GoHop: Personal VPN to Defend from Censorship

Yuzhi Wang*, Ping Ji†, Borui Ye‡, Pengjun Wang*, Rong Luo*, Huazhong Yang*

* Department of Electronic Engineering, Tsinghua University, China
† NEMO Research Lab, City University of New York, USA
‡ School of Computer Science, Beijing University of Posts and Telecommunications, China
Corresponding Author: yz-wang12@mails.tsinghua.edu.cn

***Abstract*—Internet censorship threatens people's online privacy, and in recent years, new technologies such as high-speed Deep Packet Inspection (DPI) and statistical traffic analysis methods had been applied in country scale censorship and surveillance projects. Traditional encryption protocols cannot hide statistical flow properties and new censoring systems can easily detect and block them "in the dark". Recent work showed that traffic morphing and protocol obfuscation are effective ways to defend from statistical traffic analysis. In this paper, we proposed a novel traffic obfuscation protocol, where client and server communicate on random port. We implemented our idea as an open-source VPN tool named *GoHop*, and developed several obfuscation method including pre-shared key encryption, traffic shaping and random port communication. Experiments has shown that GoHop can successfully bypass internet censoring systems, and can provide high-bandwidth network throughput.**

***Index Terms*—VPN, privacy protection, random port, traffic morphing, protocol obfuscation, censorship circumvention**

I. Introduction

Internet censorship is a threat to people's online privacy and their freedom to get information. There're some softwares, including Tor [1], SSH Tunnel, VPN, etc., which can provide encrypted tunnel helping people to bypass censoring systems. These softwares used to be effective because censoring systems could only inspect packets with plaintext.

However, with the improvement of traffic analysis techniques, it's possible for the censoring authority to extract sensitive information from encrypted packets since encryption does not change packet flow's statistical properties such as packet size, arrival time and direction.

Previous work has shown that with statistical information leaked from encrypted traffic, it's possible to detect traffic class and other information from encrypted traffic. Wright et al. [2] and White et al. [3] showed that an observer can identify key phrases or even recognize words from encrypted Voice-over-IP (VoIP) conversation since packet sizes from variable-bit-rate (VBR) voice encoder are related to the words in the speech. Dusi et al. [4] showed that network administrators can use traffic's statistical fingerprints to detect whether an application protocol is used as a tunnel to encapsulate another illegitimate protocol. Wright et al. [5] has shown that utilizing machine-learning techniques, it is not only possible to identify the protocol of an encrypted flow, but also track the number of TCP connections inside an IPSec VPN tunnel. Murdoch et al. [6] presented traffic analysis techniques enabling adversaries to detect Tor nodes and track the anonymous stream back to the initiator.

Traffic analysis techniques can not only be utilized to identify security threats, but also violates user privacy [7]. Moreover, it enables censoring authorities to detect whether an user is using anti-censorship softwares to visit blocked websites [8].

Most widely used anti-censorship software still use only encryption to protect plaintext leakage, and with traffic analysis techniques being put into practice, some anonymous network protocols were blocked [9, 10].

In order to prevent statistical property leakage, flow obfuscation protocols were introduced. Tor is one of the anonymous network protocols got blocked in many ISPs, so Tor team developed a framework named pluggable transport [11] to handle traffic obfuscation. The Tor team and community developed several traffic obfuscation tools for Tor and some of them has been released to the public. Obfsproxy [12] is the first pluggable transport implementation which transforms Tor traffic to innocent-looking traffic using encryption. Flashproxy [13] uses browsers to proxy Tor traffic, and it switches its upstream peer frequently so that it's hard to be detected by the adversary. SkypeMorph [14] implemented traffic morphing for Tor and disguise Tor traffic to Skype VoIP traffic. Shadowsocks [15] is a cross-platform SOCKS proxy which uses pre-shared key to encrypt in tunnel traffic. Blond et al. [16] proposed a novel anonymity network named Aqua as a replacement for Tor, which can provide both anonymity and high-bandwidth. Aqua is still in early development and can only support file-sharing rather than common Internet usage.

In this paper, we proposed *GoHop*, a VPN software with innate traffic obfuscation features. Unlike Tor which provide SOCKS proxy, GoHop provides a VPN, so that users need not configure browsers or other internet applications. We developed several obfuscation approaches for GoHop, including pre-shared master key, traffic shaping and random port communication, detailed description will be presented later.

The main contributions of this paper are:

- **VPN with traffic obfuscation**: At the time of writing, GoHop is the only published VPN software with build-in traffic obfuscation.
- **Simple but effective traffic obfuscation protocol**: We designed a flexible flow obfuscation framework and developed several obfuscation methods. Each of them is quite simple and easy to understand, but experiment has shown that they are very effective. Furthermore, because of GoHop's simplicity, it runs much faster than other censorship circumvention softwares.
- **Ready-to-use implementation**: We published GoHop under an open-source license [1], GoHop currently supports Linux operating system, and we have been using it to bypass censorship for several months.

The rest of the paper is organized as follows. In Section II, we first present our threat model, assumptions and GoHop's design goal. In Section III, we introduce GoHop's obfuscation methods, and implementation details comes in Section IV. Experiment results are shown in Section V. In Section VI, we discuss future work and conclude in Section VII.

[1] https://github.com/bigeagle/gohop

II. Assumptions and Design Goals

A. Threat Model and Assumptions

Our threat model is similar to other censorship circumvention tools [13], there are 4 objects in our model:

- **Client**: The client-side GoHop program, which locates in the region where Internet access is under censorship.
- **Server**: The server-side program, whose ISP does not filter Internet contents.
- **Adversary**: The censoring authority, who can inspect every packet get through its network boundary.
- **Target**: An Internet peer, such as a website, which is blocked in the region where client locates, but can be accessed by the server.

We assume that the adversary does not intend to block the whole Internet, it has a *blacklist* and only filters sites with potentially inappropriate contents. Furthermore, the adversary does not want to slow down the Internet speed too much, which means it can only use simple and fast methods to detect inappropriate traffic. This is a realistic assumption because both whitelist-based filtering and too-complexed traffic inspection methods costs too much. The adversary is actively improving its censoring techniques, so detect-and-blocking anti-censorship traffic is possible. We further assume that the adversary cannot install malicious software such as backdoors on client's computer, this is necessary because if the client is under adversary's control, any encryption or obfuscation method lose all effectiveness.

One key difference between GoHop and Tor (and Tor's plugins), is that GoHop does not intend to provide public service. Instead, it's designed as a personal VPN tool. That is, the the client user and server user are ether in trust relationship, or even the same person. Thus, there can be pre-shared information, such as PGP key pairs, between the client and server before connection.

According to these assumptions, the client and server can communicate with each other, the adversary is willing to inspect packets in the traffic between them, but does not block the server, and is not willing to decrypt the traffic using brute force. And the server has the ability to relay the traffic between client and target site, and the traffic between client and server is encrypted and obfuscated, so the client can successfully get access to the target.

B. Design Goals

GoHop's design goals are:

- **Hard to detect**: Since the adversary is improving its censoring techniques, if GoHop traffic can be easily detected, the server will be blocked. In order to achieve this goal, both encryption and obfuscation are necessary.
- **Fast**: Internet traffic now contains much richer contents than that 10 years ago, people are willing to watch HD videos and use cloud computing to help everyday work, while existing censorship circumvention tools like Tor can only achieve the speed at 200 KB/s, which can only support basic web-surfing.
- **Flexible and easy to maintain**: GoHop need to update obfuscation algorithms frequently to follow the improving censorship techniques, thus demanding GoHop can be easily configured to use different obfuscation protocols and add new algorithms.

We developed several obfuscation techniques to achieve the first goal, and these obfuscation methods are relatively simple so that the throughput performance can satisfy new internet surfing needs.

III. Obfuscation Techniques in GoHop

Some of the aforementioned obfuscation protocols has been proved to be effective and applicable in practice, and it's not difficult to find the common methods of these protocols:

- Encrypting: Encryption can easily make data values in obfuscated packets look randomly distributed, and also protect data from sniffing.
- packet padding: To hide the flow's packet-size property, packet padding is a easy way. But it's been proved that only if the padded packet is near the size of MTU, padding does not help a lot to defeat statistical traffic analysis[17].

The main properties of an encrypted packets flow that an adversary can make use of includes:

- Plaintext, or fixed bytes
- Packet size
- Arrival time, and inter-arrival time
- Packet direction

In order to thwart statistical traffic analysis, we need to disguise traffic properties as much as possible, so three obfuscation methods were put in to practice: *pre-shared key encryption* to avoid plaintext, *traffic shaping* to change packet size, and *random port communication* to disguise inter-arrival time and hide packet direction.

A. Pre-shared Key Encryption

Since GoHop is a personal VPN tool, server and client are able to share a master key (or each other's public key) before connection establishment.

One reason to use pre-shared key is to avoid unencrypted key exchange. Although it's unlikely that the censoring authority would inspect every packet and trace the key exchange to decrypt payload data, the flow is indeed at risk.

Another and more important reason is that with the help of a pre-shared key, all data in the packet can be encrypted, including header, payload and even useless padding bytes. This enables true randomness of packet data distribution. In many protocols [18], only payload data is encrypted while packet headers are still in plaintext, this enables the censoring system to identify the protocol with only some packet header information.

B. Traffic Shaping

Traffic shaping is to change the packet size and timing properties of the flow, misleading the adversary to detect the traffic as another or unable to determine the true traffic class. Padding can be treated as the most naïve traffic shaping method, while previous work has proved that small size padding does not influence the accuracy of traffic analysis very much unless every packet is padded to MTU. [17] The main difference between packet padding and traffic shaping is that padding only increase the packet size, while traffic shaping also split large packets to some smaller packet pieces, and can transform the statistical properties of on traffic class to another, thus cheating the adversary.

Wright et al. [7] has shown that traffic morphing can be quite an efficient defence against traffic analysis with both strong privacy protection ability and low overhead. Monghaddam et al. [14] implemented this idea and successfully made Tor traffic morph into Skype video traffic in two ways: *naïve traffic shaping* and *Wright's traffic morphing*.

Monghaddam's experiment showed that the two techniques have almost equally effective, while naïve traffic shaping is less efficient but much easier to implement. What's more, Wright's traffic morphing is

a offline algorithm which needs to know source traffic properties and target traffic properties before figuring out the transforming matrix, while naïve traffic shaping only need to know the target traffic properties and can transform the traffic on the go.

Unlike Tor, whose payload is mainly HTTP traffic and has a relatively more static traffic property distribution, GoHop is a VPN service and many other protocols such as FTP, SSH or POP3 can run over, and its traffic property varies as the playload protocol.

Wright's static traffic morphing technique cannot satisfy our need of dynamic source traffic, so we take Monghaddam's naïve traffic shaping as our traffic shaping method.

C. Random Port

Although we're not able to know how a real-world censoring system exactly works, according to common sense and some existing traffic analysis projects [19], we can guess that the adversary's traffic identifying module has a concept of TCP connection or UDP session. That is, if two packets between two hosts has different ports, they are thought to belong to two different traffic flows.

In GoHop, we break this traditional common logic, that GoHop's packets are transported in many ports randomly.

There're three major benefits of random port datagrams: first, traditional 5-tuple based session detection was broken, so that flow properties on "direction" would be leaked minimally; second, the inter-arrival time of packet on each port would also distribute randomly; last but not least, if the random range is large enough, for example, 10,000 ports, then the throughput on each port can be quite low, increasing the difficulty to analyze the traffic's statistical properties.

IV. Implementation

We implemented GoHop in Go [20] on Linux. In this section, we first present the architecture and data flow of GoHop, and then introduce detailed implementation of key obfuscation techniques.

A. Architecture

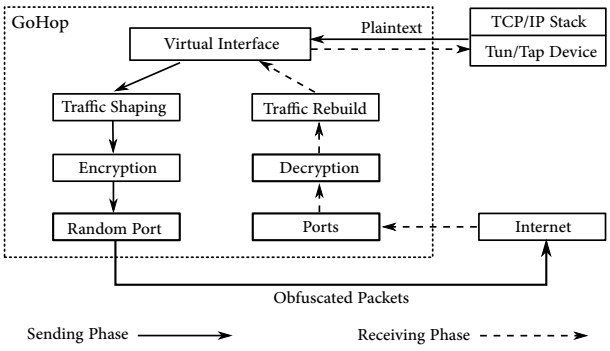


Fig. 1. GoHop Architecture and Data Flow

The basic architecture and data flow of GoHop is shown in figure 1. The server and client shares the same architecture, the only difference between them is that the server need to handle multiple clients while the client only communicates with one server.

High-level packet routing process is done by Linux's network tools, including `iptables` and `iproute2`.

1) Session Establishment: GoHop server need to start before the client. The first thing GoHop does after start is to initialize an AES-128-CBC cypher with the pre-shared key. The server then create and configure a virtual interface using Linux's `tun/tap` device API. After that, the server starts serving on several UDP ports. Now the server is ready to serve clients.

GoHop client first sends several "port knock" packets to every port that server listens, and chooses one of them to send the handshake datagram. The server would assign an IP address to the client via an ACK packet, then the client is able to initialize its virtual interface. Hence the session is successfully established.

As a proof-of-concept implementation, the session keeps little information, but nothing in principle prevents us to enhance the features. For example, there can be a dynamic, randomly generated session cypher, increasing the security of GoHop. We put this off as our future work.

2) Frame Processing and Forwarding: When GoHop reads a frame from virtual interface, which is sent from an upper-layer application, such as web browser, it first sends this frame to the traffic shaping oracle, and then this frame would be encapsulated in several transformed packets, and then chooses a random port to send these packets out to the other GoHop peer. Before the packet was being sent, the cipher would encrypt this packet.

The receiving phase is the reverse process of sending. When a packet comes from any port that GoHop listens, it would be decrypted to plaintext, and then sent to the frame-rebuild buffer. The traffic shaping oracle then rebuild these transformed packets to one frame, then send it up to the virtual interface. Then upper-layer application can be able to process that frame as normal.

The only difference between server and client in this phase is that before sending a packet out, the server need to determine which client it should send to, a.k.a, determine the right UDP address.

B. Packet Format

GoHop traffic is sent via UDP. There are 3 reasons to use UDP instead of TCP:

- UDP has less overhead: the UDP packet header is 8 bytes while TCP is 32 bytes.
- UDP is faster, especially for VPN traffic: As a VPN service, TCP will run at an upper layer, which means there is no need to guarantee reliability on VPN layer, what's more, if TCP flow control had run twice, the speed would be extremely slow.
- UDP is more hidden: TCP is a stateful, connection-oriented protocol, the 3-way handshake is very easy too be detected by the adversary.

The packet layout is shown in figure 2, where the size of each field is up on the field name in bytes. The `IV` field is the initial vector of AES cypher, other partes (shaded in figure) are encrypted using AES-128-CBC. `Flag` is an 8-bit field, it can both identify the packet type and set some options. `Seq` is a 32-bit unsigned sequence number. Because traffic shaping method may split a frame into several packets, there is a `Len` field denoting the size of frame, and `Prefix` denotes the prefix of the packet slice in the whole frame. The `Sid` denotes the session id. Since there are both payload data and noise padding in a packet, the `Dlen` field denotes the length of payload data.

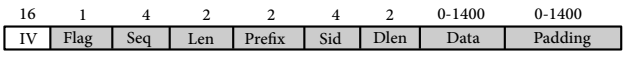


Fig. 2. GoHop UDP packet layout

C. Traffic Shaping

The traffic shaping phase is similar to SkypeMorph [14], since GoHop also utilizes Monghaddam's naïve traffic shaping method.

One key component of traffic shaping is the traffic shaping oracle, the only function of the oracle is to generate the next size of packet in the traffic. The algorithm of this function is described as follows: the oracle first read a packet-size Cumulative Distribution Function (CDF) of the target traffic, such as Skype video, denoted as $\boldsymbol{x}$, where x_i represents the probability that a packet is smaller than or equal to i; every time it needs to generate the next packet size, it first generate a random float number in $[0, 1]$ denoted as $\hat{x}$, then search from $\boldsymbol{x}$ for the x_j which is smaller than and nearest to $\hat{x}$, then return j as the next packet size.

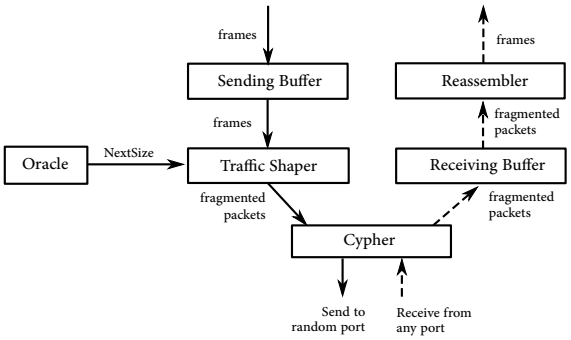


Fig. 3. Data Flow of GoHop Traffic Shaping

The data flow of the traffic shaping phase is shown in figure 3. The traffic shaper reads next packet size token from the oracle and a frame from sending buffer, and then pads the frame if it's small than token size, or fragmentates it if it's bigger than token size.

When receiving packets from the Internet, the reassembler would re-assemble the packet fragments into a whole frame and then send it to upper layer.

D. Random Port

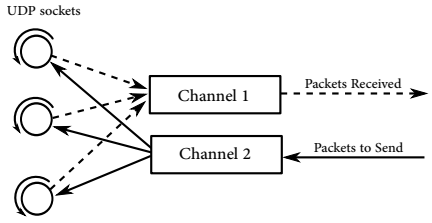


Fig. 4. Random Port Implementation

Random port can be easily implemented with the help of Go's concurrent feature. As is shown in figure 4, For each port, GoHop would spawn a *goroutine* (similar to thread) to listen on, and as soon as a packet arrives, it would be put into a global *channel*. Upper layer processes packets from this channel, just like packets are from one socket.

For sending phase, upper layer puts packets to another global channel, then the scheduling algorithm would determine which goroutine to take that packet, then that goroutine would send the packet to the Internet.

V. Experiments

Corresponding to our design goals, we perform our experiment in two phases: obfuscation effectiveness test and throughput performance test.

A. Obfuscation Effectiveness

In the effectiveness test, we deploy GoHop server on a VPS with 128MB memory located in Seattle, US, the client runs on a PC located in Beijing, china. We set MTU of virtual interface to 1400 since there is a 82-byte overhead of VPN-tunnel and encryption. The random port range is 40100 to 40200, and the target traffic shape is packet size uniformly distributed from 0 to MTU. As soon as GoHop starts, the client PC can successfully access websites blocked in China.

In order to test the effectiveness of our obfuscation method, we collect two types of traffic, HTTP and SSH, from the client. When collecting data, we collect one piece from the virtual interface, which is the traffic from and to user applications, and another piece at the physical interface, which is the traffic that can be inspected by the adversary.

We use Wright's visualized classifying method [21] to evaluate our effectiveness as shown in figure 5.

Figure 5(a)-5(e) shows the effect of traffic shaping. Figure 5(a) and figure 5(d) are heatmaps for HTTP and SSH, where x axis is time and y axis is packet size and client-to-server packet size is positive and server-to-client packet size is negative. From the figure we can see that for HTTP traffic, the server-to-client packets are often as large as MTU, while client-to-server packets are quite small. SSH traffic is very different from HTTP, both directions traffic's packet size is small, which carry human-interactive information. With this traffic visualization method, observers can easily identify the traffic's application protocol without inspecting packet content. [21]

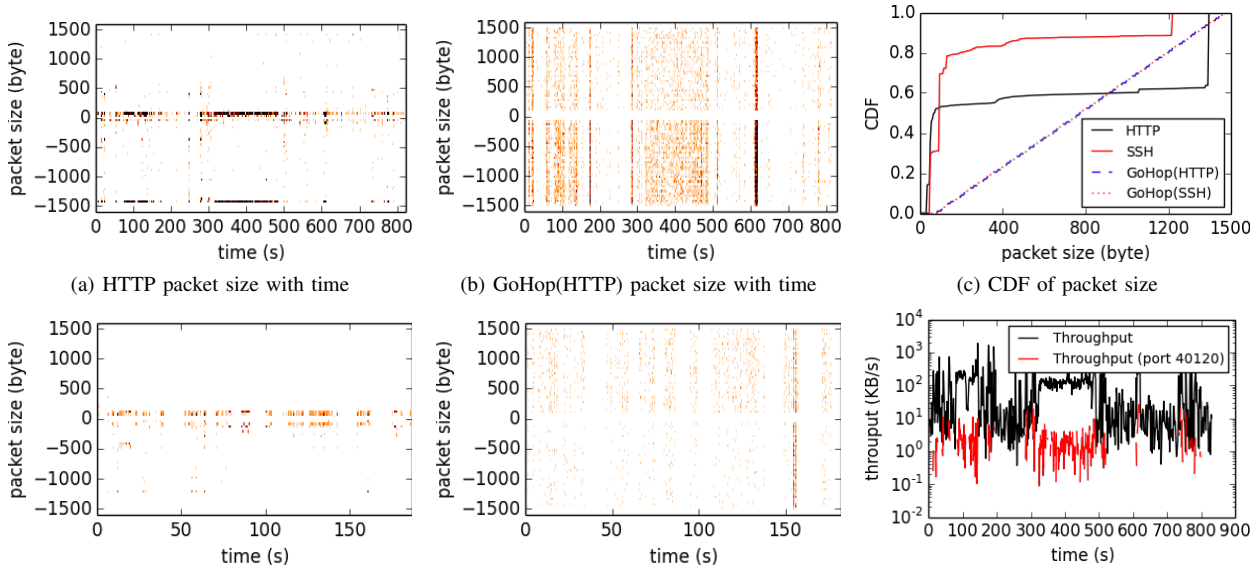
Figure 5(b) and figure 5(e) are heatmaps for GoHop packets that carry HTTP and SSH. We can see that GoHop successfully changed the in-tunnel traffic distribution, and expect for throughput, there is no much difference between HTTP and SSH in packet size distribution from human's view. Since we did not change the timing property of traffic, we can still tell that HTTP needs more packets than SSH.

TABLE I
Statistical Properties of Packet Size

Protocol	Max	Min	Mean	Variance	D (K-S)	p (K-S)
HTTP	1400	40	610	409867	-	-
SSH	1226	52	247	138053	-	-
GH(HTTP)	1482	73	783	163598	0.019	0.20
GH(SSH)	1482	90	782	163889	0.022	0.11

Figure 5(c) and table I present more detailed properties on packet size. Figure 5(c) is the CDF of packet size for HTTP, SSH and GoHop. We can see that CDF of HTTP differs much with SSH, but GoHop traffic CDF for SSH and HTTP is almost the same. Table I shows several statistical metrics for packet size. These metrics for original HTTP and SSH varies a lot, but for GoHop obfuscated traffic, the metrics are almost the same. We further conduct Kolmogorv-Smirnov test [22] for first 3000 packets of GoHop traffic, the D value for both HTTP and SSH traffic is smaller than the threshold (0.025) and p value is larger than 0.05, so we cannot reject the hypothesis that the traffic shaper successfully morphed packet size distribution of both HTTP and SSH traffic to uniform distribution.

Figure 5(f) shows the throughput comparison between whole GoHop traffic and that on one specified port. Since GoHop use



(a) HTTP packet size with time (b) GoHop(HTTP) packet size with time (c) CDF of packet size

(d) SSH packet size with time (e) GoHop(SSH) packet size with time (f) Port throughput

Fig. 5. Flow Properties GoHop

random port for UDP communication, the throughput on one single port is significantly lower than the whole traffic. So it's more difficult to be detected by the adversary.

B. Throughput Performance

1) Upper bound Performance: The first part of performance test is upper bound test, which determines the best performance that GoHop can reach. We deploy GoHop server on a desktop PC with a 4-core 3.0 GHz CPU and 8 GB memory, and GoHop client runs on a laptop with 2-core 2.6 GHz CPU and 4 GB memory. These 2 computers are connected via 1000 Mbps ethernet.

We utilize `iperf` tool to test throughput performance. We first test the direct-link speed, and that results in 936 Mbps, which is near to the theoretical speed of 1000 Mbps ethernet.

Then we test GoHop with 3 configurations:

1) traffic shaping off, communicate on one port
2) traffic shaping off, communicate on 100 ports
3) traffic shaping on, communicate on 100 ports

TABLE II
UPPER BOUND THROUGHPUT PERFORMANCE

Conf.	Virtual NIC	Physical NIC
1	76.8 Mbps	85.3 Mbps
2	78.5 Mbps	88.7 Mbps
3	58.1 Mbps	93.3 Mbps

The result is shown in table II, where virtual NIC speed is the throughput on top of GoHop, which is the effective traffic for applications, and physical NIC speed is the throughput on physical link. From the results we can see, that the number of random ports does not influence the performance of GoHop, while traffic shaping produced much traffic overhead and only about 60% of traffic is valid.

2) Application Performance: Then we test the performance in application scenario by downloading a 20 MB file[2] from an unblocked website. The GoHop server locates in Seattle, US and the client is located in Beijing, China.

[2]http://mirrors.kernel.org/debian/dists/stable/Contents-armel.gz

TABLE III
APPLICATION PERFORMANCE

Conf.	Speed	Packet Loss
direct	1544 KB/s	341/14803
1	960.8 KB/s	226/13259
2	999.1 KB/s	212/13146
3	697.3 KB/s	357/15573

Then we measure the speed of downloading the file, and the results are shown in table III, where the configurations for GoHop is same as that in upper bound test, and "direct" means direct download without connecting GoHop. From the results we can see: 1) the number of ports does not reduce throughput speed; 2) traffic shaping brought in overhead and increases the packet loss rate, since it generates more fragments in traffic.

We further use `iperf` to test the link speed between Beijing and Seattle, which results in 96.7 Mbps. Thus we can conclude that the bottleneck is the link speed between Beijing and Seattle instead of GoHop.

As a comparison, Tor can provide speed ranging in 40-300 KB/s and when introducing obfuscation plugins, the speed can only reach 30-80 KB/s [13, 14]. The overhead of traffic shaping is about 30%, which is similar to SkypeMorph [14]. So GoHop can provide much faster internet access than Tor.

VI. DISCUSSION AND FUTURE WORK

a) Performance: The performance test showed that GoHop can be faster than Tor, one reason why Tor is slow is that Tor provides high-level security and anonymity while GoHop's encryption method is relatively simpler. Whats more, Tor provides public service while GoHop provides personal service, sharing bandwidth resource also reduces Tor's speed. Skypemorph [14] reduces Tor speed a lot since it has to send the same amount of data as Skype does.

But GoHop's traffic shaping also brings in more than 30% traffic overhead. There are two directions of work to reduce the overhead. First, we can find a "better shape" as the target shape which does not produce so much overhead. Another way is to find an optimal mapping from source traffic to the target traffic, Wright's has found an optimal traffic morphing [7] method, but it's only valid on traffic with static distribution, while in practice we need an optimal traffic morphing method with dynamic property distribution. We put these off as our future work.

b) Security: GoHop is still in development, and now it's implementation is not secure enough. We did not implement dynamic session key, So every packet for GoHop shares one master key. If that key was leaked to an attacker, then he can decrypt all GoHop traffic.

To implement dynamic key, the session establishment part need to be enhanced. We can adopt D-H key exchange [23] to create a session cipher besides the master cipher, and encrypt packet payload with session cipher and header with master cipher. Thus GoHop's security is improved.

c) Ports: The random port communication is effective only if the adversary has a concept of TCP connection or UDP session. The adversary can still detect the range of ports that GoHop uses, and random port loses its effectiveness of the adversary merges sessions in that range. We can enhance this protocol that the range of random port is also random and synchronized between client and server, and there is also noise packet transmitting out side the range misleading the adversary. Thus the real port that GoHop uses gets more hidden.

VII. Conclusions

In this paper, we introduced a novel censorship circumvention tool named GoHop. GoHop is a VPN software which obfuscates its traffic to thwart censoring authorities' DPI and traffic analysis. Three obfuscation method were used in GoHop: pre-shared key encryption, traffic shaping and random port. The experiment results show that GoHop successfully changes traffic's packet size distribution, and the throughput on a specified port is significantly lower than whole traffic. GoHop is much faster than Tor, so it can provide better user experience for internet surfing. GoHop is published as an open-source software, and is available to be downloaded freely.

References

[1] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*, 2004, pp. 303–320.

[2] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations," in *IEEE Symposium on Security and Privacy*, 2008, pp. 35–49.

[3] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks," in *IEEE Symposium on Security and Privacy*, 2011, pp. 3–18.

[4] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting," *Computer Networks*, vol. 53, no. 1, pp. 81–97, Jan. 2009.

[5] C. Wright, F. Monrose, and G. Masson, "On inferring application protocol behaviors in encrypted network traffic," *The Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006.

[6] S. J. Murdoch and G. Danezis, "Low-Cost Traffic Analysis of Tor," in *IEEE Symposium on Security and Privacy*, 2005, pp. 183–195.

[7] C. Wright, S. Coull, and F. Monrose, "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis." *NDSS*, pp. 237–250, 2009.

[8] Q. Sun, D. R. Simon, Y. min Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical Identification of Encrypted Web Browsing Traffic," in *IEEE Symposium on Security and Privacy*, 2002, pp. 19–30.

[9] P. Winter and S. Lindskog, "How the Great Firewall of China is Blocking Tor," in *Free and Open Communications on the Internet*. Bellevue, WA, USA: USENIX Association, 2012.

[10] M. Maxstead, "Vpnreviewz advises mainland china users: Most vpns are blocked but not all," 2013. [Online]. Available: http://www.prweb.com/releases/2013/4/prweb10607609.htm

[11] Tor Project, "Pluggable transports." [Online]. Available: https://www.torproject.org/docs/pluggable-transports.html.en

[12] ——, "Obfsproxy." [Online]. Available: https://www.torproject.org/projects/obfsproxy.html.en

[13] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. Porras, "Evading censorship with browser-based proxies," in *Proceedings of the 12th international conference on Privacy Enhancing Technologies*, ser. PETS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 239–258.

[14] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph," in *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. New York, New York, USA: ACM Press, 2012, p. 97.

[15] Clowwindy, Madeye, and L. Max, "Shadowsocks." [Online]. Available: http://www.shadowsocks.org/

[16] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis, "Towards efficient traffic-analysis resistant anonymity networks," *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, p. 303, 2013.

[17] E. Hjelmvik and W. John, "Breaking and improving protocol obfuscation," *Chalmers University of Technology, Tech. Rep*, vol. 123751, 2010.

[18] OpenVPN, "Security overview." [Online]. Available: http://openvpn.net/index.php/open-source/documentation/security-overview.html

[19] E. Hjelmvik and W. John, "Statistical protocol identification with spid: Preliminary results," *Swedish National Computer Networking Workshop*, 2009.

[20] Google, "The go programming language," 2006. [Online]. Available: http://golang.org

[21] C. V. Wright, F. Monrose, and G. M. Masson, "Using visual motifs to classify encrypted traffic," in *Proceedings of the 3rd international workshop on Visualization for computer security - VizSEC '06*. New York, New York, USA: ACM Press, 2006, p. 41.

[22] NIST, "Kolmogorov-smirnov goodness-of-fit test." [Online]. Available: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm

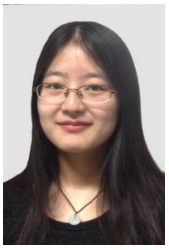
[23] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

Yuzhi Wang received his B.S. degree in 2012 from School of Telecommunication Engineering in Xidian University, Xi'an, China, and he is currently a Ph.D. student in Department of Electronic Engineering in Tsinghua University, Beijing, China, under the supervision of Prof. Huazhong Yang. Mr. Wang's research mainly focuses on Wireless Sensor Networks, Internet of Things and Network Security.

Pengjun Wang received the B.S. and Ph.D. degrees from the NICS Group, Department of Electronic Engineering, Tsinghua University, Beijing, China in 2006 and 2011, respectively. Currently, he is an Assistant Research Scientist in Department of Electronic Engineering in Tsinghua University. His recent research mainly focuses on Wireless Sensor Networks and Structural Health Monitoring.

Ping Ji received the B.A. degree in computer science and technology from Tsinghua University, Beijing, China, in 1998, and the Ph.D. degree in computer science from the University of Massachusetts, Amherst, in 2003. Since 2003, she has been a faculty member of the Master's Program in Forensics Computing at the Mathematics and Computer Science Department, John Jay College of Criminal Justice, New York, NY, and a faculty member of the Ph.D. program in Computer Science of the Graduate Center, City University of New York (CUNY), Flushing, NY. Her research interests include protocol design, performance analysis, and signaling for advanced networks services, network security, wireless and sensor networks, and network measurements and performance modeling.

Rong Luo (M'05) received the double B.S. degree in engineering physics and electronic engineering and the Ph.D. degree from Tsinghua University, Beijing, China, in 1992 and 1997, respectively. Currently, she is an Associate Professor with the Department of Electronic Engineering, Tsinghua University. Her current research work is mainly on SoC design technology, VLSI design, Wireless Sensor Networks and Data Mining.



Borui Ye is currently an undergraduate student in School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China. Ms. Ye's research interests include Artificial Intelligence, Machine Learning and Network Security.

Huazhong Yang (M'97-SM'00) received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in circuits and systems from Tsinghua University, Beijing, China, in 1989, 1993 and 1998, respectively. Since 1993, he has been with the Department of Electronic Engineering, Tsinghua University, where he has been a Full Professor since 1998. His research interests include CMOS radio-frequency integrated circuits, VLSI system structure for digital communications and media processing, low-voltage and low- power ICs, and computer-aided design methodologies for system integration, wireless sensor networks and structural health monitoring.