SkypeMorph: Protocol Obfuscation for Tor Bridges

Hooman Mohajeri Moghaddam Baiyu Li
Mohammad Derakhshani Ian Goldberg

Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{hmohajer,b5li,mderakhs,iang}@cs.uwaterloo.ca

ABSTRACT

The Tor network is designed to provide users with low-latency anonymous communications. Tor clients build circuits with publicly listed relays to anonymously reach their destinations. However, since the relays are publicly listed, they can be easily blocked by censoring adversaries. Consequently, the Tor project envisioned the possibility of unlisted entry points to the Tor network, commonly known as *bridges*. We address the issue of preventing censors from detecting the bridges by observing the communications between them and nodes in their network.

We propose a model in which the client obfuscates its messages to the bridge in a widely used protocol over the Internet. We investigate using *Skype video* calls as our target protocol and our goal is to make it difficult for the censoring adversary to distinguish between the obfuscated bridge connections and actual Skype calls using statistical comparisons.

We have implemented our model as a proof-of-concept pluggable transport for Tor, which is available under an open-source licence. Using this implementation we observed the obfuscated bridge communications and compared it with those of Skype calls and presented the results.

Keywords

Tor, bridges, Skype, pluggable transports, steganography, protocol obfuscation, censorship circumvention

1. INTRODUCTION

Tor [19] is a low-latency anonymous communication overlay network. In order to use Tor, clients contact publicly known *directory servers*, a fraction of the Tor network responsible for tracking the topology of the network and node states. Directory servers allow clients to obtain a list of volunteer-operated relay nodes, also known as *onion routers*. The client then chooses some of these relays using the Tor software and establishes a circuit through these nodes to its desired destination. Clients' traffic is then routed through the Tor network over their circuits, hiding users' identities and activities.

The Tor network not only provides anonymity, but also *censorship resistance*. To access a website censored in a user's home country, the user simply connects to the Tor network and requests the blocked content to be delivered to him. However, since a list of Tor relays can be retrieved from publicly known directory servers, blocking all Tor connections can be simply done by blocking access to all Tor relays based on their IP addresses. There have been many attempts to block access to Tor by regional and state-level ISPs. For instance, Figure 1 shows the blocking of the whole Tor network by the Great Firewall of China as of 2010.

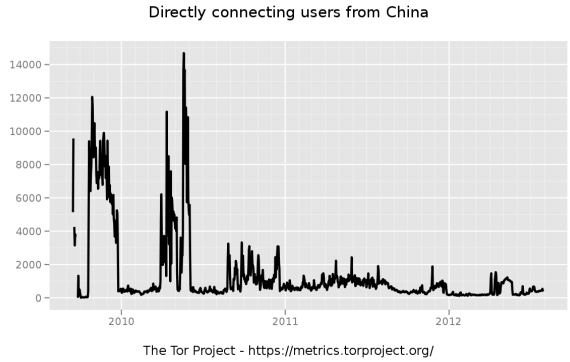


Figure 1: This graph from metrics.torproject.org shows the number of users directly connecting to the Tor network from China, from mid-2009 to the present. It shows that, after 2010, the Tor network has been almost completely blocked from clients in China who do not use bridges. [41]

In order to counteract this problem, the Tor project suggested using *bridges* — unlisted relays used as entry points to the Tor network. Since bridges are not publicly listed, the censoring authority cannot easily discover their IP addresses. Although bridges are more resilient to censorship, McLachlan and Hopper [32] showed that it is still possible to identify them, as they accept incoming connections unconditionally. To solve this problem, BridgeSPA [39] places some restrictions on how bridges should accept incoming connections.

As the censorship techniques improve, however, more sophisticated methods are being deployed to discover and block bridges. There have been reports of probes performed by hosts located in China, aimed quite directly at locating Tor bridges [43, 44]. The investigation revealed that after a Tor client within China connected to a US-based bridge, the same bridge received a series of Tor connection initiation messages from different hosts within China and after a while the client's connection to the bridge was lost. We have recently witnessed state-level SSL blocking [36] and blocking of Tor connections based on the expiry time of the SSL certificate generated by the Tor software [18].

As the censorship arms race is shifting toward the characteristics of connections, Appelbaum and Mathewson pro-

posed a framework for developing protocol-level obfuscation plugins for Tor called *pluggable transports* [9]. These transports appear to the Tor client to be SOCKS proxies; any data that the Tor client would ordinarily send to a bridge is sent to the pluggable transport SOCKS proxy instead, which delivers it to the bridge in an obfuscated way. Developers can use this framework to build their own transports, hiding Tor traffic in other protocols. On one side, the transport obfuscates Tor messages in a different form of traffic, e.g., HTTP, and on the other side it translates the HTTP traffic back into Tor traffic. Pluggable transports provide an easy way to resist client-to-bridge censorship and the ultimate goal is that censoring ISPs that are inspecting packets based on their characteristics will be unable to discover the Tor traffic obfuscated by a transport.

At the time of writing, the only available pluggable transport is "obfsproxy" [30], which passes all traffic through a stream cipher. We extend this previous work to address its limitation of *not* outputting innocuous-looking traffic; our method greatly reduces the chances of obfuscated bridge connections being detected by powerful censors. We also note that simply adding the target protocol headers to packets would not be successful when facing deep packet inspection (DPI) methods [28]. Dusi et al. [20] also suggested that obfuscating inside an encrypted tunnel might not be enough to withstand statistical classifiers since some features of encrypted tunnels such as packet sizes and inter-arrival times of packets can still be distinguished.

For the purpose of our experiment we chose the Skype [4] protocol as our target communication for several reasons. First, Skype enables users to make free, unlimited and encrypted voice and video calls over the Internet which has led to its huge popularity [3] and therefore the amount of Skype traffic in today's Internet is relatively high. Second, Skype video calls transfer a reasonable amount of data in a short period of time, making it a desirable form of target traffic since it will not introduce too much of a bottleneck to the Tor connection. Third, Skype communications are all encrypted [2], so it provides an encrypted channel for the Tor traffic.

1.1 Our Contributions

We explore methods for Tor protocol obfuscation and introduce SkypeMorph, a system designed to encapsulate Tor traffic into a connection that resembles Skype video traffic. We provide the following contributions:

- **Tor traffic obfuscation**: SkypeMorph disguises communication between the bridge and the client as a Skype video call, which is our target protocol. Protocol obfuscation is greatly needed when facing large-scale censorship mechanisms, such as deep packet inspection.

- **Innocuous-looking traffic**: A client who wishes to access a SkypeMorph bridge runs our software alongside his usual Tor client and instructs his Tor client to use the SkypeMorph software as a transport. Upon startup, SkypeMorph first attempts a Skype login process and then establishes a Skype call to the intended destination; i.e., the bridge. Once the bridge receives the call, the client innocuously drops the call and uses the channel to send the obfuscated Tor messages. We give comparisons between the output of SkypeMorph and actual video calls of Skype and we conclude that for the censoring adversary it would be difficult to differentiate between the two. Consequently, a censor would be required to block a great portion of legitimate connections in order to prevent access to the obfuscated Tor messages.

- **UDP-based implementation**: Since Skype mainly uses UDP as the transport protocol, we also use UDP. The choice of UDP as the transport protocol will also be useful when Tor datagram designs [33] are rolled out.

- **Improved traffic shaping**: *Traffic Morphing* as proposed by Wright et al. [46] is based on the premise of efficiently morphing one class of traffic into another. However, the authors neglected one key element of encrypted channels, namely the inter-packet delay between consecutive packets, from their design scope. SkypeMorph extends the previous work to fully reproduce the characteristics of Skype calls.

- **Comparison between traffic shaping methods**: We compare different modes of implementing traffic shaping and describe how each of them performs in terms of network overhead. In particular, we explore two methods, namely naïve traffic shaping and our enhanced version of Traffic Morphing, and compare them.

- **Proof-of-concept implementation**: We have made our open-source proof-of-concept SkypeMorph implementation available online at:

  `http://crysp.uwaterloo.ca/software/`

The outline of the remainder of the paper is as follows. In Section 2 we discuss related work and in Section 3 we formalize our threat model and design goals. Section 4 covers some background and we present our architecture and implementation in Sections 5 and 6. We present our results in Section 7, discuss possible future work in Section 8, and conclude in Section 9.

2. RELATED WORK

2.1 Information Hiding and Steganography

Hiding information within subliminal channels has been studied extensively in the last three decades. Simmons [38] stated the problem for the first time and proposed a solution based on digital signatures. Currently the topic is studied under the term *steganography* or the art of concealed writing, and it has recently attracted a lot of attention in digital communications [35].

Employing a steganographic technique, one needs to consider two major factors: the *security* and *efficiency* of the method. Hopper et al. [22] proposed a construction that can formally be proven to be secure and can be applied to a broad range of channels. The `OneBlock` stegosystem described in their work, however, needs an expected number of samples from the channel that is exponential in the number of bits transmitted.

2.2 Image Steganography

Hiding information within pictures is a classic form of steganography. Least Significant Bit (LSB) based image steganographic techniques [24] — using a small fraction of the information in each pixel in a cover image to send the actual data — is a common method and Chandramouli et al. [16] showed the upper bounds for the capacity of such channels. There are also some existing open-source steganographic tools available, such as `outguess`[1] and `steghide`[2].

`Collage` [15] is a recently developed system that uses user-generated content on social-networking and image-sharing websites such as Facebook and Flickr to embed hidden messages into cover traffic, making it difficult for a censor to block the contents. `Collage` has a layered architecture: a "message vector layer" for hiding content in cover traffic (using `outguess` internally as their steganographic tool) and a "rendezvous mechanism" for signalling. The authors claim the overhead imposed by `Collage` is reasonable for sending small messages, such as Web browsing and sending email.

2.3 Voice over IP and Video Streaming

Wright et al. [47] studied the effectiveness of security mechanisms currently applied to VoIP systems. They were able to identify the spoken language in encrypted VoIP calls encoded using bandwidth-saving Variable Bit Rate (VBR) coders. They did so by building a classifier which could achieve a high accuracy in detecting the spoken language when a length-preserving encryption scheme was used and they concluded that the lengths of messages leak a lot of information. Further experiments showed that it is possible to uncover frequently used phrases [45] or unmask parts of the conversation [42], when the same encryption method is employed for confidentiality.

However, the statistical properties these attacks exploit are less prevalent in streaming video data, rather than the audio data they consider; therefore, those algorithms should not be able to distinguish our Tor traffic disguised as Skype video traffic from real Skype video traffic. Nonetheless, we consider the question of matching SkypeMorph traffic to the higher-order statistics of Skype video traffic to fully resemble Skype communication to a censor.

Previous work has shown some success in determining whether a target video is being watched, using information leakage of electromagnetic interference (EMI) signatures in electronic devices [21], or revealing which videos in a database are being viewed in a household by throughput analysis [37]. However, those methods require the purported video to be selected from a set known in advance. SkypeMorph, on the other hand, attempts to disguise its traffic as a real-time video chat, which would not be in such as set.

VoIP services have also been used for message hiding. For example, Traffic Morphing [46] exploits the packet size distribution of VoIP conversations to transmit hidden messages (we will return to this method in section 4). Another example of steganographic communications over voice channels is `TranSteg` [31], in which the authors try to re-encode the voice stream in a call with a different codec, resulting in smaller payload size. Therefore, the remaining free space can be used for sending the hidden messages. The shortcoming of this method is that the most of the bandwidth is allocated to the actual voice conversation, leaving only a limited space for steganograms.

[1] `http://www.outguess.org`

[2] `http://steghide.sourceforge.net`

2.4 Steganography over Encrypted Channels

Although steganographic models similar to those mentioned above are powerful, they impose relatively large overheads on our channel. Therefore, we used a combination of methods suggested for encrypted communications [20, 46]. We argue that on an encrypted communication such as those of Skype calls, every message appears to be random (since we expect the encryption scheme to output a randomly distributed bit string), thus exploiting the channel history is not required for cover traffic and we can perform significantly better than the `OneBlock` stegosystem, `Collage`, or `TranSteg`. The only important characteristics of encrypted channels, as suggested by previous works, are packet sizes and inter-arrival times of consecutive packets [12, 29, 20]. Hence, a protocol obfuscation layer only needs to reproduce these features for an encrypted channel.

3. THREAT MODEL AND DESIGN GOALS

In this section we discuss our threat model and assumptions. In our model, we assume that the user is trying to access the Internet through Tor, while his activities are being monitored by a state-level ISP or authority, namely "the censor", who can capture, block or alter the user's communications based on pre-defined rules and heuristics. Therefore, we consider adversarial models similar to anti-censorship solutions such as Telex [48], Cirripede [23] and Decoy Routing [25]. In particular, the censor is able to block access to Tor's publicly listed routers, and to detect certain patterns in Tor's traffic and hence block them [18]. This is also true for other services or protocols for which the censoring authority is able to obtain the specification. Examples include protocols in the public domain, e.g., HTTP, and services whose provider can be forced or willing to reveal their implementation details.

However, we assume that the censoring authority is not willing to block the Internet entirely, nor a large fraction of the Internet traffic. The censoring authority is also unwilling to completely block popular services, such as VoIP protocols. Thus, the filtering is based on a "black list" of restricted domains and IP addresses, accompanied by a list of behavioural heuristics that may suggest a user's attempt to circumvent censorship; for example, a TCP SYN packet following a UDP packet to the same host may indicate a special type of proxy using *port knocking* [26]. Bissias et al. showed how such heuristics can be employed to detect certain traffic patterns in an encrypted channel [12].

The assumption that censorship is done based on "black lists" is a realistic one since usually the cost of over-blocking is not negligible. If the censor used a small "white list" of allowed content and hosts, then every new website or host on the Internet would need to sign up with the censor in order to be accessible by nodes within its control. This is a quite cumbersome task and seems unreasonable.

Also, we assume that encrypted communications, including Skype calls, are *not* blocked unless the censor has evidence that the user is trying to evade the censorship. Although there have been instances where Skype or other VoIP services were either banned or have gone inaccessible in some countries [8], to the best of our knowledge these instances are very rare and in most cases either the Skype website is

filtered or users are threatened with legal actions [7]. For instance, China has a different approach toward Skype and has partnered with it to be able to filter unwanted messages through a modified version of Skype, called TOM-Skype[3]; we strongly discourage SkypeMorph users from using this version with our software for anonymity purposes, however. Although some regimes may choose to block Skype, or more subtly, bandwidth-limit Skype so heavily that Skype video is unusable, but Skype audio persists, the diversity of methods of censoring Internet content suggests that our approach will remain pertinent.

We further assume that the censor does not have access to information about particular bridges, including their IP addresses and Skype IDs; otherwise it can readily block the bridge based on this information. (We will discuss in Section 8 how a bridge using SkypeMorph can easily change its IP address if it is detected by the censor.) SkypeMorph users, however, can obtain this information from out-of-band channels, including email, word-of-mouth, or social networking websites. We also note that it is possible to have multiple Skype calls use a single IP address but with different ports, to allow users behind NAT to make simultaneous calls using a shared IP address.

In our model, we are trying to facilitate connections to bridges outside the jurisdiction of the censor where it has no control over the network nodes. However, the censor can set up its own SkypeMorph bridges and distribute their information.

In general, SkypeMorph aims to build a layer of protocol obfuscation for Tor bridges,[4] with the following goals:

- **Hard to identify**: SkypeMorph outputs encrypted traffic that resembles Skype video calls. The details of how we try to minimize the chances of being detected by the censor are discussed in Section 4.

- **Hard to block**: Since the outputs of SkypeMorph greatly resemble Skype video calls, in order to block SkypeMorph, the censor would need to block Skype calls altogether, which we assume it is unwilling to do.

- **Plausible deniability**: The only way to prove that a node is actually using SkypeMorph software is to break into a user's machine or to coerce him to divulge his information. Otherwise, communicating through SkypeMorph should look like a normal Skype video chat.

Finally, we note that our work aims at defeating firewall and DPI tools which look for Tor flows. However, there are other approaches for enumerating bridges that are outside the scope of this paper. [17]

4. BACKGROUND

4.1 Skype

Skype [4] is a proprietary "voice over IP" (VoIP) service that provides voice and video communications, file transfer, and chat services. With millions of users and billions of minutes of voice and video conversations, Skype is undoubtedly one of the most popular VoIP services available [3].

Protection mechanisms and code obfuscation techniques used in the Skype software have made it difficult to learn about its internals, so there is no open-source variant of the Skype application. However, there have been attempts to reverse engineer and analyze the application [10, 11]. The findings from these attempts and our own experiments, alongside some insights from the Skype developers have established the following facts:

- Skype encrypts messages using the AES cipher and uses RSA-based certificates for authentication [2, 11]. Also our experiments showed that Skype utilizes some form of message authentication and would not accept altered messages. Thus an eavesdropper is neither able to access the content of a packet nor can he alter them in such a way that is not detectable. All that is possible to such an attacker is selective packet dropping or denial of service.

- There are three types of nodes in the Skype network: server nodes, which handle users' authentication when they sign in, normal nodes, which can be seen as peers in the P2P network, and supernodes, which are those peers with higher bandwidth; supernodes can facilitate indirect communication of peers behind firewalls or NAT [11, 13, 40].

- Skype calls are operated in a peer-to-peer architecture and users connect directly to each other during a call, unless some of the participants cannot establish direct connections. This is where supernodes come into the picture to facilitate the call.

- In our experiments with Skype we noticed that when a Skype call takes place there are some TCP connections which are mainly used for signalling. These TCP connections remained open even after the call is dropped. The Skype client listens to a customizable UDP port for incoming data, but when UDP communication is not possible, it falls back to TCP [10, 11].

- Skype has a variety of voice and video codecs and selects among them according to bandwidth, network speed and several other factors [13, 14].

The facts that Skype traffic is encrypted and very popular makes it a good candidate for the underlying target traffic for our purpose. We will explore this more in Section 5.

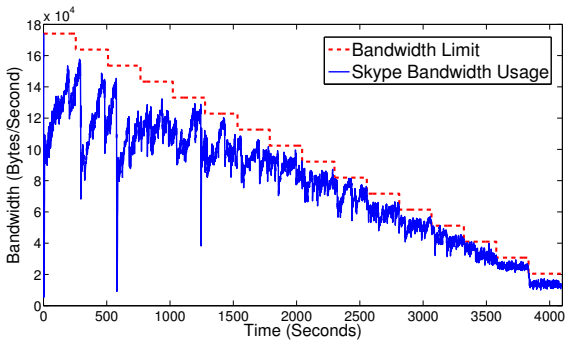
The choice of Skype video, as opposed to voice, calls as the target protocol in SkypeMorph is motivated by the fact that in voice calls, usually at any time only one party is speaking and thus we would need to consider this "half-duplex" effect in our output stream. However, this is not the case in video calls since both parties send almost the same amount of data at any given time during a video conversation, making the implementation of SkypeMorph easier, and not requiring the client or bridge to withhold data until it is its turn to "speak".

4.1.1 Bandwidth Control

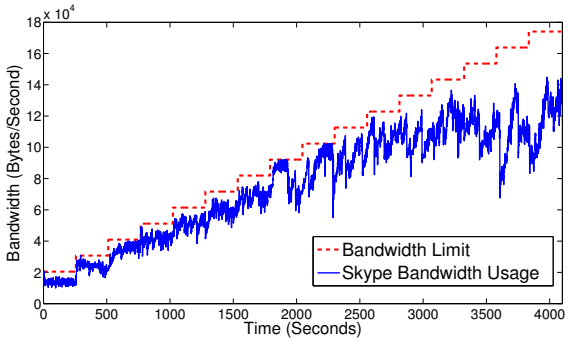
Network congestion control and bandwidth throttling is a major concern in online applications. In order to be able

[3] `http://skype.tom.com`

[4] Note that our technique can be applied to Tor's public ORs, but since blocking public ORs based on their IP address is a trivial task, we choose to present it for bridges.



(a) Decreasing bandwidth available to Skype



(b) Increasing bandwidth available to Skype

Figure 2: Bandwidth usage by Skype under different network situations. Figure 2a shows the drops in the Skype transfer rate while decreasing the network bandwidth and Figure 2b shows the increase in the rate.

to accommodate for changes in the network status, a traffic control mechanism is essential and Skype uses a congestion detection mechanism to back off whenever it is no longer possible to communicate at the current rate. Skype voice calls were shown to have a few number of possible bitrates [14], however, as shown in Figure 2, Skype video calls seem to enjoy much more flexibility in terms of bandwidth usage.[5] Figure 2a suggests that by limiting the available bandwidth, Skype's bandwidth usage drops significantly at each step to a level far below the available rate (this phenomena is more noticeable in higher rates) and then it builds up again to achieve the maximum rate possible. Also Skype is able to detect whether it is possible to send at a higher rate, as depicted in Figure 2b. Therefore, by a similar rate limiting technique, SkypeMorph is able to transfer data at a reasonable rate which at the same time complies with the bandwidth specified by the bridge operator.

4.2 Naïve Traffic Shaping

To achieve a similar statistical distribution of packet sizes in the output of our system to that of target process, a basic approach would be to simply draw samples from the packet size distribution of the target process and send the resulting size on the wire. Thus, if there is not enough data available from Tor, we have to send packets without useful information and this imposes some additional overhead on the network. An alternate approach is to consider the incoming packet sizes from the source distribution, which is how Traffic Morphing deals with the problem.

[5]The degree of flexibility in bandwidth usage of Skype video calls is due to availability of different frame rates and video codecs.

4.3 Traffic Morphing

We briefly mention how the original Traffic Morphing [46] method works. Traffic Morphing attempts to counter an adversary who is trying to distinguish traffic from the source process from that of the target process through statistical means. As previously discussed, the only statistical traces that the attacker might be able to collect from encrypted traffic are packet sizes and timing attributes. Traffic Morphing aims at obfuscating the packet size distribution by assuming that probability distributions of the source and destination processes are available.

Assume that the probability distribution of the source process is denoted by the vector $X = [x_1, \ldots, x_n]^T$, where x_i is the probability of the i^{th} packet size. Similarly $Y = [y_1, \ldots, y_n]^T$ denotes the target process packet size distribution. Traffic Morphing finds matrix A for which we have $Y = AX$ such that the number of additional bytes needed to be transmitted is minimal. Using this technique requires some considerations, for example dealing with larger sample spaces or overspecified constraints, which are discussed in the original paper.

Even though the underlying premise of Traffic Morphing is that if the source process generates a sufficiently large number of packets, the output of the morphing will converge in distribution to that of the target, it only considers *packet sizes* in the encrypted traffic. We extend this technique, below, by introducing *inter-packet timing* to it as well.

4.4 Higher-Order Statistics

Although reproducing Skype packet size and inter-packet delay distributions is a step towards defeating censoring firewalls, DPI tools can take advantage of higher-order statistics in our encrypted channel to distinguish it from a Skype video call. We observed that there are second and third order statistics, discussed in the appendix, in the Skype traces. We ensure that SkypeMorph respects those higher-order statistics in the packets and timings it outputs. An alternative for preserving all the characteristics of the Skype video call is to use the output of the audio and video encoder shipped with the Skype software to generate the statistics. If a live video source is available, we can run the encoder on it while the Tor bridge connection is in progress, and use the resulting packet sizes and inter-packet delays output by the encoder directly (replacing the encrypted packet contents with our own encrypted Tor data, of course). In this way, we can ensure that our traffic accurately mimics that of a real Skype video call.

5. SKYPEMORPH ARCHITECTURE

Skype, like any other instant messaging or voice and video calling/conferencing application, performs an authentication step before it allows a user to join the network. A user needs to sign up with the Skype website and obtain a username and password for authentication. The user then inputs these credentials to the Skype software to use them in the authentication process. After the user authenticates himself to the network, he is able to make calls or send messages. Due to the proprietary nature of the Skype protocol, it is unclear

how the login process is initiated and proceeds. The same is true for the call setup phase. To look like Skype as much as possible, SkypeMorph uses the actual Skype application to perform these actions.

In order to be able to use Skype network, we used Skype APIs which enable programmers to log in to the Skype network and have almost the same functionality as the Skype application, including making voice and video calls and sending files and text messages. Skype APIs come in two flavours, namely the `SkypeKit` [5] API that has a separate runtime executable (which must be purchased from Skype) and can operate as a command line application, and the Skype Public API, which can speak to any running instance of the usual Skype application through message passing systems such as DBus. These APIs allow us to perform the login and call initiation processes. The basic setup is discussed next and details of our implementation will appear in Section 6.

5.1 Setup

- **Step 1**: The bridge, which we denote by S, selects a UDP port number, P_S and uses the Skype APIs to log in to the Skype network, with a predefined set of credentials.[6] After successfully logging in to Skype, the bridge will listen for incoming calls. The bridge makes its Skype ID available to clients in much the same way that bridges today make their IP addresses and port numbers available — using Tor's BridgeDB [6] service, for example.
- **Step 2**: The client, denoted by C, picks a UDP port number, P_C and uses the same method to log in to the Skype network, using its own credentials.
- **Step 3**: The client generates a public key PK_C. After that, it checks to see whether the bridge is online and sends a Skype text message of the form $PK_C : IP_C : P_C$, to the bridge, where IP_C is the IP address of the client.
- **Step 4**: Upon receiving the message from the client, the bridge generates a public key PK_S and sends the following text message $PK_S : IP_S : P_S$ back to the client.
- **Step 5**: The bridge and client each compute a shared secret using the public keys they obtained and the client sends a hash of resulting key to the bridge.
- **Step 6**: The bridge then checks the received hash and if it matches the hash of its own secret key, it sends a message containing "`OKAY`".
- **Step 7**: If step 6 is successful and the client receives `OKAY`, it initiates a Skype video call to the bridge. Otherwise, it falls back to step 3 after a timeout.
- **Step 8**: The client keeps ringing for some random amount of time, then drops the call.

[6]Skype allows multiple logins, so it might seem reasonable to share the same username and password for every bridge. However, in that case all the messages sent to a certain Skype ID will be received by all the bridges currently logged in with that ID, which is an undesirable setting. We therefore require that every bridge has its own exclusive credentials, which are made available to the SkypeMorph bridge software on startup.

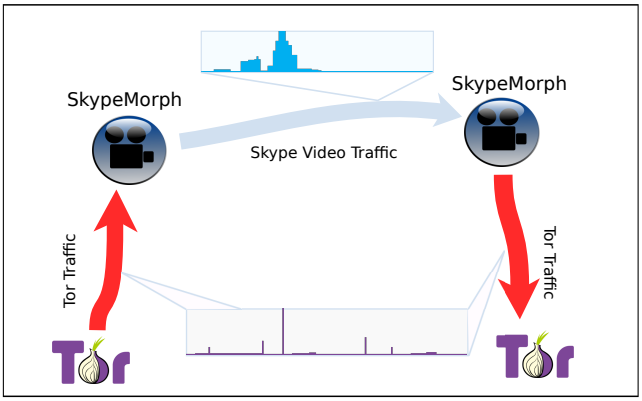


Figure 3: Overview of the SkypeMorph Architecture. The histograms show the distribution of packet sizes in Tor (at the bottom) and Skype video (at the top).

- **Step 9**: When the bridge notices the call is dropped it listens for incoming SkypeMorph messages on port P_S. The bridge selects another UDP port for the Skype runtime for it to listen for other incoming connections.
- **Step 10**: Afterwards, the client uses the shared key and the UDP port obtained in previous steps to send data.

Note that having the bridge switch to a different UDP port for the next client connection should not arouse suspicion since, as discussed in Section 3, this is how normal Skype calls to multiple users behind NAT would appear to the censor.

5.2 Traffic Shaping

Tor sends all of its traffic over TLS. We do not change this; rather, we just treat the TLS data as opaque, and send the TLS data over our own encrypted channel, masquerading it as Skype video. This means that the data is encrypted by Tor (multiple times), by TLS, and also by SkypeMorph.

After the above connection setup, it is possible to send the re-encrypted TLS messages through the established channel. As discussed in previous sections, in order to maximize the resemblance to real Skype traffic, we modify the output of our application to closely match that of Skype. The modification is done on the packet sizes and inter-arrival times of consecutive packets. For the packet sizes, two scenarios are considered: In the first scenario, we obtain our resulting packet sizes using the naïve traffic shaping method discussed in Section 4.2. We use the higher-order statistics mentioned above to produce joint probability distributions for the next inter-packet delay and packet size, given the values outputted previously. We sample from this conditional distribution to produce the delay and size of the next packet.

For the second scenario, the Traffic Morphing method of Section 4.3 is used; however, this method only supports first-order statistics.

The overview of the SkypeMorph architecture is shown in Figure 3. The red arrows represent the Tor traffic. On one side the Tor traffic is passed to SkypeMorph, where the traffic shaping mechanisms morph the traffic to resemble a

Skype call on the wire. On the other side the Tor traffic is reconstructed.

6. IMPLEMENTATION

In this section we describe our prototype implementation of SkypeMorph on Linux, which is implemented in C and C++ with the boost libraries [1]. The prototype is built into two executable files called `smclient` and `smserver`, which realize the client C and the bridge S of the previous section, respectively. We describe in the following the two phases in the execution of our prototype, namely, the setup and the traffic-shaping phases.

6.1 Setup Phase

As in the previous section, both the client and the bridge log into Skype using the Skype API, exchange public keys, and then start their Skype video conversation. As we mentioned in Section 4, there are various TCP connections accompanied with the Skype video conversation, which stay active even after the conversation is finished. In order to retain these TCP connections, our prototype implementation performs the following tricks:

- A TCP transparent proxy component is built into our prototype, which transparently relays all TCP connections the Skype runtime outputs and receives. We take advantage of the `TPROXY` extension in `iptables`, available in the Linux kernel since version 2.6.28. Corresponding `iptables` rules are created when our prototype starts execution.

- As a consequence of retaining the Skype TCP connections, the Skype runtime has to stay active during the SkypeMorph session. Hence the UDP port P_C or P_S will still be assigned to the Skype runtime when SkypeMorph starts to tunnel Tor traffic; therefore, SkypeMorph has to operate on another UDP port. To overcome this, the Skype runtime on the client operates on another UDP port, say P'_C, and `smclient` creates an `iptables` rule R_C with SNAT target to alter the source port of all Skype UDP packets from P'_C to P_C. When `smclient` starts to tunnel Tor traffic, it first deletes rule R_C, and it then operates on UDP port P_C. On the bridge side, the Skype runtime operates on UDP port P_S, and `smserver` runs on another UDP port P'_S. When `smserver` starts to communicate with `smclient`, it first creates an `iptables` rule R_S that redirects traffic towards UDP port P_S to P'_S; it then runs on P'_S. Note that SkypeMorph starts its tunneling task only when the Skype video call is finished; thus the `iptables` rules R_C and R_S affect only the client Skype runtime and the bridge SkypeMorph. This prevents the censor from noticing port changes between the genuine Skype video call traffic and the SkypeMorph traffic.

For the cryptographic features, we use the `curve25519-donna` [27] library to generate elliptic-curve Diffie-Hellman keys shared between `smclient` and `smserver`. Each SkypeMorph instance derives four keys: two for outgoing and incoming message encryption, and another two keys for outgoing and incoming message authentication purposes.

6.2 Traffic-shaping Phase

In the traffic-shaping phase, an `smclient` and `smserver` pair can be viewed together as a SOCKS proxy that relays streams between a Tor client and a Tor bridge. Between `smclient` and `smserver`, bytes in Tor streams are exchanged in segments by a simple reliable transmission mechanism over encrypted UDP communication, and they are identified by *sequence numbers*. Reliable transmission is supported by acknowledgments over sequence numbers. The cryptography functions are provided by CyaSSL[7], a lightweight SSL library also used in `SkypeKit`. We give more details below.

SkypeMorph UDP Packet Layout

First, we present the layout of the SkypeMorph UDP packets transmitted between `smclient` and `smserver`. We set the maximum size of a single packet to be 1460 bytes to avoid packet fragmentation, because 1500 bytes, including the IP header, is a common MTU over the Internet. Thus, besides a fixed 8-byte UDP header, each packet contains up to 1452 bytes of SkypeMorph data. The first 8 bytes are an HMAC-SHA256-64 message authentication code for the remaining bytes in the packet. We use the 256-bit AES counter mode stream cipher algorithm to encrypt the rest of the packet, which, prior to encryption, is formatted into five fields:

- `type:` This 1-byte field denotes the purpose of the packet. Currently there are two types: regular data and a termination message; the latter is used to inform the packet receiver to terminate the communication session.

- `len:` This is a 16-bit unsigned integer denoting the size of the contained Tor stream segment. This allows the packet receiver to discard the padding data.

- `seq:` This field contains the sequence number, a 32-bit unsigned integer, of the first byte of the contained Tor stream segment.

- `ack:` This field contains the *ack number*, a 32-bit unsigned integer used to identify those bytes that have been properly received.

- `msg:` This field is of length up to 1425 bytes and contains a Tor stream segment (of length `len`, above) and the padding data (taking up the rest of the packet).

We output the MAC, followed by the random AES initial counter, and then the encrypted payload, as seen in Figure 4.

Figure 4: SkypeMorph UDP packet body layout, where the size (in bytes) is under the name for each field. The shaded parts are encrypted using 256-bit AES counter mode. All bytes after the mac field are included in the HMAC-SHA256-64 computation.

Traffic Shaping Oracle

[7] `http://www.yassl.com/yaSSL/Home.html`

We next discuss the traffic shaping oracle component, which controls the sizes and timings of each successive UDP packet to be sent. We implement both the naïve and the Traffic Morphing methods in the oracle to compare them. The goal of the oracle is to provide traffic shaping parameters.

When using the naïve method, the oracle first reads the n^{th}-order distributions of the packet sizes and inter-packet delays of Skype traffic, as noted in the appendix. We currently have gathered data for up to $n = 3$, but nothing in principle prevents us from gathering more. For each query, the oracle remembers the last n answers $x_1, \ldots, x_n$, where x_n is the last packet size output, and the x_i alternate between packet sizes and inter-packet delay times. It then selects the n^{th}-order distribution X of inter-packet delays, conditioned on the values of $x_1, \ldots, x_n$, and randomly draws an inter-packet delay t_e from X. Next the size of the packet s_e is outputted similarly from the distribution X' of sizes, where X' depends on $x_2, \ldots, x_n, s_e$. The oracle responds to the query with the pair (s_e, t_e).

For the traffic morphing method, the oracle first reads distributions of the packet sizes of the Tor traffic, the inter-packet delays of the Skype traffic, and a pre-computed morphing matrix. We use the `morpher` library from the Morpher Project[8] to compute the expected packet size s_e. The Traffic Morphing `morpher` library does not take timings into account. It expects to receive a packet input to it, and to send out that packet immediately, possibly padded to a new size to emulate the target distribution. As such, the packet *timing* distribution of the output of Traffic Morphing is identical to its input distribution, which is not what we want. We need to decouple the arriving packets from the sent packets, so arriving data is placed into a buffer, and we adopt the technique of the naïve method to sample from the packet timing distribution to yield t_e. The oracle randomly selects a packet size s_o from the Tor traffic packet size distribution, and calls the `morpher` library to compute the output packet size s_e. The pair (s_e, t_e) is then the answer to the query.

Packetizer

The communication between `smclient` and `smserver` is handled by a *packetizer*, whose structure is shown in Figure 5. The purpose of the packetizer is to relay Tor streams with UDP packets such that the traffic exposed to the censor is indistinguishable from that of Skype video calls.

The data stream received from Tor over the pluggable transport SOCKS connection is first buffered in a *sending buffer*, and then retrieved in segments corresponding to the sizes produced by the traffic shaping oracle. On the other end, the received Tor stream segments are rearranged in order in a *receiving buffer* according to their sequence numbers, and then form the incoming Tor stream.

The packetizer creates two threads, `t_send` and `t_recv`, such that:

- Thread `t_send` first queries the oracle for the expected packet size s_e and delay t_e. It then checks the sending buffer to determine if any re-transmission is needed, and it locates and reads up to s_e bytes from the sending buffer. Currently re-transmission is triggered when three duplicated ack numbers are received, which is an approach found in most TCP implementations. Then an encrypted UDP packet of size s_e is created with any necessary random padding bytes. Next, `t_send` sleeps for time t_e and then sends the packet out.
- Thread `t_recv` is blocked until a new UDP packet is received. It decrypts the packet to get a Tor stream segment, which is then pushed into the receiving buffer. The receiving buffer returns the sequence number seq_r of the last byte that is ready to be committed to the TCP stream. Similar to TCP, seq_r+1 is used as the ack number to be sent in the next outgoing UDP packet. Any in-order segments that have been received are delivered to Tor over the pluggable transport SOCKS connection and are removed from the receiving buffer.

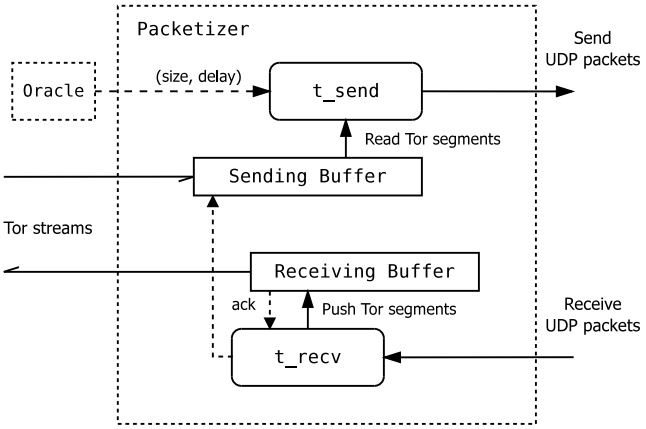


Figure 5: Structure of the packetizer.

As we observed from Skype video call traffic, when the network bandwidth is limited, the distributions of packet sizes and inter-packet delays change accordingly. To mimic this behaviour, our prototype first determines bandwidth changes by measuring the number r of re-transmissions occurring per second. Based on r, the oracle selects the most relevant distributions and computes the traffic shaping parameters. The dependence on r can be tuned through experiments to match the most relavent distributions.

As outlined above, our current implementation uses a simple TCP-like acknowledgement and retransmission scheme to ensure the in-order delivery of the underlying Tor data. An attacker may attempt to disrupt this scheme by dropping some fraction of *all* Skype video traffic. This will cause a modest decrease in the quality of actual Skype video conversations, but may cause a disproportionate decrease in SkypeMorph's effective throughput due to repeated retransmissions. We anticipate that a more advanced reliable transport algorithm, such as one using selective acknowledgements, may help to ameliorate this issue.

7. EXPERIMENTS

We performed our experiment in two parts. First we captured network traces of the Skype application to form a better understanding of how it operates. Our experimental testbed for this part consisted of several hosts running different operating systems, including Microsoft Windows, Linux and mobile devices. Using these traces we were able to obtain an empirical distribution of packet sizes and inter-packet arrival times for Skype video calls, which were used

[8] `https://gitorious.org/morpher/morpher`

as input to the SkypeMorph traffic-shaping oracle for drawing samples and generating the morphing matrix. Next, we used the SkypeMorph proxy for browsing and downloading over a bridge set up on a node in the same local network as the Tor client.

Figure 6 shows the cumulative distributions of packet sizes and inter-arrival delays between consecutive packets both for SkypeMorph (both with the naïve and enhanced Traffic Morphing traffic shapers) and for the original Tor distribution and the Skype video call distribution we obtained in the first part of the experiment. The graphs depict how closely the SkypeMorph output follows that of the Skype video calls, both in packet sizes and inter-packet delays; indeed, all three lines overlap almost perfectly. Also, the Kolmogorov-Smirnov test [34] for both the packet sizes and inter-packet delays shows no statistically significant difference between the Skype video and SkypeMorph distributions, using either the naïve traffic-shaping or the enhanced traffic morphing methods. This is of course expected, as the traffic shaping oracle is designed to match the Skype distribution. In addition, using the naïve traffic-shaping method, we also match the higher-order Skype traffic distributions. The distribution of regular Tor traffic, however, is considerably different, and this shows the utility of our method. The original Traffic Morphing [46] technique does not take into account timings, so its distributions would match Skype video for the packet sizes, but regular Tor traffic for the inter-packet timings.

In order to evaluate the performance of SkypeMorph, we tried downloading the `g++` Debian package from a mirror located in South America[9] by directly connecting to a bridge operated by us at a rate of 40–50 KB/s. Using the same bridge, we downloaded the file over SkypeMorph, using each of the naïve traffic shaping and enhanced Traffic Morphing methods, and compared the average download speed, network bandwidth used, and overhead percentage. We repeated the experiment 25 times for each method. The results are given in Table 1.

The overhead given is the percentage that the total network bandwidth (including TCP/IP or UDP/IP headers, retransmissions, padding, TLS, etc.) exceeds the size of the file downloaded. Although the very high variance makes it hard to see just by comparing the summary statistics in the table, the raw data shows that normal bridge traffic consistently incurs a 12% overhead, due to overheads incurred by Tor, TLS, and TCP/IP. The overhead of SkypeMorph is a little more than twice that; we incur the extra cost of sending padding when not enough data is available to fill the packet size informed by the traffic shaping oracle. We see that the naïve traffic shaping method and the enhanced Traffic Morphing method perform very similarly; indeed, the Kolmogorov-Smirnov test reports that there is no statistically significant difference between the results of those two methods ($p > 0.5$). Do note, however, that this overhead includes no *silent periods*, i.e., times for which we have no Tor traffic in our buffer, and so everything sent on the wire is padding. Taking these silent periods into account, the overhead is increased by the current bandwidth usage (in this experiment, about 43 KB/s). This is the same behaviour as an ordinary Skype video call; data is transmitted at an approximately constant rate, whether or not the participants are actually communicating.

[9] `http://ftp.br.debian.org/debian/pool/main/g/gcc-4.4/g++-4.4_4.4.5-8_i386.deb`

8. DISCUSSION AND FUTURE WORK

Overhead. As seen in the previous section, we found that when inter-packet timing is introduced to the Traffic Morphing technique, the traffic becomes less distinguishable from Skype traffic (the target distribution), but it also becomes less effective in reducing the overhead. The overhead in SkypeMorph is highly dependent on how much Tor traffic is available to the proxy. SkypeMorph will always send the same amount of data as a real Skype video connection would; if there is not that much useful Tor traffic to send, the rest is padding. Hence, if we experience many silent periods—when the proxy's sending buffer is empty—the overhead grows due to the padding sent by the proxy.

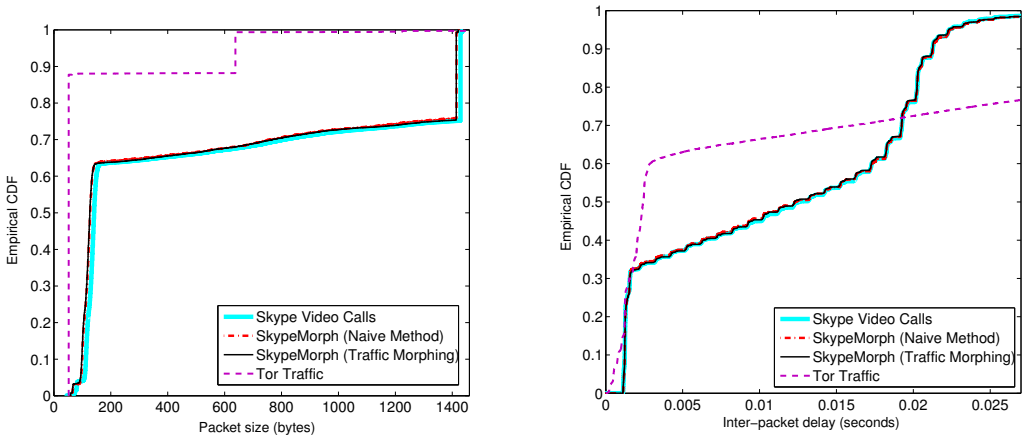
Mobile Bridges. A side advantage of SkypeMorph is that bridges can easily change their IP addresses and ports, without having to re-distribute contact information to clients or the BridgeDB. With SkypeMorph, all a client needs to know to contact a bridge is its Skype ID. This makes it harder for censors to block bridges, even once they are found.

Skype Protocol. `SkypeKit` allows peers to exchange streaming data through the Skype network. However, the data sent to the Skype network might be relayed by other nodes in the network and this can impose an overhead on the Skype network, which is not desired. Therefore, we deliberately chose not to use this feature of `SkypeKit`. SkypeMorph data is sent directly from the client to the bridge; it is *disguised* as Skype data, but it is *not* sent over the Skype network.

Attacks on SkypeMorph. In order to be able to block a SkypeMorph bridge, the censor either needs to totally ban Skype communications, or it has to verify the existence of SkypeMorph on a remote Skype node. This is only possible if the censor already knows the IP address or Skype ID of the bridge, which we have already excluded from our threat model. Also note that although we are not trying to prevent threats that may arise if the content of a SkypeMorph handshake is disclosed by Skype, it is still possible to use steganographic methods to hide the handshake in innocuous-looking messages.

The censoring authority can as well run its own SkypeMorph bridges and distribute its descriptor to users. Even though this is possible, users' privacy is not threatened because of this, as they are still selecting their own relays and connecting to the Tor network. Therefore, the censor can only detect that a user is connected to its own instance of SkypeMorph.

SkypeMorph and Other Protocols. Our current implementation of SkypeMorph is able to imitate arbitrary encrypted protocols over UDP. The target protocol, Skype in our case, can be replaced by any encrypted protocol that uses UDP as long as distributions of packet sizes and inter-arrival times are available. The source protocol, Tor, can also be replaced by an arbitrary TCP protocol. Note that if Traffic Morphing is going to be used, the morphing matrix needs to be recalculated for every pair of source and target protocols based on their distributions. Moreover, the current formulation of Traffic Morphing is not amenable to higher-order statistics. However, if naïve traffic shaping is used, the system is actually completely independent of the source protocol and it is possible to mimic higher-order statistics.



(a) Packet size distribution

(b) Inter-packet delays distribution

Figure 6: Experimental cumulative distributions for Skype video, SkypeMorph, and Tor. We show packet size and inter-packet delay distributions in (a) and (b) respectively.

Table 1: Download speed (goodput), network bandwidth used, and overhead imposed by (a) Normal Tor-over-TCP, (b) SkypeMorph-over-UDP with naïve traffic shaping, and (c) SkypeMorph-over-UDP with our enhanced Traffic Morphing.

	Normal bridge	SkypeMorph (Naïve Shaping)	SkypeMorph (Traffic Morphing)
Goodput	200 ± 100 KB/s	33.9 ± 0.8 KB/s	34 ± 1 KB/s
Network bandwidth used	200 ± 100 KB/s	43.4 ± 0.8 KB/s	43.2 ± 0.8 KB/s
Overhead	$12\% \pm 1\%$	$28\% \pm 2\%$	$28\% \pm 3\%$

SkypeMorph Software. Our current proof-of-concept implementation targets the Linux operating system. We would naturally like to extend the work to support Windows and other platforms. The only obstacle is the port redirection described in Section 6.1. This can be handled either with native firewalling support, or possibly by running Linux-based software on a Tor-aware home router.[10]

Also as discussed in Section 4.4, we plan to experiment with using the audio and video encoders shipped with the Skype in order to more easily match Skype's packet size and timing patterns perfectly.

9. CONCLUSIONS

We have presented SkypeMorph, a pluggable transport for Tor that disguises client-to-bridge connections as Skype video traffic. We present two methods to morph Tor streams into traffic with indistinguishable packet sizes and timings to Skype video; the first method uses naïve traffic shaping to emulate the target distribution, independent of the source distribution. The second method takes the source distribution into account, enhancing Wright et al.'s Traffic Morphing [46] to also account for packet timings. The two methods have statistically similar performance, but the naïve traffic shaping method is much easier to implement, is unaffected by a changing source distribution, and can match the higher-order patterns in Skype traffic. While our methods are effective at matching the desired distributions, they come at some cost in extra bandwidth used between the client and the bridge—but no more so than if an actual Skype video call were in progress. Our software is freely available, and is easily adaptable to other encrypted UDP-based target protocols.

[10] `https://trac.torproject.org/projects/tor/wiki/TheOnionRouter/Torouter`

Acknowledgements

The authors wish to thank George Kadianakis, Paul Syverson, and the anonymous reviewers for their helpful discussions and suggestions on how we can improve our work. We also thank NSERC, ORF, Mprime, and The Tor Project for funding this work.

10. REFERENCES

[1] Boost C++ Libraries. `http://www.boost.org/`. [Online; accessed April 2012].

[2] Does Skype use encryption? `https://support.skype.com/en/faq/FA31/Does-Skype-use-encryption`. [Online; accessed April 2012].

[3] Number of Skype users in 2010. `http://about.skype.com/press/2011/05/microsoft_to_acquire_skype.html`. [Online; accessed April 2012].

[4] Skype Software. `http://skype.com`. [Online; accessed April 2012].

[5] SkypeKit. `http://developer.skype.com/public/skypekit`. [Online; accessed April 2012].

[6] Tor BridgeDB. https://gitweb.torproject.org/bridgedb.git/tree. [Online; accessed April 2012].

[7] Ethiopian Government Bans Skype, Google Talk And All Other VoIP Services. http://techcrunch.com/2012/06/14/ethiopian-government-bans-skype-google-talk-and-all-other-voip-services/, 2012. [Online, accessed May 2012].

[8] Skype ban in the UAE could be lifted, as it is "purely a licensing matter". http://thenextweb.com/me/2012/04/21/skype-ban-in-the-uae-could-be-lifted-as-it-is-purely-a-licensing-matter/, 2012. [Online, accessed May 2012].

[9] J. Appelbaum and N. Mathewson. Pluggable transports for circumvention. https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/180-pluggable-transport.txt. [Online; accessed April 2012].

[10] S. A. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–11, 2006.

[11] P. Biondi and F. Desclaux. Silver Needle in the Skype. BlackHat Europe, http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf, March 2006. [Online; accessed April 2012].

[12] G. Bissias, M. Liberatore, D. Jensen, and B. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11. 2006.

[13] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking Down Skype Traffic. In *INFOCOM 2008. 27th IEEE Conference on Computer Communications.*, pages 261 –265, 2008.

[14] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing Skype Traffic: When Randomness Plays with You. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 37–48, 2007.

[15] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, 2010.

[16] R. Chandramouli and N. Memon. Analysis of LSB based image steganography techniques. In *Proceedings of International Conference on Image Processing*, volume 3, pages 1019 –1022, 2001.

[17] R. Dingledine. Research problems: Ten ways to discover Tor bridges. https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges. [Online; accessed April 2012].

[18] R. Dingledine. Iran blocks Tor; Tor releases same-day fix. https://blog.torproject.org/blog/iran-blocks-tor-tor-releases-same-day-fix, Sept. 2011. [Online; accessed April 2012].

[19] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th conference on USENIX Security Symposium*, SSYM'04, pages 303–320, 2004.

[20] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli. Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 53(1):81 – 97, 2009.

[21] M. Enev, S. Gupta, T. Kohno, and S. N. Patel. Televisions, video privacy, and powerline electromagnetic interference. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, CCS '11, pages 537–550, New York, NY, USA, 2011. ACM.

[22] N. Hopper, L. von Ahn, and J. Langford. Provably Secure Steganography. *IEEE Transactions on Computers*, 58(5):662 –676, 2009.

[23] A. Houmansadr, G. T. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, CCS '11, pages 187–200, 2011.

[24] N. F. Johnson, Z. Duric, and S. Jajodia. *Information hiding: Steganography and watermarking—attacks and countermeasures*. Kluwer Academic Publishers, 2000.

[25] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI 2011)*, August 2011.

[26] M. Krzywinski. Port knocking: Network authentication across closed ports. *SysAdmin Magazine*, 12(6):12–17, 2003.

[27] A. Langley. curve25519-donna. http://code.google.com/p/curve25519-donna/. [Online; accessed April 2012].

[28] C. S. Leberknight, M. Chiang, H. V. Poor, and F. Wong. A Taxonomy of Internet Censorship and Anti-Censorship. http://www.princeton.edu/~chiangm/anticensorship.pdf. [Online; accessed April 2012].

[29] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *13th ACM conference on Computer and Communications security*, pages 255–263, 2006.

[30] N. Mathewson. The Tor Project, A simple obfuscating proxy. https://gitweb.torproject.org/obfsproxy.git. [Online; accessed April 2012].

[31] W. Mazurczyk, P. Szaga, and K. Szczypiorski. Using Transcoding for Hidden Communication in IP Telephony. *ArXiv e-prints*, Nov. 2011.

[32] J. McLachlan and N. Hopper. On the risks of serving whenever you surf: Vulnerabilities in TorâĂŹs blocking resistance design. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 31–40, 2009.

[33] S. J. Murdoch. Moving Tor to a datagram transport. https://blog.torproject.org/blog/moving-tor-datagram-transport. [Online; accessed April 2012].

[34] NIST/SEMATECH. e-Handbook of Statistical Methods. http:

//www.itl.nist.gov/div898/handbook/index.htm. [Online; accessed April 2012].

[35] F. Petitcolas, R. Anderson, and M. Kuhn. Information Hiding - A Survey. *Proceedings of the IEEE, special issue on protection of multimedia content*, pages 1062 –1078, 1999.

[36] I. M. Program. Persian cyberspace report: internet blackouts across Iran;. http://iranmediaresearch.com/en/blog/101/12/02/09/840. [Online; accessed April 2012].

[37] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. Devices that tell on you: privacy trends in consumer ubiquitous computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 5:1–5:16, Berkeley, CA, USA, 2007. USENIX Association.

[38] G. J. Simmons. The Prisoners Problem and the Subliminal Channel. In *Advances in Cryptology: CRYPTO 1983*, pages 51–67. Springer, 1984.

[39] R. Smits, D. Jain, S. Pidcock, I. Goldberg, and U. Hengartner. BridgeSPA: Improving Tor bridges with Single Packet Authorization. In *Proceedings of the 10th annual ACM workshop on Privacy in the Electronic Society*, WPES '11, pages 93–102, 2011.

[40] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and Detecting Skype-Relayed Traffic. In *INFOCOM 2006. Proceedings of 25th IEEE International Conference on Computer Communications.*, 2006.

[41] The Tor Project. Tor Metrics Portal: Users. https://metrics.torproject.org/users.html, 2012. [Online, accessed July 2012].

[42] A. White, A. Matthews, K. Snow, and F. Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on fon-iks. In *IEEE Symposium on Security and Privacy (SP)*, pages 3 –18, 2011.

[43] T. Wilde. Knock Knock Knockin' on Bridges' Doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors, Jan. 2012. [Online; accessed April 2012].

[44] P. Winter and S. Lindskog. How China Is Blocking Tor. Technical report, Karlstad University, 2012.

[45] C. Wright, L. Ballard, S. Coull, F. Monrose, and G. Masson. Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations. In *IEEE Symposium on Security and Privacy (SP)*, pages 35 –49, 2008.

[46] C. Wright, S. Coull, and F. Monrose. Traffic Morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE, February 2009.

[47] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of 16th USENIX Security Symposium*, 2007.

[48] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the Network Infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

APPENDIX

In this section we examine the higher-order statistics in Skype video calls.

- **Second-order Statistics**: We explored the space of second-order statistics in Skype calls by measuring ordered pairs (s_i, t_i) where s_i is the size of a packet sent on the wire and t_i is the delay until the next packet. For all possible packet sizes the empirical shown in Figure 7a.[11]

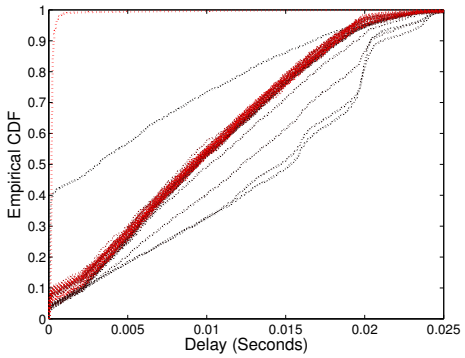
  The same experiment was repeated with all (t_i, s_i) pairs with t_i being the delay and s_i the packet size sent on the wire after t_i milliseconds. As Figure 7b shows, smaller delays tend to result in larger packet sizes following them.

- **Third-order Statistics**: For third-order statistics we considered tuples (s_i, t_i, s'_i), where s_i and s'_i are consecutive packet sizes and t_i is the delay between the two packets. Thus, for all possible s'_i we formed the distribution $P[s'_i|(s_i, t_i)]$ and compared them. Figure 7c shows this distribution for a fixed t_i, demonstrating that there are third-order statistics not explained by the second-order statistics.

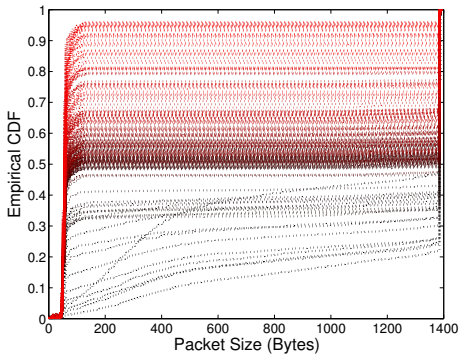
  Similarly, we also considered tuples of the form (t_i, s_i, t'_i). Figure 7d shows the distribution of $P[t'_i|(t_i, s_i)]$ for a fixed s_i, again revealing nontrivial third-order statistics.

- **Higher-order Statistics**: Extending this method to higher orders is a straightforward task and depending on the precision needed, we can find these statistics up to the desired order. This allows us to fully mimic the traffic characteristics of a Skype call.

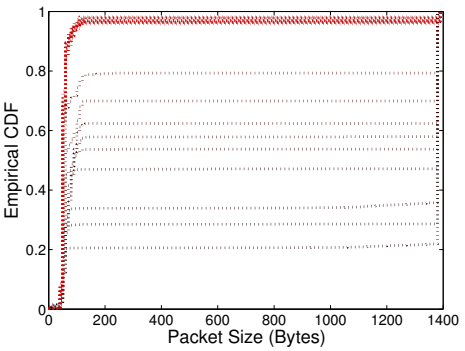
[11] As the space of all possible tuples was huge, we binned the packet sizes and delays to make the figure easier to read.



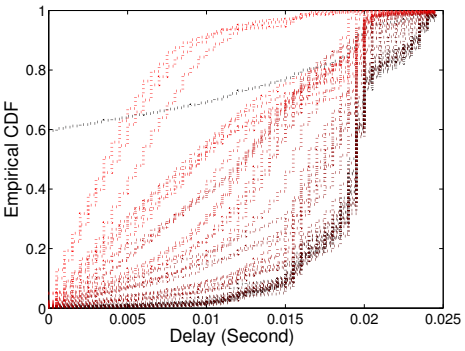
(a) Second-order statistics (size - delay)



(b) Second-order statistics (delay - size)



(c) Third-order statistics (size - delay - size)



(d) Third-order statistics (delay - size - delay)

Figure 7: Second and third-order statistics of a Skype video call for arbitrary bins of size and delay.