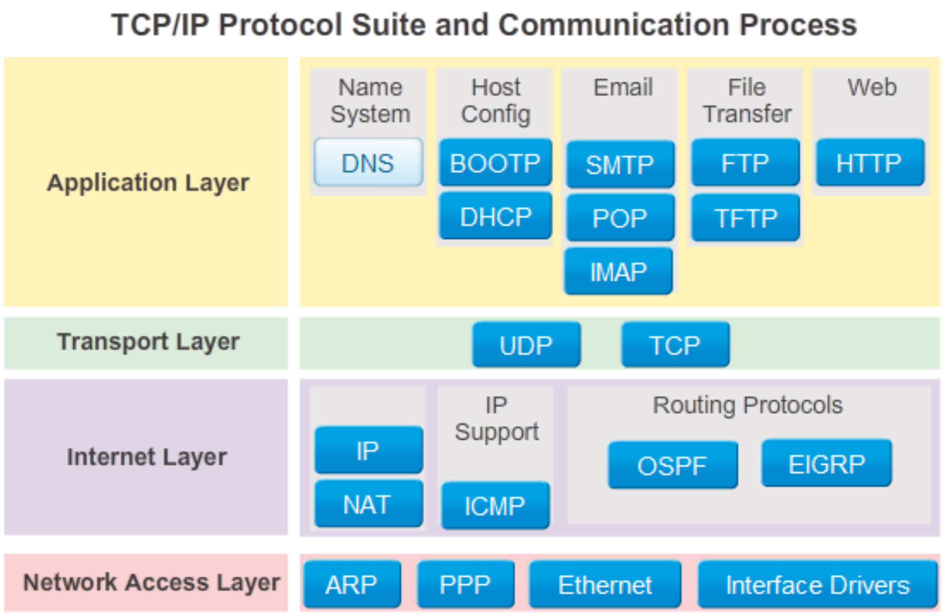
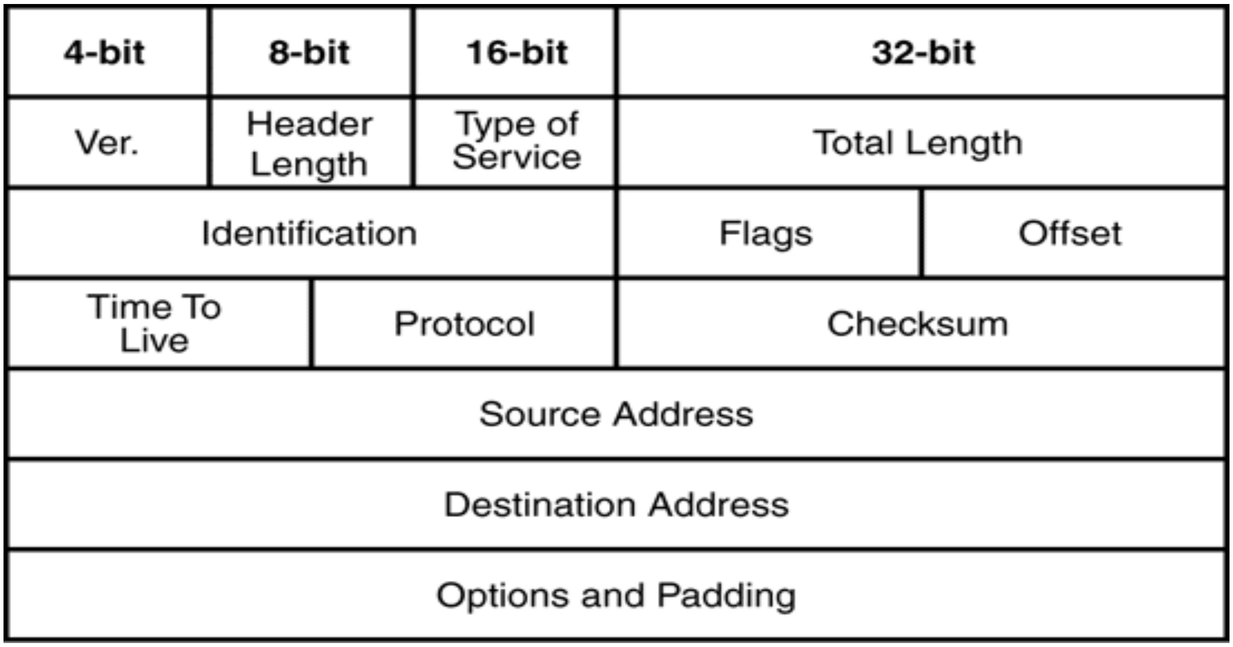


Figure 4: TCP/IP Protocol Suite.

assigned to every device on the system, including hosts as well as intermediary devices like routers.

Mileva and Panajotov[8] discuss the myriad different covert channels that exist in the various protocols of the Internet Protocol Suite. As previously stated, many of these covert channels utilize arbitrary or redundant fields in the headers of packets. In the Internet Protocol, one such arbitrary field is the *Identification* field, which is a 16-bit field used to identify packets and track packets that have been fragmented, or separated into multiple packets to aid transmission. Although the number is typically incremented from the number of the last packet sent, it is effectively arbitrary as long as it remains the same across fragmented packets. Thus Rowland [9] proposes, for each character of data to be transmitted, taking that character's ASCII



<table>
<tr><th>4-bit</th><th colspan="2">8-bit</th><th>16-bit</th><th colspan="2">32-bit</th></tr>
<tr><td>Ver.</td><td colspan="2">Header Length</td><td>Type of Service</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="4">Identification</td><td>Flags</td><td>Offset</td></tr>
<tr><td colspan="2">Time To Live</td><td colspan="2">Protocol</td><td colspan="2">Checksum</td></tr>
<tr><td colspan="6">Source Address</td></tr>
<tr><td colspan="6">Destination Address</td></tr>
<tr><td colspan="6">Options and Padding</td></tr>
</table>

Figure 5: an IP packet.

value multiplied by 256, and transmitting it in the Identification field of IP packets, transmitting the same character across fragmented packets. This would be classified as a storage channel. This is just one of many examples of the use of arbitrary or redundant fields in IP packets.

Covert channels are abound in TCP as well. TCP also uses packets, with a different set of headers, equally useful for covert channels. Smeets and Koot[6] describe a covert channel which utilizes the 3-way handshake that occurs at the beginning of any TCP connection. To start a connection with a host, a TCP client will send a packet with the value SYN, along with a random *Initial Sequence Number,* or ISN. The host, upon receiving the SYN packet, will then send a packet containing the value SYN/ACK, with an acknowledgment number that is usually ISN+1. The client will then send an ACK packet, completing the three-way handshake. Their covert channel takes advantage of the fact that the 32-bit ISN is random, and any data can be inserted into this number without disrupting the TCP mechanism.

As with the other protocols in the suite, HTTP has plenty of covert channels of its own. Brown et. Al[3] describe numerous such channels. One of these channels takes advantage of the fact that client-side cookies can be made visible to any server the client wants, and thus can be used to take in information from one server and transmit it to another. Smeets and Koot[6] describe a channel using custom HTTP headers to transmit information. Essentially, field names for the headers in an HTTP packet typically follow common, predefined values like “Host: (url)” and “Content Type: #”, etc. However, custom headers allow adversaries to transmit covert information (e.g. Foobar:secret). This technique can also be used in the body of both HTTP requests and responses, which is precisely the channel used in CovertNet.

DETECTION/PREVENTION

Mitigating and preventing covert channels is a challenging task, and in some cases downright impossible. This is by virtue of the fact that covert channels often transmit information using mechanisms that cannot be totally eliminated without compromising some part of the system. In addition to this, covert channels can be found in practically every part of a computer system, seemingly to no end. In other words, the only way to ensure that there are no covert channels in a system is to have no system at all. The abundance of covert channels, especially low-bitrate channels in hardware and operating systems, is such that the DoD Trusted Computer Systems Evaluation Criteria allows systems with covert channels of 1 bit per second or less to be considered “secure”. [2]

Despite the difficulty of detecting and mitigating covert channel use, several systems have been proposed. Nagatou and Watanabe [10] propose a system for detecting covert channels which uses a mock version of a system’s security policy called an emulator to determine if a covert channels have occured. Essentially, system security policies on computers are enforced through the principles of flow control and access control. Flow control monitors traffic to ensure that it is only moving in a direction that aligns with the security policy. Access control performs a similar task by assigning to every element in the system a number of access privileges. In order to determine if covert channels have occurred, one can set up an emulation of a system's security policy by creating "dummies" of each element in the system, along with their respective privileges. Then whenever a system call is made, one can compare the results of the system call to the results of that same system call on an emulator. If the two or not the same, it is likely that a covert channel has occurred. This method could be very effective on smaller systems or subsets of larger systems, but the cost of creating an emulator for large computer systems is significant.

Kemmerer [11] describes a technique for determining whether covert channels may exist in a system. The idea is to create a large table called a Shared Resource Matrix (SRM), the rows of which contain every attribute of every shared resource in the system (e.g. every shared file, hardware resource, etc), and the columns of which contain all the primitive operations available on the system (e.g. system calls, open file, read file, write file, execute, etc). In the table, one can then determine whether or not the primitive operation in a column pertains to an attribute of a shared resource, e.g. if it can be performed on the attribute. A covert channel is possible if:

1. The sender and receiver have access to the same attribute/resource.
2. The sender can modify the resource or cause it to change.
3. The receiver can detect the change in the resource.
4. The sender and receiver can synchronize their operation such that they can communicate.

A covert channel exists if these criteria are met. System designers can then begin the process of eliminating covert channels by changing aspects of the attributes/resources and the primitive operations so that the covert channel is no longer possible.

Covert Flow Trees (CFT's) are another technique for detecting covert channels developed by Kemmerer and Porras [12]. A covert flow tree is essentially a data structure that maps out all of the different operation sequences that might change a shared resource in a system. The tree makes it easier to determine where covert channels exist and which ones might be more useful to adversaries. The disadvantage of CFT's is that on a large computer system, they can become massive, and traversal can be difficult.

Shieh [16] proposes a system for measuring the bandwidth of covert channels based on a model of covert channels that views the variables in the channel in states, and the channel being the transmission of encoded messages using the transition of variables between those states. Shieh constructs a graph with nodes representing possible states of the variables and edges pertaining to transitions of variables between states. The total time for the transition from state i to state j is denoted as T_{ij}, where T_{ij} equals $2T_{cs} + T_{aij} + T_{vij} + T_{envij}$, where T_{cs} is the context switch time, T_{aij} is the time it takes for an operation primitive to alter a variable, T_{vij} is the time it takes for a receiver to view an altered primitive, and T_{envij} is the setup time for the channel environment. The maximum bandwidth of a covert channel is then defined as

$$C = \lim_{t \to \infty} (\log_2 N_i(t)) / t$$

Where $N_i(t)$ is the number of state transitions possible in a given time t.

Lucena et al. [17] propose a type of network monitor called a warden, which filters and processes traffic looking for covert channels. A warden does extensive analysis on network packets, searching for information that does not match with normal traffic or appears out of place. Wardens can be of three types: stateless, stateful, and network-aware. Stateless wardens only inspect the contents of individual packets, without regard for other packets in the flow of traffic. Stateful wardens do take into account other packets in the network flow and could inspect packets on an individual level as well as in the context of the flow of traffic in which they reside. Network-aware wardens are not only aware of the flow of traffic, but are also aware of the topology of the entire network in which a packet is sent. The three types of wardens represent increasing levels of awareness with regards to covert channel detection, and also increase in scale and difficulty to implement.

The practical use of covert channels is essentially two-fold. The first major use of covert channels is that of circumventing levels of security clearance, i.e. to transmit information only accessible to someone of a high clearance to someone of a low clearance. This has been discussed in depth. The other major use of covert channels is to send prohibited data in a system guarded by a censor. Covert channels are particularly useful for this purpose because they are hard to detect and are designed to appear like normal traffic. This use of covert channels takes us back to the Great of Firewall of China and the circumvention of web censorship. The idea is that covert channels can be used as a mechanism by which a user under the Great Firewall, or any mass censorship program for that matter, can make contact with a server outside of the censorship program and exchange censored information under the censor's nose.

INFRANET

The Infranet [37] is a protocol developed in 2002 at MIT which disguises web traffic using a covert channel as well as steganography. Infranet acts as a two-part protocol, the first part being a client-side proxy and the other side being a dedicated Infranet server. Essentially, Infranet filters web traffic through a web proxy on the client's computer. The proxy disguises the content using a covert channel, and sends the disguised content to an Infranet server. The server then interprets the data in the channel to extract the original requests. It sends these requests to the websites they are designated for and fetches the response. It then sends back information to client proxy in the form of steganographic images.

The first part of Infranet is the tunnel setup. The tunnel setup initializes an Infranet connection and allows the exchange of shared keys between a client and an Infranet server. The tunnel setup protocol begins when the client first makes a request to an Infranet server by requesting an HTML document such as index.html. The server sends the client the HTML document, and the client responds by using a modulation function specific to the Infranet server, u_{init}, to send a key, SKEY, that will be used by the server later in the setup. SKEY is also encrypted using asymmetric before being sent in u_{init} so as to ensure confidentiality. SKEY is sent using the server's public key modulated by u_{init}; the server applies the inverse of u_{init} and then uses its private key in order to get SKEY. Optionally, the server can specify which modulation function u_{init} it wants the client to use in order to send SKEY. The server then responds by identifying the modulation function, u_{tunnel}, that the client should use in order to covertly request data in the future. The server sends this function by encrypting it using symmetric encryption with SKEY. The client retrieves the function using SKEY and commences the covert exchange of information using u_{tunnel}.

The covert channel used in Infranet from the client-side proxy to the web server, u_{tunnel}, is a behavioral channel based on the navigation of web pages. The basic idea is that the server is set up as a normal, functional website, and the user (client-side proxy) sends HTTP requests to links

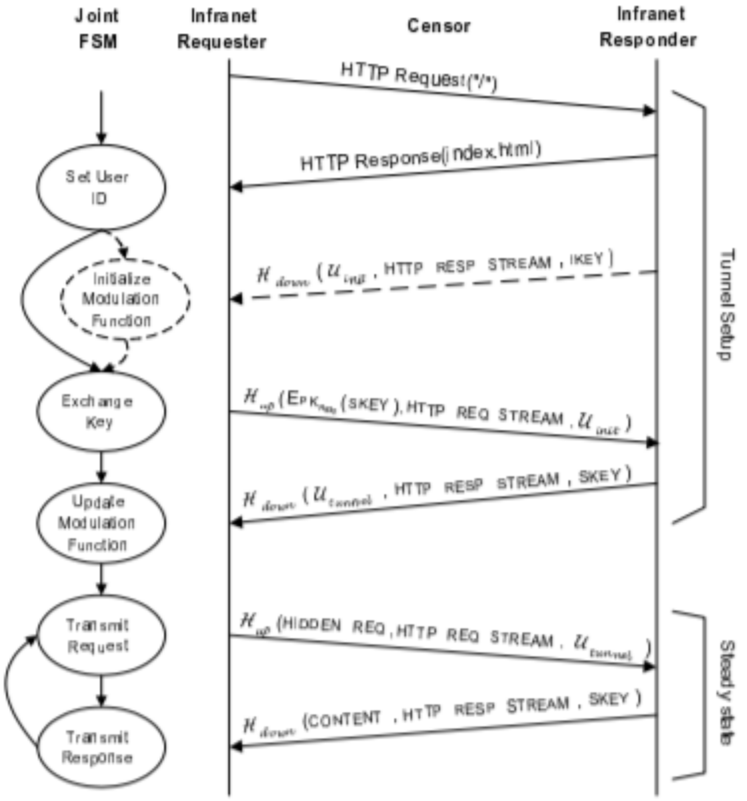

Joint FSM
Infranet Requester
Censor
Infranet Responder
HTTP Request("/")
HTTP Response(index.html)
Set User ID
Initialize Modulation Function
$\mathcal{H}_{down}$ ($\mathcal{U}_{init}$, HTTP RESP STREAM, IKEY)
Exchange Key
$\mathcal{H}_{up}$ ($E_{PK_{resp}}$ (SKEY), HTTP REQ STREAM, $\mathcal{U}_{init}$)
Update Modulation Function
$\mathcal{H}_{down}$ ($\mathcal{U}_{tunnel}$, HTTP RESP STREAM, SKEY)
Tunnel Setup
Transmit Request
$\mathcal{H}_{up}$ (HIDDEN REQ, HTTP REQ STREAM, $\mathcal{U}_{tunnel}$)
Transmit Response
$\mathcal{H}_{down}$ (CONTENT, HTTP RESP STREAM, SKEY)
Steady state

Figure 6: The Infranet tunneling protocol.

on the site in such an order that they can be interpreted by the Infranet server as content. The order in which the client-side proxy sends requests is dependent on a modulation function agreed upon by the client and server beforehand. This modulation function can be one of a number of different functions. One of these is implicit mapping, which simply maps a bit to a particular HTTP request. This function can be implemented but has extremely low bandwidth and was not implemented by the Infranet team. The modulation function used in the implementation of Infranet is a function called range mapping, first described by Adler and Maggs [38]. The basic idea behind range mapping is to map chunks of hidden messages to specific URL's rather than single bits. In the implementation of Infranet, they choose the mapping of URL's to chunks of messages based on the probability distribution of normal traffic on the website hosted by the server. In this way, chunks of messages that the user clicks on more often are represented by links that a normal user of the website would be more likely to click on often. This channel is particularly covert because it appears to a censor as the normal navigation of web content. There is no data stored anywhere, making the channel quite hard to detect. The steganographic agent from the web server to the client proxy is established using Outguess. Outguess is a steganographic program which hides bits in the high-frequency components of JPEG images. Essentially Outguess finds redundant bits in JPEG images, and then embeds bits to be sent in some subset of those redundant bits. The subset is determined by a key shared with the user, SKEY.

While Infranet is a very useful protocol, it has several disadvantages. First and foremost, the protocol is now outdated. It utilizes Outguess, which is a program that was taken off the web long ago. I was only able to download it by accessing an archival site. Furthermore, users may not want to have to install another piece of software in order to get Infranet up and running. The other problem with Outguess is that the traffic from the server to the client is essentially a massive stream of JPEG images. This might look suspicious, even if the guardians of the Great Firewall were unaware of the Infranet protocol. If they were, it would be easy to spot somebody who is receiving a seemingly endless stream of JPEG images from a site, as well as a site that is serving thousands of clients with only JPEG images. Another disadvantage is that, while the covert channel used from the user to client is extremely covert, it has a relatively low bandwidth, due to its nature as a behavioral channel. Essentially, the measure of bandwidth that they take in Infranet starts with a measure of the average number of links it takes to transmit one message. In this case, a message would be a single HTTP request to a censored website. They calculated that this number is on average

$$\left\lceil \frac{16.5}{\lg k} \right\rceil + 2$$

Where k is the number of links on a website run by an Infranet server. Because this number is greater than 1, the implication is that Infranet inevitably slows down bandwidth, i.e. it always has a slower bandwidth than normal traffic would.

For these reasons, I sought to develop a covert-channel based protocol for circumventing web surveillance which improved upon Infranet. The protocol had to have several characteristics:

1. Simple to implement.
2. No outdated dependencies.
3. Does not slow down web traffic by a significant amount.
4. Will operate covertly under the Great Firewall of China.

CovertNet is the culmination of these four goals. It was relatively simple to implement, using a C++ HTTP server library on the client side and a Java Servlet on the server side. It does not require the installation of outdated programs. It does not slow down traffic as much as Infranet; in fact, an entire HTTP request is hidden inside of another request, making the speed almost the same (save for processing time on the part of the client/server). Most importantly, the covert channel is designed specifically so that it operates covertly under the Great Firewall of China, without sacrificing bandwidth. What is significant is that a balance has been achieved between detectability and throughput, taking into consideration the context of the channel (the GFW).

COVERTNET

The basic setup of CovertNet is as follows. On the client side, the user sends HTTP requests for censored content to the installed CovertNet proxy on the client's computer. The CovertNet proxy bundles these requests in a layer of encryption, namely AES. It then assembles them in accordance with the covert channel chosen for CovertNet. The client-side proxy then sends the requests, encrypted and hidden in the covert channel, to the server side of CovertNet. The server side of CovertNet would act as a normal website, most likely a blog or another kind of website which intakes large numbers of POST requests on a daily basis. In order to circumvent the GFW, the client-side proxy will first send the server an initial GET request for one of its pages that does not contain requests for any censored content. The POST requests following the initial GET request will contain censored content. The server disassembles the covert channel in order to retrieve the encrypted content. It then decrypts the content, and assembles it into the request initially sent to the client-side proxy by the user. The server sends this request to its intended host (presumably a censored server), and then retrieves the response of the censored server for the request made. The server then encrypts this response, and bundles it into the covert channel. It sends the content back, bundled in the covert channel, which is received by the client-side proxy. The client-side proxy disassembles the covert channel to get the encrypted response, decrypts it, and sends it back to the user as the full response.

The channel used in CovertNet was chosen by examining the environment it was meant to be placed in and measuring potential channels based on each classification, as well as ease of implementation. The Great Firewall of China was used as a basis for the environment that the CovertNet channel was meant to be used in, for multiple reasons. First of all, the Great Firewall of China is the largest censorship system of its kind in the world. There are more internet users in

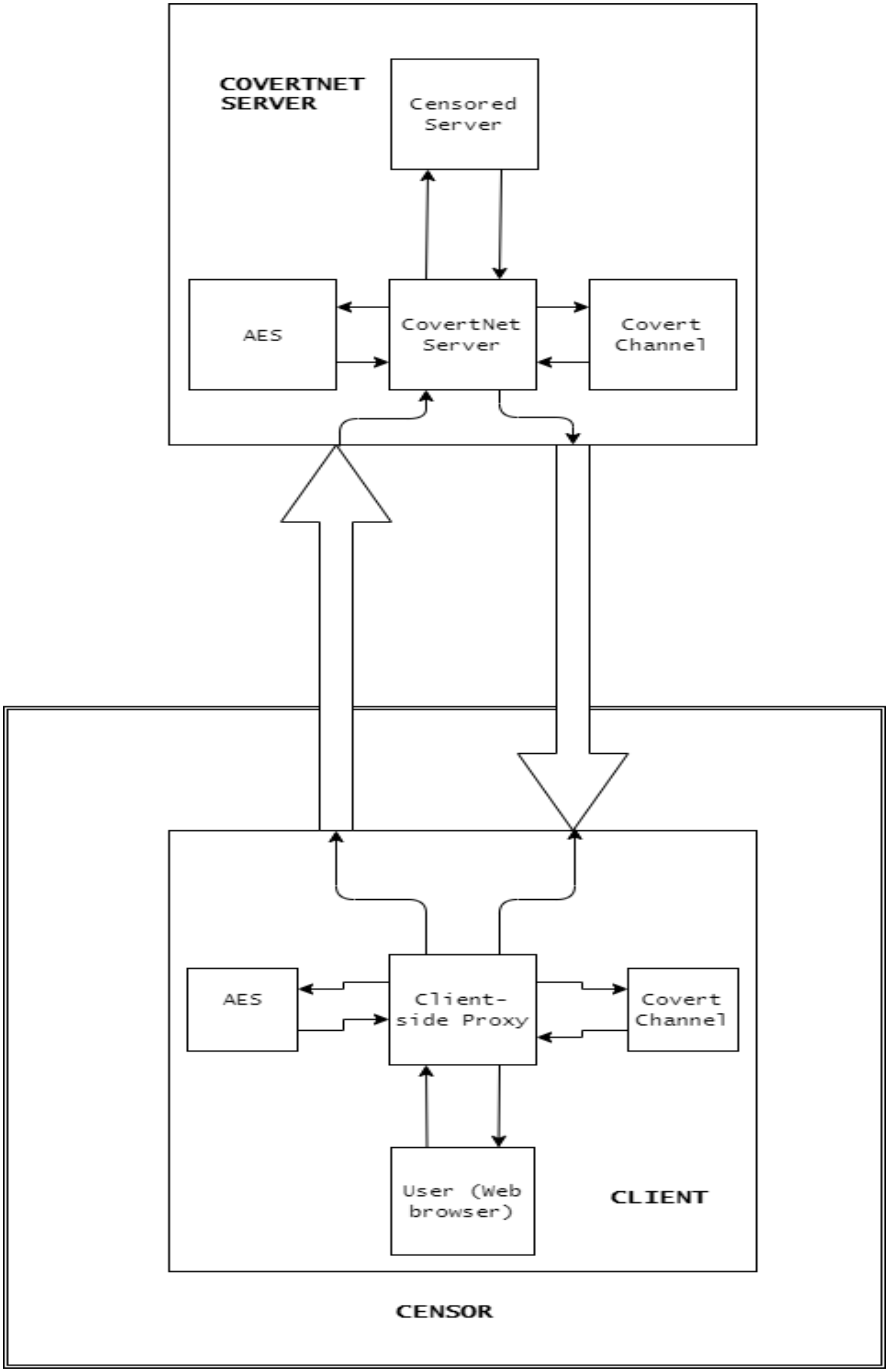


Figure 7: The CovertNet Protocol.

China than in any other country in the world, and so if CovertNet were successfully deployed, China would likely be the nation in which it is used most. Perhaps more importantly, the Great Firewall is also one of the stricter censorship programs in the world, so a channel which operates well in the Great Firewall can be assumed to operate well in other censorship programs without the need for too much alteration. In addition, the phasing out of both TCP RST attacks as well as deep packet inspection of HTTP requests other than GET requests has actually made it easier to find a covert channel in HTTP that will work in the Great Firewall.

There were several inspirations for the covert channel used in CovertNet. These include several of the channels introduced by Brown et. Al [3]. One of these channels hides information in the body of a GET request. Normally, GET requests do not contain message bodies, although they can; hidden information can be stored in them. This was a good start, but my concern with this channel was two-fold: 1. Message bodies in GET requests could appear suspicious to the GFW. 2. The GFW, according to [28], does most of its deep packet inspection on the first GET request in a given connection. This would make it a bad place to put a lot of censored data, encrypted or not. I decided a more covert way to store information would be in the message bodies of other types of HTTP requests, which do normally contain message bodies and are not as thoroughly inspected by the GFW.

COVERTNET AND THE HTTP PROTOCOL

To fully understand the covert channel used in CovertNet, it is necessary to understand the HTTP protocol more deeply. HTTP is a text-based protocol which sends out requests to host websites based on the navigation of users throughout the web. Each HTTP request contains a method (the action intended by the request), url, and several headers which provide information about the request. Some requests also contain a body, which holds data to be sent to a server.

Requests are generally categorized by their method, which is the first attribute in a request. The most common types of requests are GET and POST. Other methods include HEAD, PUT, DELETE, and OPTIONS. These perform special operations and will not be discussed in the context of CovertNet.

GET requests ask a host to send a particular webpage to the client. Whenever you open up a webpage, your computer is sending a GET request to the host website. GET requests typically do not have a body, although it is possible to send a GET request with a body, and this actually works as an effective covert channel. POST requests send data to a server, and are typically used when filling out forms and queries. POST requests always have a body. The data in the body can take one of three forms, `application/x-www-form-urlencoded` , `multipart/form-data` , or `text/plain` . The type is specified in the Content-Type header. The first is URL encoding, and is the most common. URL encoding, also called percent-encoding, is an encoding standard that, in addition to being used in POST requests, is used most often to encode data in a Uniform Resource Identifier, or URI, which identifies a resource on a network. URL encoding uses special reserved characters, such as “/”, “!”, and “;”, to represent different pieces of information in a URL. For example, “/” is used to separate parts of a URL for a DNS (Domain Name System) to parse. When used in the body of POST requests, it is usually used to represent an HTML form as a series of key-value pairs. Pairs are instantiated using “=”, and separated using “&”. For example, the form

```
<form action="/urlencoded?firstname=foo&lastname=bar" method="POST" enctype="application/x-www-
form-urlencoded">
    <input type="text" name="username" value="foobar"/>
    <input type="text" name="password" value="foosecret"/>
    <input type="submit" value="Submit" /></form>
```

Can be represented in URL-encoding as:

```
username=foobar&password=foosecret
```

This makes the submission of form data on the internet a much less bulky process than sending full html code. `multipart/form-data` is similar to URL encoding, but allows clients to send a file as well as HTML code. `text/plain` is simply plaintext and is generally only used for testing purposes.

HTTP responses are very similar to HTTP requests. The primary differences are that they lack a method, contain a response code instead, and typically always contain a message body. The response code of an HTTP response indicates whether the connection has succeeded or failed. The response code is essentially a number (in the hundreds) and a word. For example, a typical response line for a fulfilled HTTP request would be `HTTP/1.1 200 OK.` There are many response codes, which are lumped into categories denoted by the first digit of the number. Response codes in the 200's are those that have been successful. Response codes in the 300's are those that have been redirected. Response codes in the 400's and 500's have failed. 400's generally denote a failure on the part of the client (bad request); response codes in the 500's indicate server failure. After the response line come a number of different headers, particular to each response line. These usually indicate length and type of the content of the response. If a response is successful, the content requested by the user is located in the body of the response.

The channel used in CovertNet hides HTTP requests in the body of POST requests using URL encoding. CovertNet parses an HTTP request from the client and sends a POST request of type `application/x-www-form-urlencoded` to a CovertNet server. The body of the POST request contains a carefully prepared set of key-value pairs. Each key is a random lowercase letter from a-z, followed by a number incremented for each key-value pair. The letters are the same for each key-value pair. The values for each key-value pair are the elements of the HTTP request (methods, headers, body), encrypted using the AES (Rijndael) block cipher. The method, protocol, and query URL of the request, which always appear as the first line of the request, count as one element, as does each header. If the request to be transmitted is a POST request or another HTTP request which includes a body, then it too is included as the value in a key-value pair. As an example, the following GET request:

```
GET /doc/foo.html HTTP/1.1
Host: www.bar.com
Accept: image/gif, image/jpeg, */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
```

would be URL-encoded as follows (square brackets denoting AES-encrypted values):

```
x1={GET /doc/foo.html HTTP/1.1}&x2={Host:
www.bar.com}&x3={Accept: image/gif, image/jpeg, */*}&x4={Accept-
Encoding: gzip, deflate}&x5={User-
Agent:Mozilla/4.0}&x6={Content-Length: 35}
```

When the CovertNet server receives the POST request, it extracts all of the values from the key-value pairs and decrypts them. It then assembles them back into the request as sent initially by the client, and makes the request using the host and URI specified therein. Upon receiving the response from the censored server, the CovertNet server performs a similar operation to that which was performed by the client-side proxy, albeit slightly different. The CovertNet server first encrypts each element of the response. It then assembles the encrypted response obtained from the server and sends it as the body of its own response to the client-side proxy. The client-side proxy then first disassembles the body of the response from the server and decrypts it, revealing the response of the censored server. The client-side proxy finally sends this response to the client, which the client can view in their web browser.

There are several reasons why the CovertNet channel would operate in the Great Firewall. First and foremost, the design of CovertNet is that it is intentionally not entirely encrypted. Only data which is assembled and hidden in the body of requests and responses are hidden; the methods and headers themselves are not. This is important because it automatically makes a CovertNet connection less suspicious to the GFW and thus less likely to be killed than a connection over a VPN. In addition to this, sources have shown [28], [30] that filtering of HTTP requests has been decreased, if not entirely phased out, over the past decade. In 2009, it appeared as if the only HTTP requests being filtered were the initial GET requests in a connection.[28] HTTP responses from servers were not filtered at all. Importantly, CovertNet does not send transmit any censored information from the initial GET request of a connection, but does so in later POST requests and, from the server side, HTTP responses. Essentially what this means is that 1. The GFW is likely to only scan the initial GET request in a CovertNet connection for banned keywords, 2. If the GFW scans the POST requests coming after the initial GET request, it will not see suspicious keywords as the censored data is hidden encrypted in the body of the POST request, and 3. The GFW is not likely to scan the HTTP responses at all, and if it does, it will not see censored information as it sits encrypted in the response body.

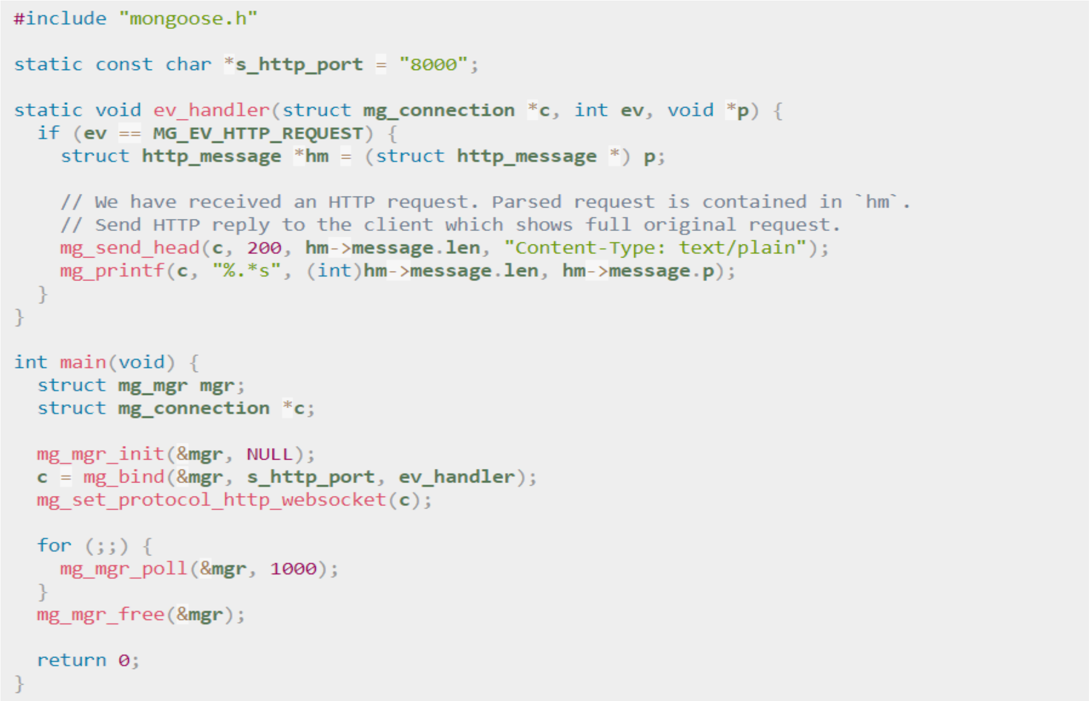
For the most part, the implementation of CovertNet achieved these goals. The current implementation of CovertNet, when hooked up to a web browser, does allow connection to HTTPS sites with a seemingly normal speed. However, the biggest issue that was faced in the development of CovertNet was support for SSL/TLS. The only course of action that was found in order to be able to process and make requests using full HTTPS was to obtain a CA-validated SSL certificate by purchasing a domain and hosting services and registering a certificate with that domain. The reason why this was the only option is because the client-side proxy that is built into CovertNet must act as an HTTPS server when requests from the web browser are forwarded to the localhost through the proxy settings of the computer. If the client-side proxy does not have a CA-validated certificate, meaning the certificate was generated by a validated authority such as GoDaddy.com, the web browser simply will not forward HTTPS requests to the proxy. This is to ensure security, which is the primary purpose of HTTPS. The web browser cannot ensure the safety of certificates that are self-signed. The implications of this are that CovertNet may not be the first option for individuals who are not tech savvy or who are not inclined to purchase a domain name and SSL certificate. The good news is that CovertNet may be a viable option for bigger corporations, or those who are tech savvy or who already own a website and SSL certificate.

IMPLEMENTATION

CovertNet was developed in two parts - the client-side proxy in C++, and the CovertNet server as a Java servlet running on Apache Tomcat. Initially, the idea was to build an HTTP application using simple socket programming in C++. However, attempting to re-implement functions of the HTTP protocol using socket programming proved to be much like re-inventing the wheel- pointless and not worth the time and effort. For this reason, the client-side proxy of CovertNet was ultimately developed in C++ using the Mongoose networking library. Mongoose is essentially a C++ library for developing networking applications using HTTP and HTTPS. It allows users to set up HTTP servers and clients without worrying about implementing the nitty-gritty details of the HTTP protocol. For SSL and crypto support, I also used the OpenSSL libraries, which contain support for SSL and various other cryptographic algorithms.

Mongoose builds relationships between HTTP servers and clients using connections. A connection represents an HTTP transmission between a server and a client. It is implemented using `struct mg_connection`. All connections in a Mongoose C++ program are managed by an "event manager." The event manager is initialized at the beginning of the program using `mg_mgr_init()`, which takes as a parameter the address of a struct representing the event manager, a `struct mg_mgr`. The role of the event manager is essentially to control all connections and determine which events are associated with which connection. Every connection is either outbound, representing a client, or listening, representing a server. Servers listen on ports; the initialization function for a server is `mg_bind()`. This returns a connection which represents that from server to client.

Clients make outbound connections to listening servers using `mg_connect()`. Both `mg_bind()` and `mg_connect()` take event handler functions as parameters. Event handler functions are functions that are called whenever an event occurs which is associated with a certain connection. An event is essentially an HTTP request or response. The event handler function is implemented by the user, and determines what the user does with the request or response received. Event handler functions are initialized as ev_handler() and passed, among other parameters, to the initialization functions of connections.



```
#include "mongoose.h"

static const char *s_http_port = "8000";

static void ev_handler(struct mg_connection *c, int ev, void *p) {
  if (ev == MG_EV_HTTP_REQUEST) {
    struct http_message *hm = (struct http_message *) p;

    // We have received an HTTP request. Parsed request is contained in `hm`.
    // Send HTTP reply to the client which shows full original request.
    mg_send_head(c, 200, hm->message.len, "Content-Type: text/plain");
    mg_printf(c, "%.*s", (int)hm->message.len, hm->message.p);
  }
}

int main(void) {
  struct mg_mgr mgr;
  struct mg_connection *c;

  mg_mgr_init(&mgr, NULL);
  c = mg_bind(&mgr, s_http_port, ev_handler);
  mg_set_protocol_http_websocket(c);

  for (;;) {
    mg_mgr_poll(&mgr, 1000);
  }
  mg_mgr_free(&mgr);

  return 0;
}
```

Figure 8: A simple Mongoose web server.

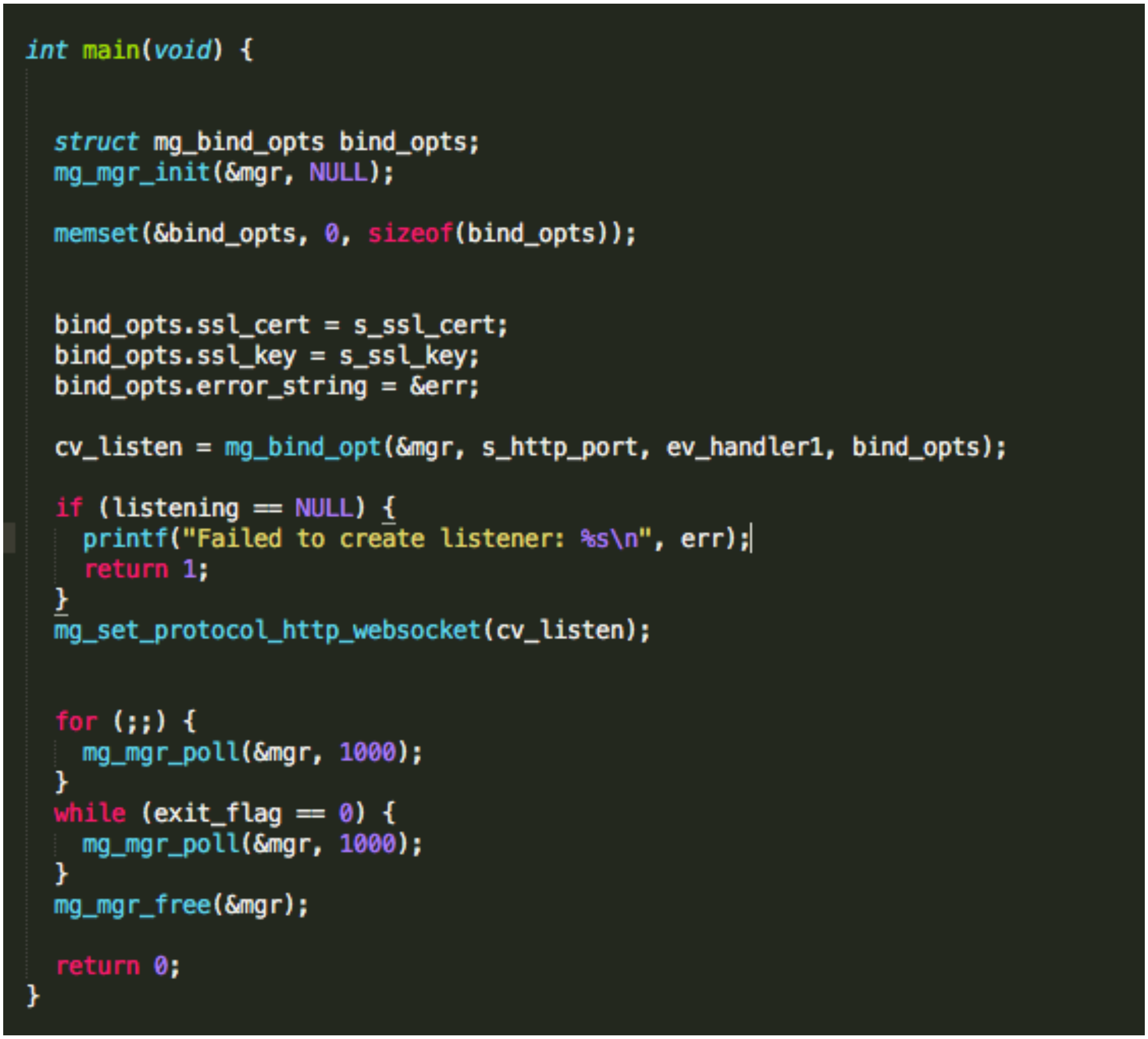
One of the parameters passed to the event handler function is a `struct http_message`. A `struct http_message` holds the data of the request or response when it occurs as an event. A client that has made an outbound connection and gets a reply can access the reply from this struct.

Different functions are available for listening and outbound connections in the Mongoose networking library. Listening connections, upon receiving a request, can use `mg_send_head()` to send a response to the client. The response is fully customizable given the parameters passed to the function, which generally consist of the headers to be sent in the response, along with an optional body.

With regards to encryption and SSL support, I utilized OpenSSL for both. OpenSSL provides libraries for both SSL support and the support of various other cryptographic algorithms. For symmetric encryption between the two parties, I used AES. AES is a block cipher algorithm which requires only one symmetric key that is kept secret and only shared between the two communicating parties. After the client makes its first GET request to the CovertNet server, the server generates a random 256-bit number which will be used as the key in the encryption process. This key is stored in the body of the HTTP response. The security of this key is dependent upon the fact that, as mentioned previously, the Great Firewall does not scan HTTP responses for content. In the future, I would like to use an asymmetric public-key algorithm like RSA to encrypt this key, which would be standard practice. However, time constraints meant that for now it will have to be trusted that the Great Firewall does not poke around in those parts. The encryption and decryption functions are handled in encrypto.cpp and encrypto.h, using the class “encrypto” and its public functions.

The basis for the client-side proxy in CovertNet is Proxy.cpp. This program acts as both a client and a server, listening in for HTTP requests being made on the localhost as well as making requests to the CovertNet server. `Proxy.cpp` also hands off all data to be encrypted on the client side and receives it back. To start, after initializing the event handler as well as other miscellaneous variables such as error codes, the proxy creates a connection called `cv_listen.` `cv_listen` represents the connection listening on the localhost. Localhost is the loopback address, which is essentially the IP address of the client’s own computer. Packets that are sent to the localhost are simply routed back to the client’s computer. The user’s web browser will be connected to the localhost in order to link up with the listening connection in `Proxy.cpp.` `cv_listen` is initialized using `mg_bind_opt()` which initializes a listening connection with SSL support. Provided to this function are the SSL certification and key needed to process HTTPS over SSL.

An SSL connection begins when a client attempts to connect to an HTTPS server. Upon first contact, the client requests that the web server identify itself using an SSL certificate. The



```
int main(void) {

  struct mg_bind_opts bind_opts;
  mg_mgr_init(&mgr, NULL);

  memset(&bind_opts, 0, sizeof(bind_opts));


  bind_opts.ssl_cert = s_ssl_cert;
  bind_opts.ssl_key = s_ssl_key;
  bind_opts.error_string = &err;

  cv_listen = mg_bind_opt(&mgr, s_http_port, ev_handler1, bind_opts);

  if (listening == NULL) {
    printf("Failed to create listener: %s\n", err);
    return 1;
  }
  mg_set_protocol_http_websocket(cv_listen);


  for (;;) {
    mg_mgr_poll(&mgr, 1000);
  }
  while (exit_flag == 0) {
    mg_mgr_poll(&mgr, 1000);
  }
  mg_mgr_free(&mgr);

  return 0;
}
```

Figure 9: Main from Proxy.cpp.

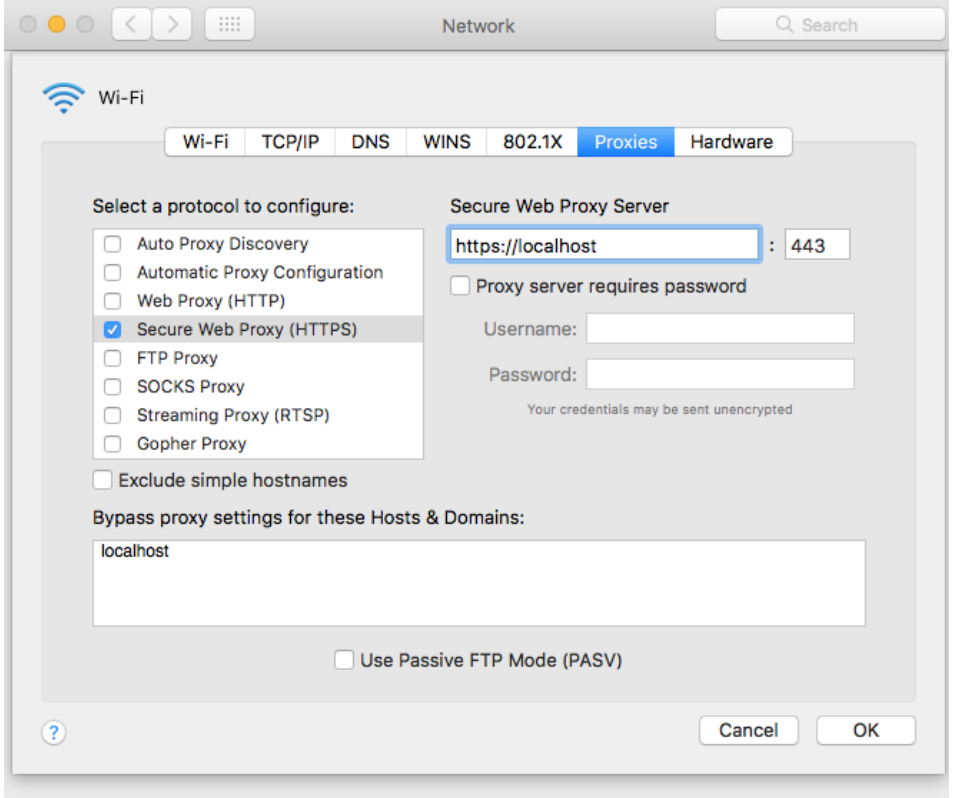
server then sends the client a digital certificate created by a trusted certificate authority, along with a public key to be used by the client. The client verifies the validity of the digital certificate, and then uses the public key to agree on a “session key” with the server. The client and server then can exchange information using symmetric encryption with the agreed-upon session key. In the context of CovertNet, the certificate and public key ultimately allow the web browser to verify the validity of the client-side proxy, even if the two are on the same computer. Without these, the web browser will refuse to talk with the client-side proxy over HTTPS.

The event handler function for `cv_listen` is triggered whenever the user's web browser makes a request to an HTTPS website. The event handler function first stores the request made by the web browser in a global variable so that it may be accessed by other connections in the future. It then makes a GET request to the CovertNet server, which creates a new outbound connection. This connection has its own event handler, which is triggered when the client-side proxy receives a reply from the CovertNet server. This reply to the first GET request will, if successful, contain the key needed for encryption between the proxy and server. The event handler function assigns the key in Proxy.cpp to this value. It then calls `sendRequest(),` which makes another outbound connection to the CovertNet server. Using this connection, the client-side proxy and CovertNet server will exchange POST requests containing the covert messages. Before this connection is made, `sendRequest()` calls `packRequest(). packRequest()` takes the request made by the web browser and stored earlier in a `struct http_message` and bundles it into the covert channel. In this function is where each piece of data is encrypted using `encrypt()`, which is a method from the `encrypto` class.

Upon receiving the response from the CovertNet server containing censored data in the body, the event handler function for this outbound connection is called. This event handler function disassembles the data using `parseResponse()`. This function takes apart the data bundled in the covert channel and decrypts it. It then assembles this decrypted data into three parts: a response code, headers, and the message body. These three parameters are used to send a reply to the web browser using the original listening connection `cv_listen`. These responses will contain the censored data.

The server side of CovertNet is implemented using a Java servlet, built on top of Apache Tomcat. The main Java file for the servlet is separated into two parts: `doGet()` and `doPost()`. These are called when the servlet receives requests with the respective methods GET and POST. For HTTPS support, I utilized the Java HttpsUrlConnection class, which offers client-side HTTPS support. `doGet()` generates a random 256-bit key and sends it in the body of an HTTP response with the response code `200 OK`. `doPost()` is where the bulk of the work is done. Essentially, the servlet disassembles the body of the POST request and decrypts the fields therein. It then assembles this back into a legitimate request and makes the request to the server specified in the `Host:` field of the request. When it receives the response, it encrypts the fields of the response and bundles them into the covert channel. It then sends this bundled response in the body of an HTTP response with the response code `200 OK`. It should be noted that the servlet also utilizes OpenSSL, much the same as the client-side proxy, in order to handle AES encryption.

To run CovertNet, one must first configure their web browser to use a proxy. The proxy must be set to https://localhost:443, as this is the address that the CovertNet proxy listens on. It should be noted that at this stage, CovertNet only supports connections to HTTPS sites, and not HTTP. However, at this time more than ½ of the websites on the Internet use HTTPS[16], and nearly all of the world's major websites which account for the majority of network traffic do use HTTPS.



Select a protocol to configure:

□ Auto Proxy Discovery
□ Automatic Proxy Configuration
□ Web Proxy (HTTP)
☑ Secure Web Proxy (HTTPS)
□ FTP Proxy
□ SOCKS Proxy
□ Streaming Proxy (RTSP)
□ Gopher Proxy

Secure Web Proxy Server

https://localhost : 443

□ Proxy server requires password

Username:

Password:

Your credentials may be sent unencrypted

□ Exclude simple hostnames

Bypass proxy settings for these Hosts & Domains:

localhost

□ Use Passive FTP Mode (PASV)

Figure 10: CovertNet proxy setup.

FUNCTIONALITY/RESULTS

The functionality of CovertNet at this stage of development is extensive, although not complete. I was able to access several major websites using CovertNet as a proxy with Google Chrome, including Google and Wikipedia. Many other websites do not connect, such as Facebook and many smaller websites. The reason for this is uncertain. I suspect it may have something to do with the SSL certificates issued; the SSL certificates were purchased cheap and do not have the same level of trustworthiness as the certificates used by major websites, which cost hundreds of dollars. However, the ability to access websites like Google is very promising; it means the mechanism works to some extent. One of the major issues with CovertNet project is that in order to fully test its functionality, it must be tested under the surveillance of a large censorship network such as the GFW. Until then, its true ability to circumvent web censorship is uncertain.

With regards to speed, the CovertNet protocol operates on the slow end of normal. The loading speeds of websites was measured using Google Chrome's built in console tools. Most major websites took around 4-5 seconds to upload. Google took 3.91 seconds. Wikipedia took longer, at 8.21 seconds. Twitter came in at 4.25 seconds. This is rather slow, but nevertheless functional.

My implementation shows that it is possible to build a covert-channel based network protocol that offers decent functionality. There are many improvements that can be made upon CovertNet. For one thing, it would be of great interest to develop a protocol that does not require users to purchase a domain and SSL certificate. This may involve taking an entirely different approach to developing CovertNet, for example, implementing it as a web extension rather than a standalone program. Another issue that may arise in the further development of protocols such as CovertNet is distribution. This issue arises by virtue of the fact that CovertNet is intended as a threat to web censorship programs, and thus cannot simply be distributed freely under those programs, unless one were to download it someplace else and then use it under the censorship program. One approach is to limit distribution to physical media such as CD-ROM's, although this approach may be considered outdated. Another approach is that of bootstrapping, whereby the protocol can be covertly downloaded upon connection to a CovertNet server.

In conclusion, CovertNet shows that covert channels may be a realistic solution to the problem of circumventing web surveillance, although more work will be needed on CovertNet for it to provide full internet functionality. They provide a solution to some of the problems that VPN's and outside proxies have and offer similar bandwidth. In addition, CovertNet shows that covert channels can be tailored to adapt to specific censorship programs, such as the Great Firewall. In the future, covert-channel based protocols can be developed that are specific to the censorship programs of other countries as well. Future work is also needed to provide covert means of distribution and increased portability of the protocol. Protocols with different channels can be developed to determine which are most covert and offer the best Internet speed and functionality. I hope that CovertNet will ultimately increase the capabilities of those across the world to access the Internet, and inspire more research into the circumvention of web censorship using covert channels.

<u>References</u>

[1]Lampson, B.W. (1973) “A note on the confinement problem,” Communications of the ACM, vol. 16, no.10, pp. 613–615.
[2]U. S. A. Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, ser. Defense Department Rainbow Series. Fort George G. Meade, Maryland: Department of Defense / National Computer Security Center, December 1985, no. DoD 5200.28-STD.
[3] Brown, E., Yuan, B., Johnson, D., Lutz, P. (2009). “Covert channels in the HTTP network protocol: Channel characterization and detecting man-in-the-middle attacks.” In: Proc. 5th Intern. Conf. Information Warfare and Security. Ohio, USA. 2010, pp. 56–65.
[4] Okhravi, H., Bak, S., King, S. “Design, Implementation, and Evaluation of Covert Channel Attacks.” IEEE International Conference on Technologies for Home Security (HST).
[5] Cioranesco, J., Ferradi, H., Geraud, H., Naccache, H. “Process Table Covert Channels: Exploitation and Countermeasures.” Research, École normale supérieure, Équipe de cryptographie, Paris, France. March 2016.
[6] M. Smeets and M. Koot, “Research report: Covert channels,” Master’s thesis, University of Amsterdam, February 2006.
[7] Jaskolka, J. and Khedri, R. “Exploring Covert Channels.” Proceedings of the 44th Hawaii International Conference on System Sciences, 2011.
[8] Mileva, A., Panajotov, B. “Covert Channels in TCP/IP Protocol Stack (Extended Version).” Central European Journal of Computer Science. June 2014.
[9] Rowland, C. H. “Covert channels in the TCP/IP protocol suite, DoIS Documents in Information Science.” May 1997.
[10] Nagatou, N., Watanabe, T. “Run-time detection of covert channels.” The First International Conference on Availability, Reliability and Security, 2006.
[11] R. Kemmerer, “Shared resource matrix methodology: An approach to identifying storage and timing channels,” ACM Transactions on Computer Systems, vol. 1, no. 3, pp. 256 – 77, August 1983.
[12] R. Kemmerer and P. Porras, “Covert flow trees: A visual approach to analyzing covert storage channels,” IEEE Transactions on Software Engineering, vol. 17, no. 11, pp. 1166 – 1185, November 1991.
[13] Faris, R., Villeneuve, N. “Measuring Global Internet Filtering.” in *Access Denied: The Practice and Policy of Global Internet Filtering*.
https://opennet.net/sites/opennet.net/files/Deibert_02_Ch01_005-028.pdf
[14] "Global Internet User Survey 2012" Archived 14 March 2013 at the Wayback Machine, Internet Society, 20 November 2012
[15] “OpenNet Initiative Global Internet Filtering Map.” part of the OpenNet Initiative.
https://onimap.citizenlab.org/
[16] Scott Helme, “Alexa Top 1 Million Analysis - August 2018.”
https://scotthelme.co.uk/alexa-top-1-million-analysis-august-2018/
[17] Riley, C. “Anarchy, State, or Utopia? Checks and Balances of Power in Internet Governance.” Independent. March 2, 2013.
[18] Elmer-Dewitt, P. “First Nation in Cyberspace.” TIME Magazine. December 6, 1993.

[19] Son, S., Shmatikov, V. "The Hitchhiker's Guide to DNS Cache Poisoning." International Conference on Security and Privacy in Communication Systems: SecureComm 2010: Security and Privacy in Communication Networks pp 466-483.
[20] OpenNet Initiative. https://opennet.net/
[21] OpenNet Initiative Research - Region Profile, United States and Canada. https://opennet.net/research/regions/united-states-and-canada
[22] Zittrain, J., Edelman, B. "Internet Filtering in China." IEEE Internet Computing. March 2003.
[24] "The Great Firewall of China: Xi Jinping's Internet Shutdown." The Guardian. June 29, 2018.
[25] The World Bank. "Individuals Using the Internet (% Of Population) - China". https://data.worldbank.org/indicator/IT.NET.USER.ZS?locations=CN
[26] Dillinger, J. World Atlas. "List Of Countries By Internet Users". https://www.worldatlas.com/articles/the-20-countries-with-the-most-internet-users.html
[27] Anderson, D. "Splinternet Behind the Great Firewall of China." Association of Computing Machinery, Web Security, Volume 10, Issue 11. November 20, 2012. https://queue.acm.org/detail.cfm?id=2405036
[28] Park, J. C., Crandall, J. R."Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China." 30th International Conference on Distributed Computing Systems, Genoa, Italy. June 21-25, 2010.
[29] Yan, B., Fang, B., Li, B., Wang, Y. 2006. Detection and defense of DNS spoofing attacks. *Computer Engineering* 32(21).
[30] Tang, C. "In-depth Analysis of the Great Firewall of China." Department of Computer Science, Tufts University. December 14, 2016. https://www.cs.tufts.edu/comp/116/archive/fall2016/ctang.pdf
[31] Arthur, C. "China tightens "Great Firewall" Internet Control with New Technology." The Guardian, December 14, 2012.
[32] Winstein, K. J. "China blocks MIT Web addresses." 2002 *The Tech* 122(58); http://tech.mit.edu/V122/N58/58web.58n.html.