2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics

A Robust Protocol for Circumventing Censoring Firewalls

Junlin Chen
Department of Electrical Engineering & Computer Science
York University
Toronto, CANADA
chen256@cse.yorku.ca

Uyen Trang Nguyen
Department of Electrical Engineering & Computer Science
York University
Toronto, CANADA
utn@cse.yorku.ca

***Abstract*—Internet censorship infringes the right of free speech, invades people's privacy and oppresses citizens who have different political or religious views from those of the censors. Existing solutions to circumvent censoring firewalls suffer from one or more weakness: slow, prone to failure or easy to be detected and blocked. We propose a protocol to circumvent censoring firewalls that is distributed, robust, reasonably fast and more resistant to existing detection mechanisms.**

***Keywords*-firewalls; Internet censorship; proxy servers; distributed network; content filtering; deep packet inspection; black lists;**

I. Introduction

The use of Internet firewalls is common all around the world. On one hand, Internet firewalls enforce the security of a network and protect its users from harmful materials (e.g., spam, malware, pornography). On the other hand, firewalls have been used in many countries as a from of censorship. In this case, Internet censorship infringes the right of free speech, invades people's privacy, and oppresses citizens who have different political or religious views from those of the censors.

Therefore, many circumvention tools have been deployed to bypass Internet firewalls. The main idea behind these tools is to use alternative paths for data packets, so that a firewall cannot identify the packets and let them go through. These alternative paths traverse one or more proxy servers in order to hide the original source of the data packets, bypassing the blocking mechanism of the censoring network.

A. How Do Censoring Firewalls Work?

Firewalls identify packets for blocking using three main approaches: (1) blacklisting IP addresses (or domain names, or URLs); (2) deep packet inspection; and (3) machine learning algorithms.

There are some trade-offs for using IP addresses (or domain names, or URLs) to block Internet traffic. The more exhaustive the black list is, the more computational resources the firewall needs to search and to maintain the list. If the black list is perfunctory and broad, or if the IP address listed in the white list is allowed to bypass the firewall, the network behaves more like a local network without Internet access.

Deep package inspection (DPI) is a typical technique used by censoring firewalls. Many tools (e.g., IPSec, PPTP, Tor) have unique packet characteristics that are very easy to identify [1][2][3]. Deep package inspection also includes inspections of the the packet content. If specific keywords appear in the packets, the content censoring firewall may block these packets.

Recently, machine learning algorithms are implemented in many content censoring firewalls for detecting patterns of specific proxy programs [4][5]. This approach is the most complex and consumes a lot of computational resources.

In addition to the above three approaches, many firewalls also keep track of connection duration. Such a firewall would detect long active connections and terminate them.

In addition to blocking data packets, some censoring firewalls may decide to terminate the whole connection by resetting the TCP connection. The firewall returns a forged TCP packet with the RST flag set to the client, which makes the client stop receiving data. On the other side, the firewall send a forged TCP packet with the FIN flag set to the server to force the server to close the connection.

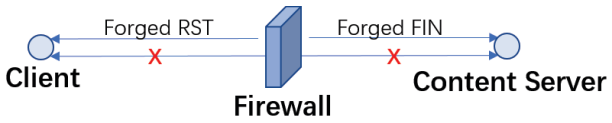

Figure 1. Forged TCP packets terminate the connection.

B. Existing Circumvention Solutions

Existing circumvention tools can be classified in to three broad categories (Fig. 2):

1) *Proxies in the Free Internet*: This is the most common method to circumvent censoring firewalls that prevent citizens from reaching certain websites (e.g., Wikipedia, Twitter). One or more proxy servers are needed between the firewall and the blocked website. The client manages to reach the proxy server located on the other side of the firewall, which relays data packets from the blocked website to the client.

978-1-5386-7975-3/18/$31.00 ©2018 IEEE
DOI 10.1109/Cybermatics_2018.2018.00299

2) *Proxies in the Censored Network*: This method is usually deployed by organizations, companies and individuals that want to make their blocked websites available to people under censorship behind a firewall. They set up proxy servers (usually illegally) inside the censored network to bypass the firewall.
3) *Virtual Private Networks (VPN)*: VPN servers can be used as forward proxy servers. People using VPN as a proxy are actually connected to an open public network, the Internet, rather than a "private network".

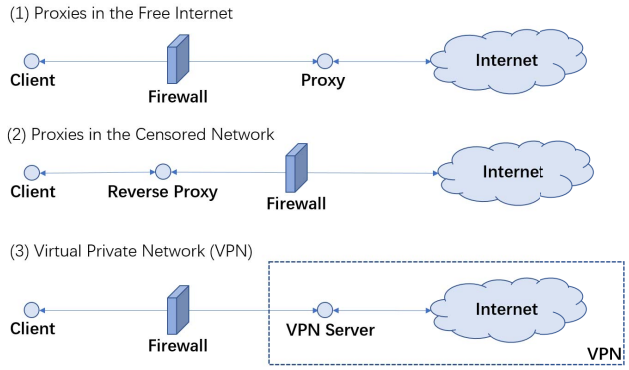

Figure 2. Categories of circumvention tools

C. Weaknesses of Existing Solutions

Most of the tools for circumventing firewalls (e.g., IPSec, PPTP, L2TP, HTTP/HTTPS Proxy and SSH tunnel) are vulnerable to single points of failure due to one single proxy server between the firewall and the banned website (see Fig. 2). Only one single server also means only one connection between the firewall and the free Internet. If the connection is active for a long time (persisting connection), this makes the connection easy to be detected and blocked.

Some proxy protocols (e.g., SOCKS) use some particular handshake procedure, which can be easily identified by firewalls [5].

Some tools (e.g., HTTP/HTTPS Proxy) incur significant data overhead, which slows down the connection speed and negatively impacts user experience.

It is also worth mentioning Tor, a well-known distributed network (Fig. 3). Although Tor can avoid the problem of single points of failure, it needs "bridge peers" to bypass a firewall. Tor maintains a global list of relay nodes that is publicly available. Firewalls can use this list to block those relay nodes. Hence, available unblocked bridge peers are rare resources of a Tor network, and thus can become the bottleneck of a connection, slowing down its speed. Furthermore, many users are not willing to relay data for other people, making bridge peers even a rarer resource.

Currently, Tor requires users to obtain the "bridges" manually by visiting their website, which has usually been black listed, and solving a RECAPTCHA [6]. Improvements on Tor include the addition of a pluggable layer (e.g., Obsf4, Meek) on top of Tor to obfuscate the proxy traffic [7]. This is a sophisticated solution that incurs very high overhead. Generally speaking, Tor is designed for people to communicate anonymously rather than circumventing firewalls. Finally, criminals have used Tor for illicit purposes; thus many websites (e.g., Wikipedia) and content delivery networks (e.g., Cloudfare) have blocked Tor's exit nodes, which prevents legitimate users from using their services.

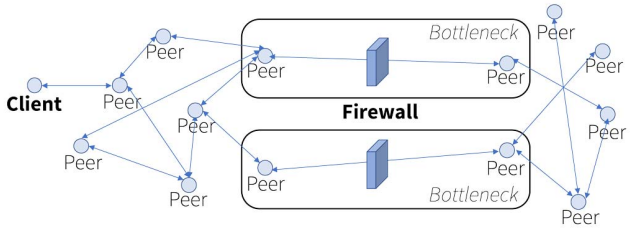

Figure 3. Tor network to bypass firewalls

D. Contribution of this Paper

We propose a distributed proxy system that addresses the above weaknesses of existing solutions. Multiple proxy servers and deployed and serve a connection (see Fig. 4). The packet relaying task is randomly divided among the proxies, forming multiple shorter lived connections and thus avoiding the "persisting connection" problem. Thanks to multiple proxies and random packet distribution among them, there is no bottleneck in our system. If one or more proxies are detected and blocked by the firewall, the remaining proxies should still work, avoiding the problem of single point of failure. Furthermore, the user should not experience any interruption, because the lost packets (relayed previously by detected and blocked proxies) will be automatically retransmitted by other proxies through other routes.

To circumvent detection and protect user privacy, the proxy traffic is encrypted, but encapsulated in data units of non-encrypted protocol (e.g., HTTP), disguising as non-encrypted regular traffic. Our proxy system minimizes the trivial packet characteristics to make traffic identification computationally expensive to firewalls. Furthermore, the system introduces random factors into the traffic to increase the difficulty of pattern recognition. Our proxy system uses a handshake procedure at the beginning of data relaying in order to establish a shared key to be used for encryption proxy traffic. Once the proxy connections are established, the steps of the handshake procedures of the upper layer protocols (e.g., HTTPS, SOCKS) are randomly distributed over different proxy connections to make detection more difficult.

E. Outline of the Paper

The remainder of the paper is organized as follows. In Section II, we describe the design and implementation of our circumvention proxy system. In Section III, we present a performance analysis, comparing our proposed proxy system with existing tools such as IPSec and Shadowsocks. We conclude the paper and outline future work in Section IV.

II. Design And Implementation

We design our circumvention system to address the following weaknesses of existing tools:

- single points of failure
- persisting connections
- low speed caused by bottlenecks
- ease of detection
- violation of user privacy

A. Architecture of the Proposed Proxy System

Our proposed proxy system is based on the first circumvention approach, *proxies in the free Internet*. As shown in Fig. 4, the client application is located on one side of the firewall (in the censored network). The server application and the SOCKS5 server are on the other side of the firewall (in the free Internet).

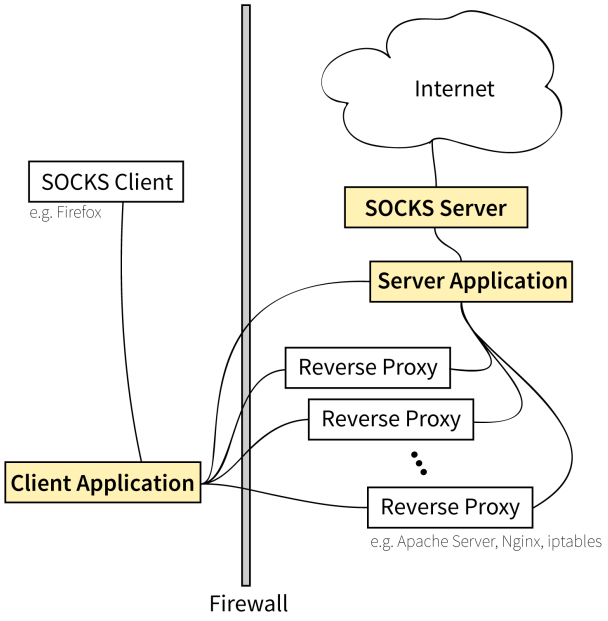

Figure 4. Structure of the proxy system

We have chosen SOCKS to implement of the system because this proxy protocol is widely supported by many browser and web applications. SOCKS can also wrap all common packets with low overhead compared to other existing protocol (e.g., HTTP). SOCKS does not interpret the network traffic between the client and the server, which minimizes the use of computational resources. Furthermore, SOCKS does not modify original TCP packets, thus preserving the integrity of the data.

The client application is installed on the user's computer. The user can use any software that supports SOCKS5 protocol as a SOCKS client to connect to the client application.

Several reverse proxy servers are set up on the other side of the firewall to relay the data between the client application and the server application on different addresses (e.g., IP addresses, port numbers). In cases where some of the addresses are blocked by the firewall, there are still available interfaces to support the connection.

The server application listens on multiple interfaces; hence it can accept connections from the client through different routes. The client can have direct or indirect connections to the server application through different routes.

The server application connects to the SOCKS5 server application based on the request from the client application. The SOCKS5 server analyses the SOCKS5 protocol and sends corresponding request to the Internet. The SOCKS5 server acts as the exit point of this proxy network. We can only select one exit point for a client. If we chose multiple exit points which are located at different geo-locations, this may confuse the content distribution network (CDN) when the user is browsing web pages.

Both the client application and the server application maintain a buffer of transmitted packets. If some packets are lost while passing through the firewall, they can be retransmitted. Both the client and the server application maintain a buffer of received packets, which are sorted before being delivered to the upper layer. (Data packets may arrive out of order because they take different routes through different proxy servers.)

B. Distributed Anti-interference Proxy (DAIP) Protocol

Our proxy system requires a new protocol named distributed anti-interference proxy (DAIP) protocol that sits between the SOCKS protocol and a non-encrypted application protocol (e.g., HTTP) in the protocol stack, as shown in Fig. 5. DAIP is needed for the following reasons:

1) DAIP protocol data units (PDUs) are encrypted to avoid packet inspection or content filtering, and to maintain data integrity and user privacy.
2) Using encrypted DAIP PDUs, we can identify whether a connection is routinely closed by the content provider (e.g., Wikipedia), or forcefully and prematurely closed (reset) by the firewall (see Section I-A). When the content provider sends a TCP message to close the connection (e.g., after completing the transmission of a requested document), the legitimate TCP message (with the FIN flag set) will be passed to the SOCKS server. The SOCKS server then forwards the TCP message to the server application, which then generates a message (signal) to ask the client application to close

the connection. When the client application receives the DAIP PDU, it will extract the connection termination message. The client application then closes the connection. On the other hand, a forceful TCP connection termination message sent by the firewall would not be encapsulated by DAIP, helping the client application to identify such forged TCP messages. (The same explanation applies to a TCP connection termination message going from the client or the firewall to the content provider site.)

DAIP PDUs are encrypted and hidden in a message generated by a dummy non-encrypted application protocol such as HTTP (see Fig. 5). The non-encrypted traffic of the dummy HTTP application will confuse the firewall in order to make detection more difficult.

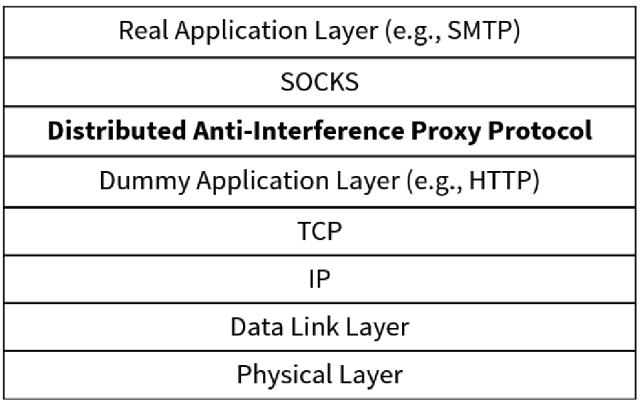

Figure 5. DAIP in the protocol stack

Following is a description the DAIP PDU. As shown in Fig. 6, the PDU contains a header and payload. The payload is the actual proxy data. The header contains some redundancy for error checking. The PDU header has the following fields; each is 16-bits long.

1) *Dispatcher ID*
   The server application maintains multiple connections to the SOCKS server. The client application also maintains multiple connections to the SOCKS client. A dispatcher ID is needed to identify the corresponding connections on both sides.
2) *Type*
   This field defines the behavior of the payload. It has the following types.
   a) Connection establishment
   b) Connection termination
   c) Data packet
   d) Acknowledgment
3) *Sequence Number*
   Because the data packets are relayed over the network through different proxies and routes, the recipient may receive them out of order. This field is needed to reorder packets arriving out of order. Sequence numbers are also needed to identify lost packets (e.g., due to some proxies being detected and blocked by the firewall) for retransmission by the SOCKS server.
4) *Size*
   This is the size of the payload.

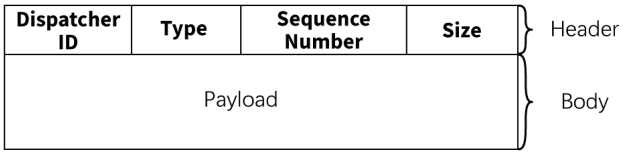

Figure 6. Distributed Anti-Interference Proxy Protocol

The size of all the fields is longer than actually needed. This allows for some redundancy to help with error checking. For example, only two bits are need for the "Type" field, to indicated four types of PDU. If the received value is not one of the four possible values, this means that there is an error in the header.

The DAIP protocol uses cipher feedback (CFB) mode of AES for encryption. AES is a commonly used encryption algorithm. The cipher feedback (CFB) mode allows us to encrypt and decrypt the data on streaming without padding the data to a multiple of the cipher block size. While decryption the data using CFB mode, because the cipher has feedback from the ciphertext, an error, if any, is carried on to the next PDU. Hence, we can detect the transmission error when we get the next PDU header. Furthermore, the PDU header is designed to have some redundancy for error checking as discussed above.

Both the header and the body of the PDU are encrypted The key and initial vector for encryption are exchanged and confirmed by the client and server during a DAIP handshake procedure before data transmission starts.

C. Connection Life-cycle

The life cycle of a connection is shown in Fig. 7. At the beginning, the client and server applications authenticate each other (steps "Verify" and "Confirm") and establish a shared key to be used from encrypting DAIP data units, as discussed above. After that they can send and receive data ("Read & Write").

If a connection is idle for a long time (determined by a preset timer), the proxy system will close it to avoid the "persisting connection" problem. After a predetermined amount of time, the connection is re-established. In short, all connections through our proxy system are active intermittently to avoid detection.

D. Data Transmission

When the client has data to send to the server, the client application reads the data from the SOCKS client. If the

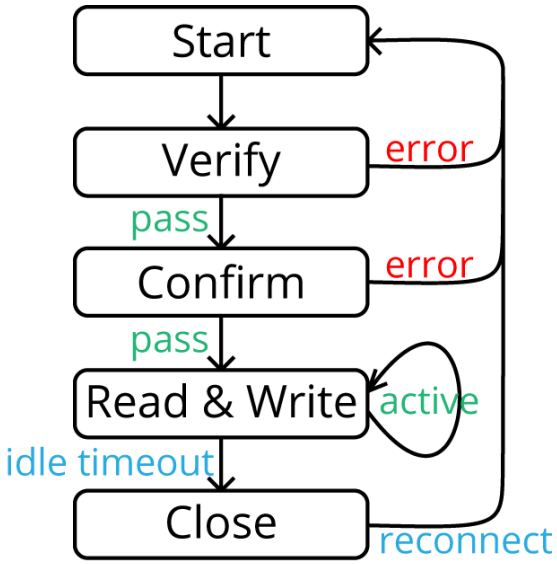

Figure 7. Connection life-cycle

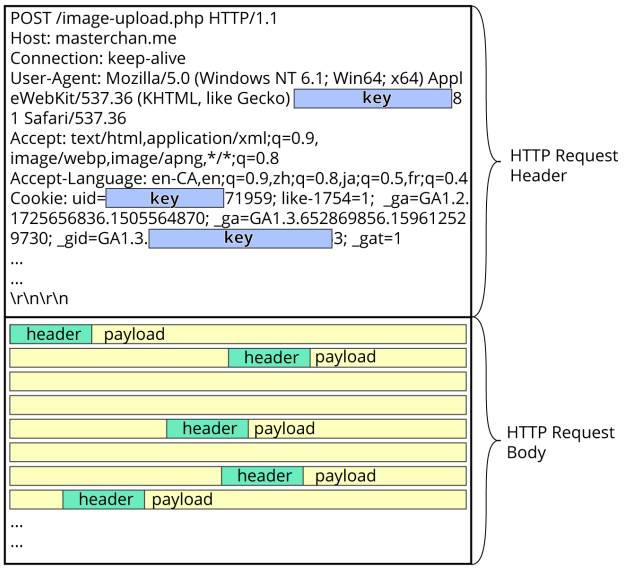

Figure 8. Dummy HTTP message

data is large enough (e.g., its size greater than xx bytes), the client application segments the data into multiple units, stores a copy of each unit in a buffer (for re-transmission when needed), adds a DAIP PDU header, encrypts the unit, and randomly selects one available active proxy connection to send the unit over. (There exist multiple proxy connections between the client and server applications to avoid the "single point of failure" and "persisting connection" problem, as mentioned earlier.) The server application follows the same procedure when it has data to send.

DAIP data units received correctly are acknowledged by the recipient (similarly to TCP acknowledgments). Unacknowledged packets are retransmitted and buffers are managed in a similar manner to TCP operations. In order to keep the overhead low, the recipient uses cumulative acknowledgment as in TCP, i.e., one message acknowledging multiple consecutive data packets correctly received. In cases where a proxy connection is interrupted by the firewall via TCP resetting/termination, the sender is able to detect this intervention by the firewall as discussed in II-B-2. The sender then retransmits the packet via another proxy connection. This design adds another random factor to the system to increase the difficulty of pattern recognition.

DAIP PDUs sent out by the client application are hidden in the body of a "HTTP request", as shown in Fig. 8. Similarly, DAIP PDUs sent out by the server application are hidden in the body of a "HTTP response".

DAIP packets are randomly distributed among all the available proxies. The quality of the randomness and the number of available proxies determine the entropy of the data and the level of difficulty for detection.

III. Performance Analysis

We choose two typical circumventing tools, IPSec and Shadowsocks, for the analysis. We also compare our system with direct connections (no circumventing tool or proxy).

A. Numerical Analysis

The most common type of local networks on the Internet is Ethernet, which has a maximum size of 1500 bytes. Subtracting 20 bytes for an IP header and 20 bytes for a TCP header, we assume a payload of 1460 bytes in our numerical analysis.

1) IPSec: IPSec is widely supported by many devices and has different implementations. In this analysis, we choose one of the most common encryption methods used in IPSec, which is AES cipher block chaining (CBC). The cipher block chaining mode requires a message be padded to a multiple of the cipher block size [8][9]. Following are the overheads incurred by IPSec:

- AES Padding: 12 bytes
- Padding Identifier: 1 byte
- SHA-1 Message Length: 8 byte
- SHA-1 Padding: 55 bytes

- ESP Tunnel Mode Header: 20 bytes
- ESP Header: 8 bytes
- ESP Initialization Vector: 16 bytes
- ESP Trailer: 16 bytes

The overhead percentage for a 1460-byte payload is thus

$$\frac{12+1+8+55+20+8+16+16}{1460} = 9.32\% \quad (1)$$

2) Shadowsocks: Shadowsocks is one of the most popular proxy software and uses encryption above SOCKS. The SOCKS protocol itself does not alternate the traffic among multiple proxy connections, but it requires one handshake per connection (for connection request and authentication) [10]. The cost of a handshake can go up to 519 bytes, which is the sum of the following fields in a SOCKS PDU header.

- Version Identifier − Version: 1 byte
- Version Identifier − Number of Methods: 1 byte
- Version Identifier − List of Methods: 1 to 255 bytes
- Request − Version: 1 byte
- Request − Command: 1 byte
- Request − Reserve: 1 byte
- Request − Address Type: 1 byte
- Request − Address: up to 256 bytes
- Request − Port: 2 bytes

Assuming that each connection carries P packets, we obtain the overhead percentage as follows:

$$\frac{1+1+255+1+1+1+1+256+2}{1460} \times \frac{1}{P} = \frac{35.54\%}{P} \quad (2)$$

Assume that a SOCKS connection carries 50 packets ($P = 50$). Then the overhead incurred by Shadowsocks is

$$\text{overhead} = \frac{35.54\%}{50} = 0.71\% \quad (3)$$

3) Distribute Anti-Interference Proxy Protocol: The DAIP protocol incurs the following overheads:

- Each DAIP PDU has a header of 8 bytes long (Fig. 6). Each DAIP acknowledgement (ACK) message is thus 8 bytes long (empty body, with the ACK field set).
- Assume that each ACK message acknowledges on average three data PDUs. That is, the probability of sending acknowledgements is $\alpha = 30\%$.
- The average cost of a DAIP handshake is H bytes. Assume that the total number of DAIP PDUs generated by the client-server connection is N. Then the average cost of a DAIP handshake *per PDU* is H/N.
- Since the SOCKCS protocol runs on top of the DAIP protocol, we must include overheads incurred by SOCKS. The SOCKS handshake needs two DAIP PDUs, or 8 bytes $\times 2 = 16$ bytes for the DAIP PDU headers, plus 519 bytes of the SOCKS PDU header as calculated above.

Assume that a SOCKS connection carries P packets. Then the overhead incurred by the DAIP protocol is

$$\frac{8+8\alpha+\frac{H}{N}+\frac{16+519}{P}}{1460} \quad (4)$$

The DAIP handshake information such as cryptographic keys are hidden in a non-encrypted protocol such as HTTP. For example, in Fig 8, the keys are hidden in the header of a HTTP message. In this case, the dummy HTTP request takes approximately 300 bytes ($H = 300$). Assuming $N = 100$, $P = 50$ and $\alpha = 30\%$, we can calculate the DAIP overhead percentage as follows:

$$\frac{8+8\times 0.3+\frac{300}{100}+\frac{16+519}{50}}{1460} = 1.65\% \quad (5)$$

Table I
COMPARE OVERHEAD

Proxy	Overhead
No Proxy	0.00%
IPSec	9.32%
Shadowsocks	0.71%
DAIP	1.65%

B. Experimental Results

We ran experiments to compare our proxy system with IPSec, Shadowsocks, and direct connections. The experiments were performed on a real network. The structure of the network is shown in Fig. 9. The delay between the user and a proxy server is approximately 190 ms. The delay between a proxy server and the content provider is approximately 100 ms. The delay of a direct connection between the user and the content provider is approximately 120 ms. These delay values were measured using the ping command. (The actual delay could be shorter during data transmission, because the firewall might have slowed down the first couple of packets.)

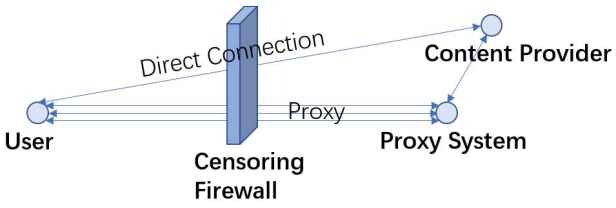

Figure 9. Architecture of the testing network

We varied the size of the files to be downloaded from the content provider, and measured the download time for each file size. The experimental results are illustrated in Fig. 10.

As the graph shows, the DAIP protocol is faster than IPSec, because DAIP does not need to pad the data to a specific size for encryption. However, DAIP is slower than Shadowsocks, a single-proxy tool. In DAIP, packets

are relayed on different routes through different proxies, and thus may arrive at the destination out of order. This requires packets to be sorted before being delivered to the higher layer. In addition, DAIP retransmits lost packets in cases of connection time-out or interruption. Due to these two main factors, DAIP is slower than Shadowsocks. However, that is the cost to pay for higher robustness and resistance to detection, which is lacking in Shadowsocks.

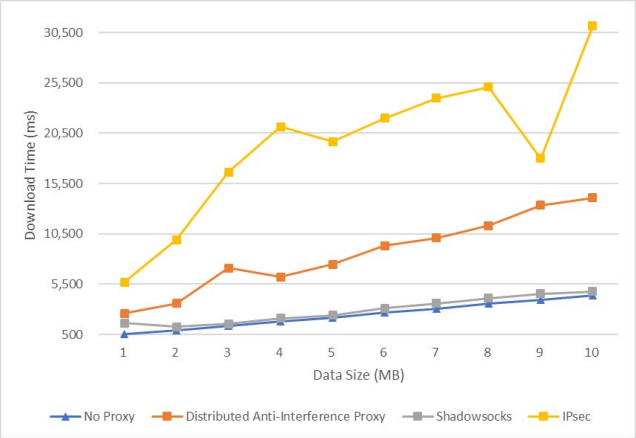

Figure 10. Architecture of the testing network

IV. Conclusion

We propose a distributed proxy system to circumvent censoring firewalls. Data units are randomly distributed to different proxies and routed though different paths to avoid detection, single points of failure, persisting connections, and bottlenecks. To circumvent detection and protect user privacy, data units of the DAIP protocol are encrypted and hidden in dummy HTTP messages.

Our preliminary numerical analysis and experiments results show that our proxy system is faster than IPSec. It incurs slightly more overhead and download time than Shadowsocks (and probably some other circumvention tools). This is the price to pay for better robustness, detection avoidance and user privacy.

It is worth noting that circumvention tools are double-edged swords: criminals may use these tools, including the scheme we propose, to bypass firewalls of companies, organizations or government offices. Our goal in this paper is to provide a tool for circumventing censorship; the task of protecting their networks belongs to those organizations.

V. Future Work

We are working on several issues to improve the performance and robustness of DAIP. For instance, client and server applications exchange keys by hiding the keys in the header of dummy HTTP messages (Fig. 10). Censors will eventually figure out this mechanism and implement some form of pattern recognition in their firewalls. Better methods to overcome this issue are needed in future versions of DAIP. We are also looking for improvements to make the entire proxy network flow in a manner that is closely matched the flow of a non-encrypted non-proxy network to avoid detection.

Another issue to consider is to avoid detection caused by the use of the same set of IP addresses within a short amount of time. One solution to avoid this type of detection is to set up a sufficiently large number of proxies and alternate among them. Deploying a large number of proxies is easy and inexpensive these days thanks to readily available cloud services. We simply need to configure one proxy server and copy the server instance to other proxies. The number of proxies to be deployed depends on the number of people using the circumvention service. We leave this analysis to future work.

References

[1] M. Zhuli, L. Wenjing and G. ZhiPeng, “Context Based Deep Packet Inspection of IKE Phase One Exchange in IPSec VPN,” 2010 International Conference on Innovative Computing and Communication and 2010 Asia-Pacific Conference on Information Technology and Ocean Engineering, Macao, 2010, pp. 3-6.

[2] K. Kazemi and A. Fanian, “Tunneling protocols identification using light packet inspection,” 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), Rasht, 2015, pp. 110-115.

[3] F. A. Saputra, I. U. Nadhori and B. F. Barry, “Detecting and blocking onion router traffic using deep packet inspection,” 2016 International Electronics Symposium (IES), Denpasar, 2016, pp. 283-288.

[4] J. M. Reddy and C. Hota, “Heuristic-Based Real-Time P2P Traffic Identification,” 2015 International Conference on Emerging Information Technology and Engineering Solutions, Pune, 2015, pp. 38-43.

[5] Z. Deng, Z. Liu, Z. Chen and Y. Guo, “The Random Forest Based Detection of Shadowsock’s Traffic,” 2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, 2017, pp. 75-78.

[6] Tor Project, “Tor Project: Bridges” [Online]. https://www.torproject.org/docs/bridges

[7] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver and V. Paxson, “Examining How the Great Firewall Discovers Hidden Circumvention Servers”, Internet Measurement Conference, ACM, 2015

[8] Frankel, S., Glenn, R., and S. Kelly, “The AES-CBC Cipher Algorithm and Its Use with IPsec”, RFC 3602 [Online]. https://tools.ietf.org/html/rfc3602

[9] Kent, S., “IP Encapsulating Security Payload (ESP)”, RFC 4303 [Online].https://tools.ietf.org/html/rfc4303

[10] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, ”SOCKS Protocol Version 5”, RFC 1928 [Online].https://tools.ietf.org/html/rfc1928