SkypeMorph: Protocol Obfuscation for Censorship Resistance

by

Hooman Mohajeri Moghaddam

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The Tor network is designed to provide users with low-latency anonymous communication. Tor clients build circuits with publicly listed relays to anonymously reach their destinations. Low-latency anonymous communication is also an essential property required by censorship circumvention tools and thus Tor has been widely used as a censorship resistance tool. However, since the Tor 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*.

In recent years, there have been attempts to achieve fast and real-time methods to discover Tor, and specifically bridge, connections. In this thesis we address the issue of preventing censors from detecting a certain type of traffic, for instance Tor connections, by observing the communications between a remote node and nodes in their network.

We propose a generic 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.

Although our method is generic and can be used by any censorship resistance application, we present it for Tor, which has well-studied anonymity properties. We have implemented our model as a proof-of-concept proxy that can be extended to a pluggable transport for Tor, and it is available under an open-source licence. Using this implementation we observed the obfuscated bridge communications and showed their characteristics match those of Skype calls. We also compared two methods for traffic shaping and concluded that they perform almost equally in terms of overhead; however, the simpler method makes fewer assumptions about the characteristics of the censorship resistance application's network traffic, and so this is the one we recommend.

Acknowledgements

I would like to thank my supervisor, Ian Goldberg, for his always constructive suggestions, feedback and support. I also thank my co-authors of the paper (Baiyu Li, Mohammad Derakhshani and Ian Goldberg) that this work is extended from. Finally, I thank my thesis readers (Urs Hengartner and Srinivasan Keshav) for their time.

Dedication

This is dedicated to my parents and my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Tor Network and Bridges

Tor [DMS04] 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* (ORs). 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 1.1 shows the blocking of the whole Tor network by the Great Firewall of China as of 2010. This weakness indicated that relying on static information should not be considered an effective anti-censorship solution.

In order to counteract this problem, the Tor project proposed 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, they still suffer from some weaknesses; McLachlan and Hopper [MH09]

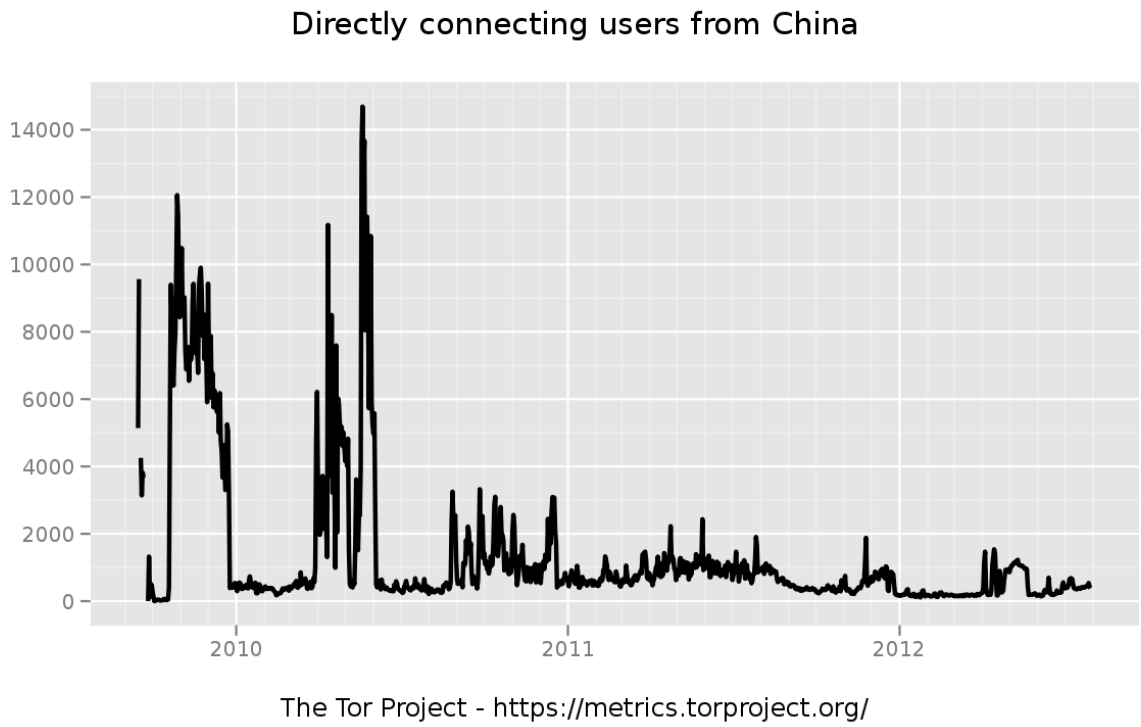


Figure 1.1: This graph from metrics.torproject.org shows the number of users directly connecting to the Tor network from China, from mid-2009 to the present. It shows that, after 2010, the Tor network has been almost completely blocked from clients in China who do not use bridges. [Tor12]

showed that it is still possible to identify them, as they accept incoming connections unconditionally. To solve this problem, BridgeSPA [SJP+11] places some restrictions on how bridges should accept incoming connections.

Another seemingly difficult challenge is how to stop censors from masquerading as several legitimate users and obtaining a list of all or a great portion of available Tor bridges and thus getting around the rate limiting mechanism currently in place for bridge discovery [Din11a]. Recent evaluations show that a possible solution is to employ a vast quantity of short-lived volunteer-operated proxies that makes the task of finding all entry points to the network almost impossible for the censor [FHE+12]. Our work though, emphasizes on another and rather orthogonal aspect of the bridge access problem, namely protocol identification and blocking, which we will discuss next.

1.2 Protocol Obfuscation and Pluggable Transports

As censorship techniques improve, more and 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 [Wil12, WL12]. 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 and blocking of Tor connections based on the expiry time of the SSL certificate generated by the Tor software [Din11b].

As the censorship arms race shifted toward the characteristics of connections, Appelbaum and Mathewson proposed a framework for developing protocol-level obfuscation plugins for Tor called *pluggable transports* [AM10]. 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 and build their own transports, hiding Tor traffic in other protocols. As shown in Figure 1.2, 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. 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 this thesis, there are only a few available pluggable transports,

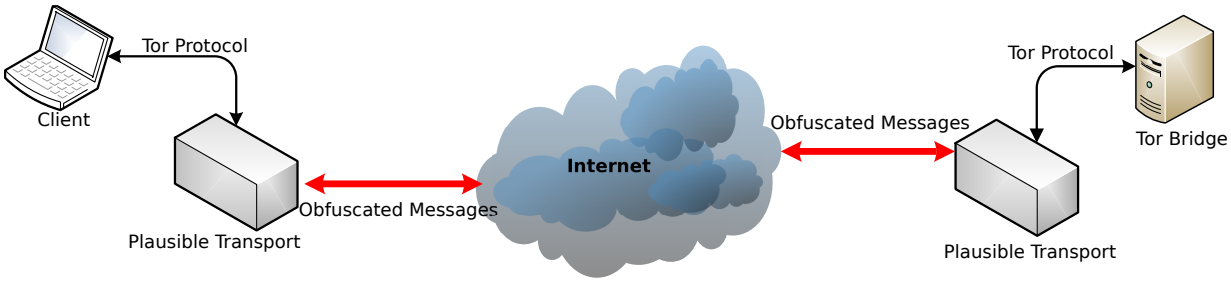


Figure 1.2: Pluggable transport overview: Messages are reformated or embedded in a cover protocol to bypass detection tools.

including "obfsproxy" [KM11], which passes all traffic through a stream cipher, though it does not disguise the sizes and timings of packets.

Recently though, we have witnessed more sophisticated methods of traffic camouflaging including StegoTorus [WWY+12], a tool designed for the purpose of hiding Tor traffic in widely used cover channels, such as HTML, Flash SWF files and PDF files. Similar to StegoTorus, we extend previous works to address their limitations of *not* outputting innocuous-looking traffic; compared to StegoTorus, our method greatly reduces the chances of obfuscated bridge connections being detected by powerful censors, while providing much larger bandwidth to Tor users.

For the purpose of our experiment we chose the Skype [Skyc] 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 [Skyb] 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 [Skya], so it provides an encrypted channel in which to hide Tor traffic.

We also note that simply adding the target protocol headers to packets would not be successful when facing deep packet inspection (DPI) methods [LCPW12]. Dusi et al. [DCGS09] 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.

1.3 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. Our method, to the best of our knowledge, is the first of its kind, designed for high-bandwidth and low-latency obfuscation of Tor 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 proxy. 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**: Our experiments showed that Skype uses UDP whenever a direct UDP connection is possible between participants in a call. Consequently in our tool we also chose UDP as our transport protocol. The choice of UDP as the transport protocol will also be useful when Tor datagram designs [Mur11] are rolled out. In addition, UDP is much more flexible than TCP for traffic shaping techniques, as explained in Section 5.2. This is mainly because UDP has no reliable delivery and congestion control built into it at the kernel level, allowing customized reliable transmission mechanisms to be implemented at the user level, which has the level of flexibility required for introducing the traffic shaping, discussed next.

- **Improved traffic shaping**: *Traffic Morphing* as proposed by Wright et al. [WCM09] 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 software can be used on Linux operating systems or on home routers that can run the OpenWrt software distribution.

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

Chapter 2

Related Work

2.1 Protocol Obfuscation and Anti-censorship Solutions

The idea of protocol obfuscation as a method of censorship resistance has been around for at least the past couple of years. Dust [Wil11b] is an early example of an obfuscation tool, where each connection is wrapped inside a layer of encryption to fully hide every textual aspect of the encapsulated protocol, including handshakes and protocol signatures. This ensures that no additional information is leaked from the patterns transmitted and traffic flows protected by Dust look random in the eyes of a censor inspecting byte patterns. Similarly, with the new pluggable transport framework of Tor, the idea is to support a wide range of transport or camouflaging techniques. The first officially released transport is obfsproxy [KM11]. Acting similarly to Dust, obfsproxy adds encryption to Tor's handshake and connetions. However, obfsproxy's modularity makes it possible to extend it to other custom-built transports, and therefore makes pluggable transport design and implementation much easier as new transports need only implement a subset of functionalities, such as the encryption mechanism.

Unlike the above-mentioned examples, where the traffic is modified to look like no other traffic, some systems choose to mimic other innocuous looking protocols. Infranet [FBH+02] is a censorship resistance system that leverages HTTP protocol messages to hide requests for censored materials and a combination of HTTP messages and arrays of uncensored images for downstream data. StegoTorus [WWY+12] has a smilar approach, hiding intended requests and responses in a range of target media such as PDF and Javascript

code. Additionally, the authors of StegoTorus introduce a modular construction for easy modification and replacement of different parts of the implementation. Overall, new methods are pushing the arms race to the limits of censors, engaging more attack surfaces [EG12] and consequently requiring many more technical capabilities and resources to beat, and eventually making the collateral damage harder to evade. However, high efficiency and bandwidth is still an unmet requirement for modern protocol obfuscation tools. Moreover, systems such as Infranet and StegoTorus do not provide the level of deniability required for anticensorship, since a censor spending enough time and computation can detect malformed web traffic; we will try to address these issues in this thesis.

In addition to the protocol obfuscation methods mentioned above, recent proposals have suggested systems specifically built for censorship resistance such as Telex [WWGH11], Cirripede [HNCB11] and Decoy Routing [KEJ+11]. While these systems can be effective, our method has the advantage that it does not require any cooperation from other intermediary points on the Internet, whereas those other anti-censorship systems have to rely on various ISPs deploying their software (and currently there is no clear market incentive for such tools to be deployed at a large scale).

2.2 Information Hiding and Steganography

Hiding information within subliminal channels has been studied extensively in the last three decades. Simmons [Sim84] 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 attracted a lot of attention in digital communications [PAK99].

Employing a steganographic technique, one needs to consider two major factors: the *security* and *efficiency* of the method. Hopper et al. [HvAL09] proposed a provably secure framework for evaluating steganographic protocols and a construction, proven to be secure under their model and applicable to a broad range of channels. The `OneBlock` stegosystem described in their work, however, requires having access to target protocol distribution and needs an expected number of samples from the channel that is exponential in the number of bits transmitted.

2.3 Image Steganography

Hiding information within pictures is a classic form of steganography. Least Significant Bit (LSB) based image steganographic techniques [JDJ00] — 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. [CM01] showed the upper bounds for the capacity of such channels.

Hiding information inside images might seem a straightforward task at first glance, however, there are numerous statistical analysis methods [WP00] capable of detecting anomalies in pictures and images. There are various image stegosystems resistant to such statistical attacks including OutGuess [Pro01]. In OutGuess, message embedding happens in two phases. For each image, first a subset of redundant bits of the image is detected, where redundant bits are those bits that changing them does not degrade image quality. Next, a fraction of these redundant bits are selected using a pseudo random number generator (PRNG) and encrypted messages are embedded into these bits. Steghide [HM05] is another tool that preserves first-order statistics by applying a graph-theoretic exchange of bit patterns; it is available as open-source software.

Recently, Collage [BFV10] was introduced as a 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.

All the image steganographic tools discussed in the literature suffer from low efficiency and also one major drawback of these systems is that they require a large image database that either has to be generated as the system is being used or should be gathered beforehand. Moreover, unless the transferred cover images are user generated, simply forwarding images taken from the Internet might arouse suspicion, if plausible deniability is required.

2.4 Voice over IP and Video Streaming

Wright et al. [WBMM07] 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 that 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 [WBC+08] or unmask parts of the conversation [WMSM11], when the same length-preserving encryption method is employed for confidentiality.

However, the statistical properties these attacks exploit seem to be less prevalent in streaming video data, rather than the audio data they consider; therefore, we have not witnessed any method capable of distinguishing our Tor traffic disguised as Skype video traffic from real Skype video traffic so far. 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 [EGKP11], or revealing which videos in a database are being viewed in a household by throughput analysis [SLH+07]. 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 [WCM09] exploits the packet size distribution of VoIP conversations to transmit hidden messages (we will return to this method in Chapter 4). Another example of steganographic communications over voice channels is TranSteg [MSS11], 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 most of the bandwidth is allocated to the actual voice conversation, leaving only a limited space for steganograms.

2.5 Steganography over Encrypted Channels

Although steganographic models similar to those mentioned above are powerful, they impose relatively large overheads on our channel. Therefore, we use a combination of methods suggested for encrypted communications [DCGS09, WCM09]. We argue that on an encrypted communication channel such as that used by 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 [BLJL06, LL06, DCGS09]. Hence, a protocol obfuscation layer only needs to reproduce these features for an encrypted channel.

Chapter 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 local or 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 [WWGH11], Cirripede [HNCB11] and Decoy Routing [KEJ+11]. In our threat model, as depicted in Figure 3.1, the censor has complete authority over the network structure in its jurisdiction, including routers, wireless access points, cables, etc. In particular, the censor is able to block access to Tor's publicly listed relays, and to detect certain patterns in Tor's traffic and hence block them. 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 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* [Krz03]. Bissias et al. showed how such heuristics can be employed to detect certain traffic patterns in an encrypted channel [BLJL06].

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"

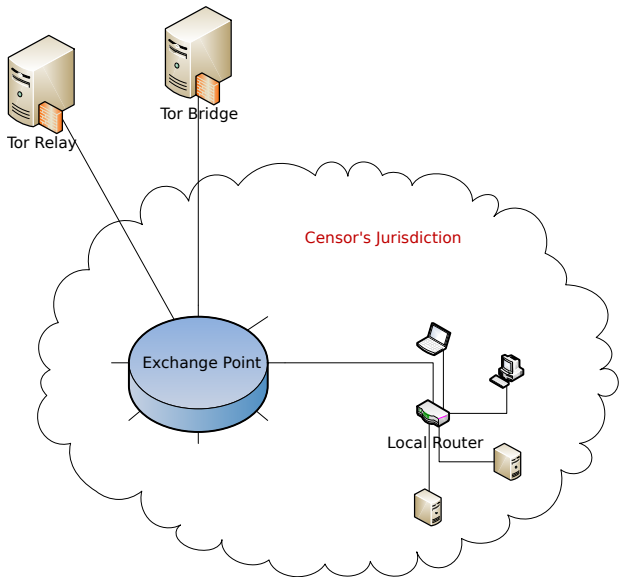


Figure 3.1: In our model, the censor has complete view and access to its internal network, but cannot influence nodes outside its jurisdiction.

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 over UDP, 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 banned in some countries [Nan12], to the best of our knowledge these instances are very rare and in most of such cases the protocol and the network are not blocked, but either the Skype website is filtered or users are threatened with legal actions [Lar12]. 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 [Skye]; 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.

Moreover, currently we are relying on Skype to maintain the confidentiality and integrity of text messages, but our extremely small size messages allow for other crypto-

graphic and steganographic tools to be used efficiently to add the desired level of security and privacy needed. In addition to this, another possible denial of service attack is to frequently drop Skype calls taking longer than a few minutes (or a certain threshold) to interrupt SkypeMorph users. This attack can be simply mitigated by hopping through SkypeMorph bridges periodically.

We further assume that the censor does not have access to information about particular bridges (thus it does not make sense to apply our method to Tor's public ORs since blocking public ORs based on their IP address is a trivial task), 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. Although there are other approaches for enumerating normal bridges [Din11a], they are outside the scope of this thesis; our work aims at defeating firewall and DPI tools that look for Tor flows.

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. Although the censors can set up their own SkypeMorph bridges and distribute their information, they can only obtain a list of users connecting to their SkypeMorph bridge and not much more, as anonymity is preserved by using Tor. Thus, by having a large number of SkypeMorph bridges serving users, we can reduce the chances of connecting to a SkypeMorph bridge run by the censor.

Moreover, our approach assumes that DPI boxes can investigate suspicious traffic flows with high granularity; therefore every aspect of a Skype video call needs to be reproduced by our tool, in order to evade such DPI tools.

In general, SkypeMorph aims to build a layer of protocol obfuscation for Tor bridges 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 Chapters 5 and 6.
- **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**: Unless a user connects to a SkypeMorph bridge operated by the censor, 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.

Chapter 4

Background

4.1 Skype

Skype [Skyc] 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 [Skyb].

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 [BS06, BD06]. The findings from these attempts and our own experiments, alongside some insights from the Skype developers and results of independent studies of the Skype source code and network [Ber05] have established the following facts:

- Skype encrypts messages using the AES cipher and uses RSA-based certificates for authentication [Skya, BD06]. Users' identities are verified and signed by Skype certificate authorities [Ber05]. During registration, each user sends its username along with a hash of its password and an RSA public key. The Skype central server issues a certificate containing the public key and the username. Before each communication over Skype, the user's application performs a session-establishment protocol and generates session keys. However, the details of these steps are unknown. 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 a manner that is not detectable.

All that is possible to such an attacker is selective packet dropping or denial of service. Moreover, although previous research suggests that there are some distinctive patterns in Skype packets [BMM+07], we could not verify these patterns in our experiments. In any case, if patterns are discovered later, they can be easily reproduced by our tool.

- 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 [BD06, BMMdT08, SFKT06].

- 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 that 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 [BS06, BD06].

- Skype has a variety of voice and video codecs and selects among them according to bandwidth, network speed and several other factors [BMMdT08, BMM+07].

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 Chapter 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 continuous streams 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 are major concerns in online applications. In order to be able 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 small number of possible bitrates [BMM+07]; however, as shown in Figure 4.1, Skype video calls seem to enjoy much more flexibility in terms of bandwidth usage.[1]

We ran a video call from a node running Linux to another remote node running Windows. Using the traffic control tool `tc` under Linux, we first gradually decreased the bandwidth limit on one side and observed the traffic on the other side. We then increased the bandwidth gradually to observe the effect. Results are shown in Figure 4.1a; they suggest that by limiting the available bandwidth, Skype's bandwidth usage drops significantly at each step to a level far below the available rate (this phenomenon 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 4.1b. Therefore, by a similar rate limiting technique, SkypeMorph is able to transfer data at a reasonable rate that complies with the bandwidth specified by the bridge operator.

4.2 Naïve Traffic Shaping

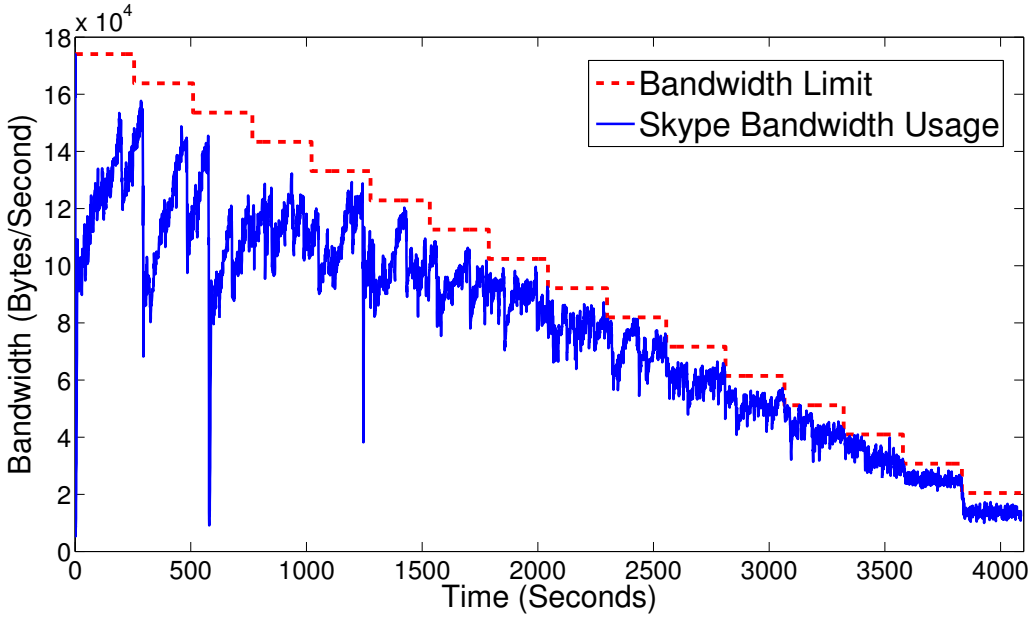
To achieve a similar statistical distribution of packet sizes in the output of our system to that of a target process, a basic approach would be to simply draw samples from the packet size distribution of the target process and send the resulting size on the wire. This method can be easily applied to the inter-packet arrival time distribution as well. Thus, if there is not enough data available from Tor to fill a packet, we need to add padding to the packet that can be as large as the whole payload size; i.e., we might have to send dummy packets.

An alternate approach is to consider the incoming packet sizes from the source distribution, which is how Traffic Morphing deals with the problem and is described next.

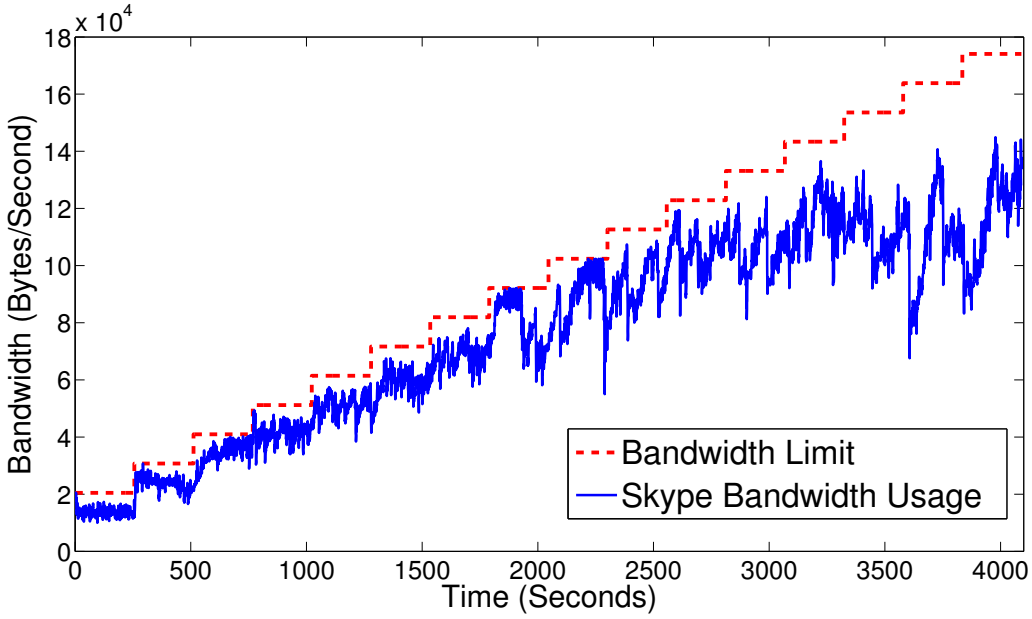
4.3 Traffic Morphing

We briefly mention how the original Traffic Morphing [WCM09] method works. Traffic Morphing attempts to counter an adversary who is trying to distinguish between traffic produced by a source process from that of a target process, through statistical means. As

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



(a) Decreasing bandwidth available to Skype



(b) Increasing bandwidth available to Skype

Figure 4.1: Bandwidth usage by Skype under different network situations. Figure 4.1a shows the drops in the Skype transfer rate while decreasing the network bandwidth and Figure 4.1b shows the increase in the rate.

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.

Let the vectors $S = [s_1, \ldots, s_n]^T$ and $S' = [s_1', \ldots, s_n']^T$ represent different packet sizes (in increasing order) in the source and target communication, respectively. Let the probability distribution of the source process be denoted by $X = [x_1, \ldots, x_n]^T$, where x_i is the probability of the s_i; similarly let $Y = [y_1, \ldots, y_n]^T$ denote the target process probability distribution. Traffic Morphing finds the matrix A for which we have $Y = AX$ such that the number of additional bytes needed to be transmitted is minimal. More formally, A is the stochastic morphing matrix that satisfies the following optimization problem [WCM09]:

$$\begin{aligned} \text{minimize} \quad & \sum_{i,j} x_j a_{ij} |s_i' - s_j| \\ \text{subject to} \quad & Y = AX \\ & \sum_{i=1}^{n} a_{ij} = 1 \\ & a_{ij} \geq 0 \end{aligned}$$

Thus, a packet with length s_j in the source will be morphed to a packet with length s_i' in the target with the probability a_{ij}, resulting in a probability distribution in every column of A. For each packet in the source process with size s_j we form the cumulative probability of j^{th} column of A and generate a random number $r \in (0, 1)$ and calculate the corresponding s_j'. Using this technique requires some considerations, for example dealing with larger sample spaces or overspecified constraints that 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 by introducing *inter-packet timing* to it as well. Once Traffic Morphing is combined with the timing techniques, it, like naïve traffic shaping, has to send dummy packets when no source data is available at the time a packet must be sent. Unfortunately, as we will see in Chapter 7, these dummy packets cause the reduction in overhead, as compared to naïve traffic shaping, to become negligible.

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 ran a test on the same testbed discussed in Section 4.1.1 but with no bandwidth limit and observed that there are second and third order statistics, discussed below, in the Skype traces.

- **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 CDF of delays following that packet size is shown in Figure 4.2a. In this figure, each line represents the CDF of delays following a given packet size.[2]

  The same method was applied to 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. In Figure 4.2b, the relationship between the inter-packet delays and the next packet size is shown and there is a negative correlation between the two. Consequently, these relationships should be considered when packets are shaped, which we will talk more about in Section 6.2.

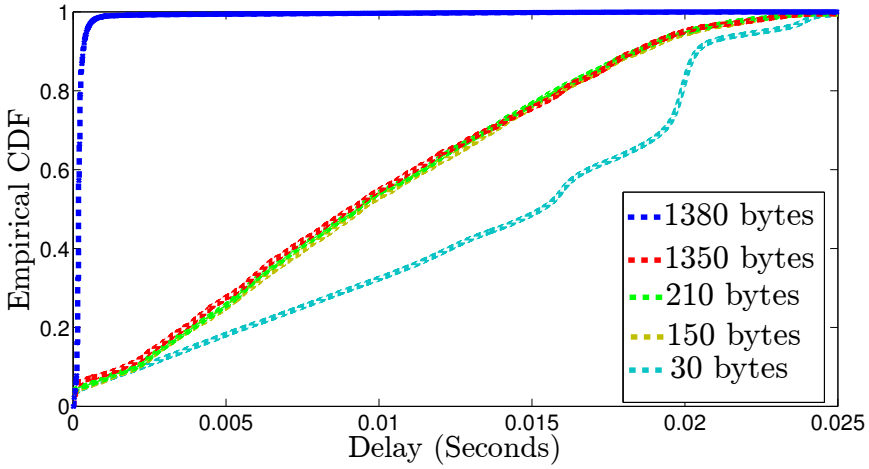
- **Third-order Statistics**: For third-order statistics we considered tuples (s_i, t_i, s'_i), where s_i and s'_i are consecutive packet sizes and t_i is the delay between the two packets. Thus, for all possible (s_i, t_i) we formed the distribution $P[s'_i|(s_i, t_i)]$ and compared them. Figure 4.3a 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 4.3b shows the distribution of $P[t'_i|(t_i, s_i)]$ for a fixed s_i, again revealing nontrivial third-order statistics.

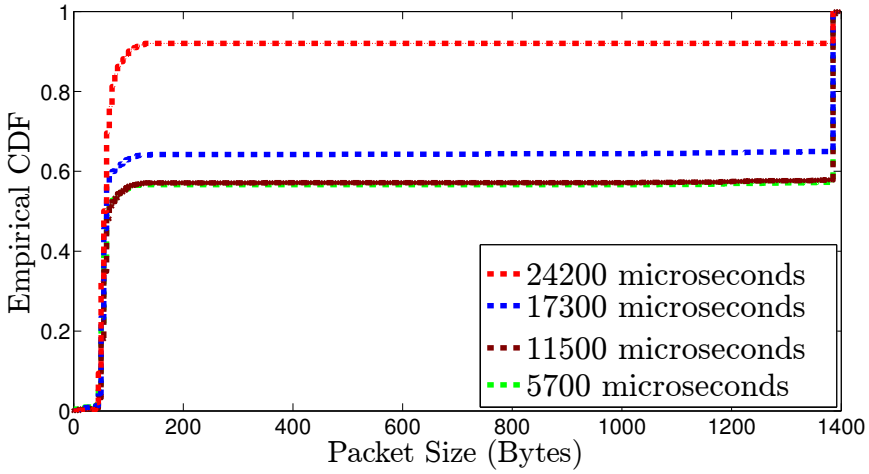
- **Higher-order Statistics**: Extending this method to higher orders is a straightforward task and depending on the precision needed, we can find these statistics up to the desired order. This allows us to fully mimic the traffic characteristics of a Skype call.

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

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

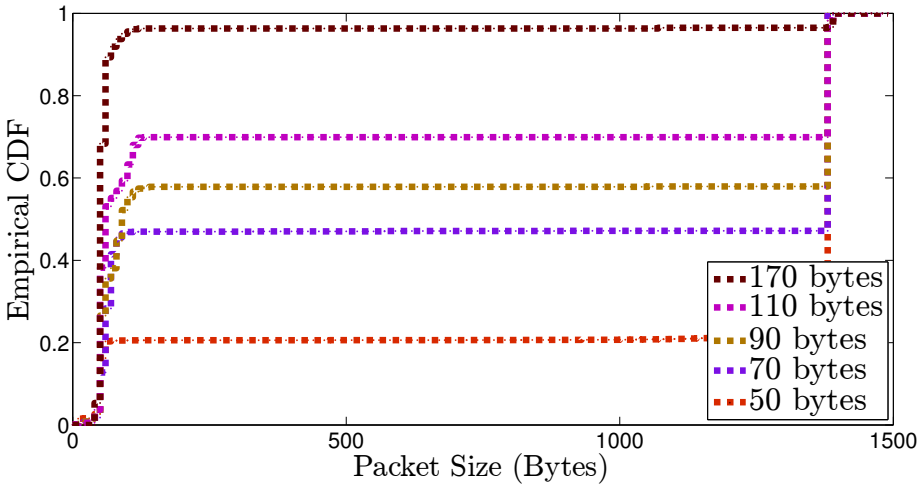


(a) Second-order statistics (size - delay)

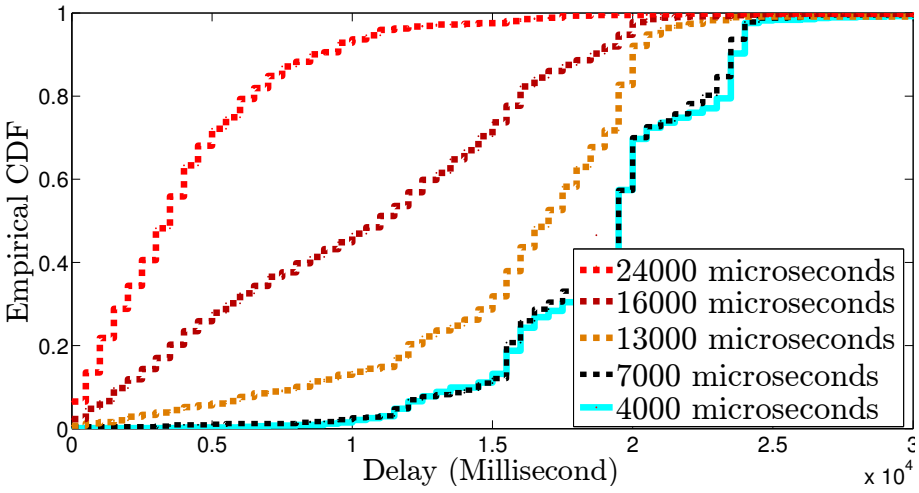


(b) Second-order statistics (delay - size)

Figure 4.2: Second-order statistics of a Skype video call.



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



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

Figure 4.3: Third-order statistics of a Skype video call for a fixed value of delay (1000 microseconds) are shown in Figure 4.3a and for a fixed packet size (60 bytes) are shown in Figure 4.3b.

to generate the statistics. However it is still unclear whether the encoder can be easily controlled via APIs. Thus, this is left as future work.

Chapter 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 the 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 [Skyd] API that has a separate runtime executable (which can be purchased online from Skype for less than US$10) 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. Our implementation supports both methods of communicating with Skype. The basic setup is discussed next and details of our implementation will appear in Chapter 6.

5.1 Setup

- **Step 1**: The bridge, which we denote by S, selects a UDP port number P_S at random and uses the Skype APIs to log in to the Skype network with a predefined set of credentials.[1] 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 [Tor] 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 in Skype 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 the 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.

- **Step 9**: When the bridge notices the call is dropped it listens for incoming Skype-Morph messages on port P_S.

- **Step 10**: Afterwards, the client uses the shared key and the UDP port obtained in previous steps to send data. The bridge listens for other incoming connections.

[1] 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.

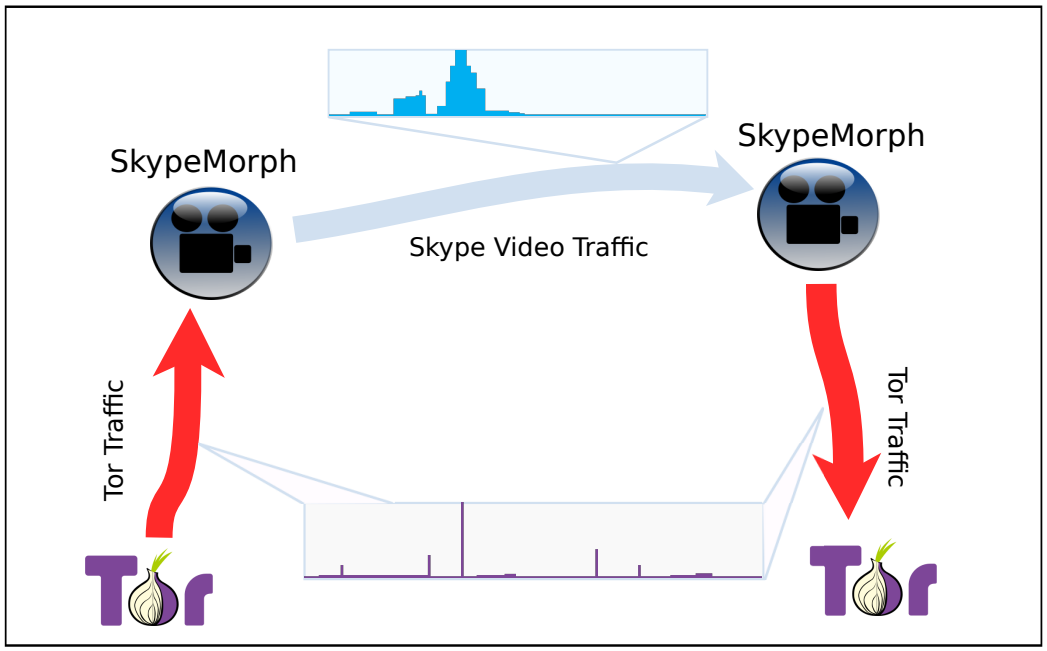


Figure 5.1: High-level overview of the SkypeMorph Architecture. The histograms show the distribution of packet sizes in Tor (at the bottom) and Skype video (at the top).

Note that having the bridge switch to a different UDP port for the next client connection should not arouse suspicion since, as discussed in Chapter 3, this is how normal Skype calls to multiple users behind NAT would appear to the censor. However, we will describe in Chapter 6 how we can make our technique more stealth by changing firewall rules, which allows us to avoid the port switching described above on the server side.

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, we can 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; although this method only supports first-order statistics, and only of packet sizes, not inter-arrival times, we extend it to incorporate timing (first-order arrival times) as well.

The overview of the SkypeMorph architecture is shown in Figure 5.1. 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.

Chapter 6

Implementation

In this chapter we describe our prototype implementation of SkypeMorph on Linux, which is implemented in C and C++ with the boost libraries [Riv07]. The prototype is built into two executable files called `smclient` and `smserver`, which realize the client C and the bridge S of the previous chapter, 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 chapter, both the client and the bridge log into Skype using the Skype API, exchange public keys, and then start their Skype video conversation. To use the Skype network, as already mentioned, one can either purchase a SkypeKit SDK and runtime, or use a running instance of the Skype application.

The SkypeKit runtime acts as a TCP server, which responds to proper requests (sent as binary streams), and conducts the operations requested, including logging in, sending messages, making calls, etc. The fact that the runtime can run as a command-line application and can be accessed over TCP makes it a suitable solution for the bridge side, where a single SkypeKit package, purchased for a very low price, can serve several users. On the other hand, the Skype Public API allows communicating with any running Skype application. On Linux, DBus is used to deliver text-formatted commands to the Skype application and on Windows this is done through the Window Messaging system. This makes it extremely easy for clients to use their normal Skype application to log in and make calls.

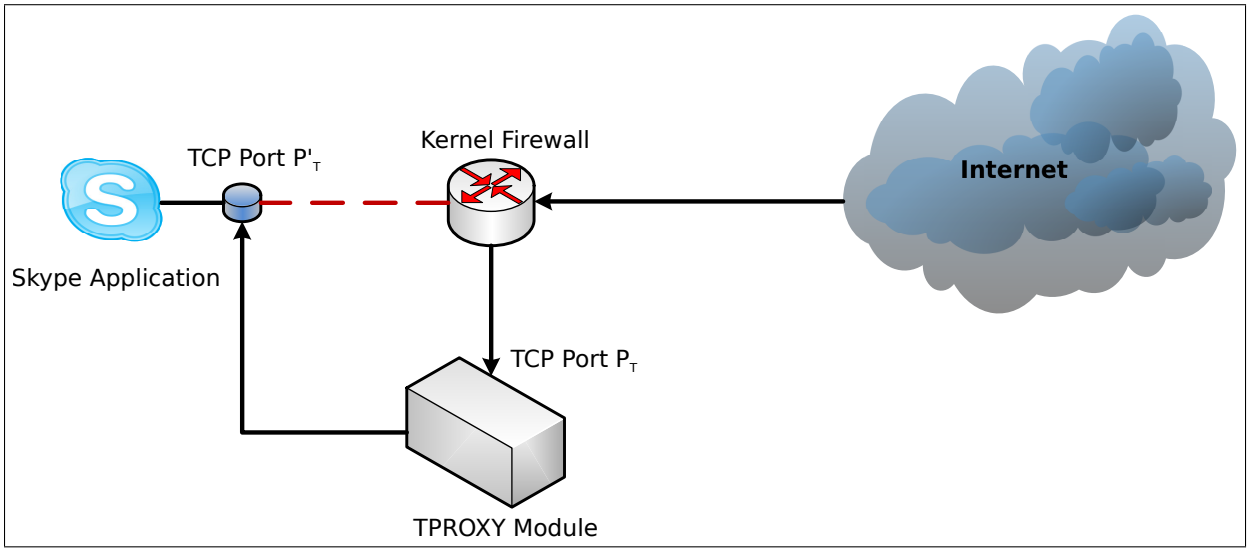
As we mentioned in Section 4.1, there are various TCP connections accompanying the Skype video conversation, which stay active even after the conversation is finished. In order to retain these TCP connections, our prototype implementation performs the following tricks:

- 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. `iptables` rules for redirecting traffic are added prior to the execution of our traffic shaping module.

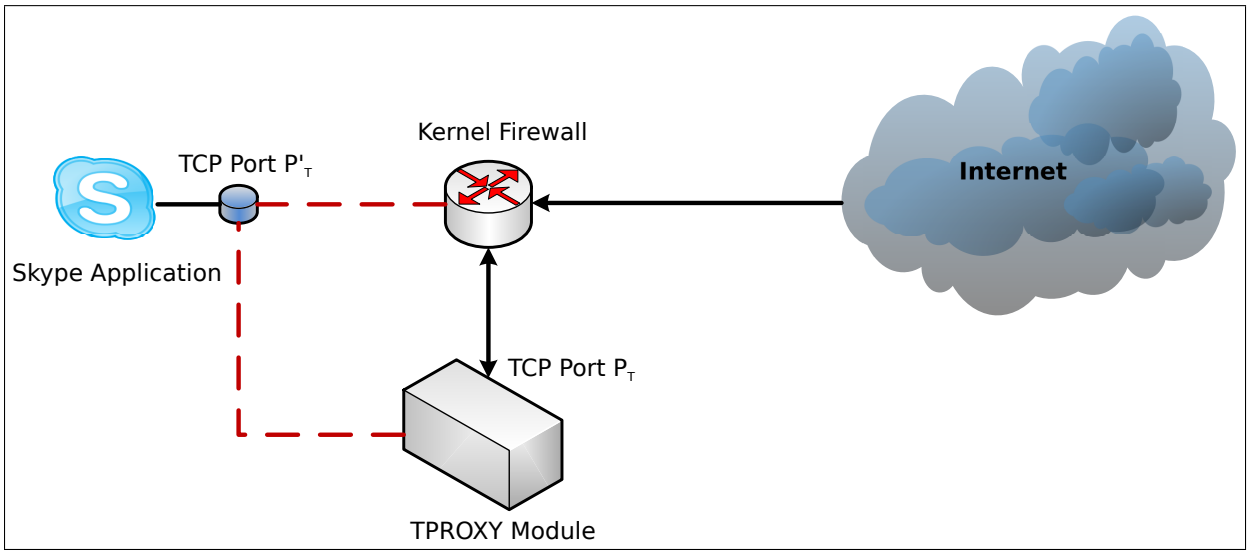
  Transparent proxying allows us to keep the TCP connection between the bridge and the client alive after the call is dropped. As shown in Figure 6.1, the TPROXY module listens on an arbitrary TCP port P_T and all the traffic to Skype TCP port P'_T is redirected to P_T. However, before and while a call is being placed, packets are relayed to P'_T by our software. As soon as the call is interrupted, our software starts producing TCP packets as though a Skype video call is in progress; i.e., we match the patterns produced by a call in our TCP output.

- As a consequence of the need for retaining the Skype connections to the Skype network for indistinguishablity, the Skype runtime has to stay active during the SkypeMorph session. Hence the UDP port P_C or P_S, described in Section 5.1, will still be assigned to the Skype runtime when SkypeMorph starts to tunnel Tor traffic; therefore, similar to TCP connections to the bridge described above, SkypeMorph has to operate on another UDP port. However, we do not need TPROXY, as UDP is connectionless and can be redirected more easily.

  For UDP we use the following procedure to redirect packets after the call is dropped: the Skype runtime on the client operates on UDP port P_C. When `smclient` starts to tunnel Tor traffic, it binds to a port P'_C. Then, an `iptables` rule R_C is created with an SNAT target to alter the source port of all Skype UDP packets from P'_C to 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 SkypeMorph application. This prevents the censor from noticing port changes between the genuine Skype video call traffic and the SkypeMorph traffic, as all the packets sent (either before or after the call is dropped) on the wire have P_S and P_C as their UDP ports (Figure 6.2 has a schematic view of this process).

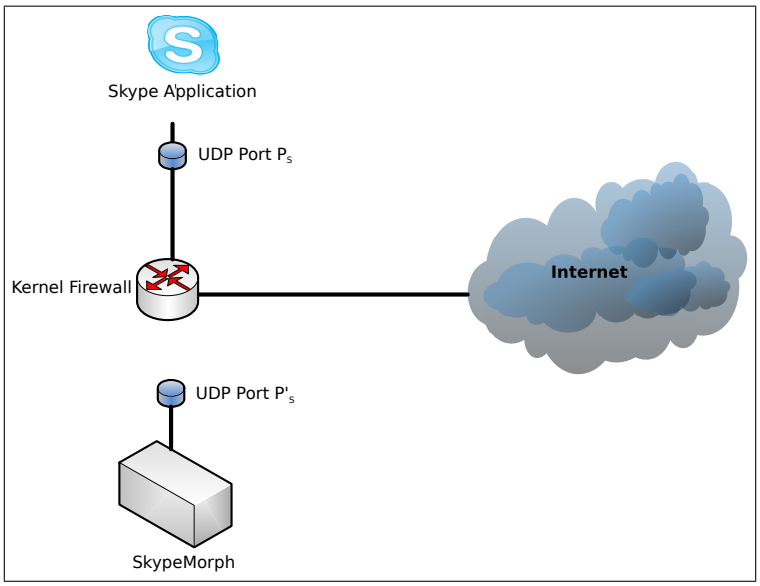


(a) TPROXY Relaying

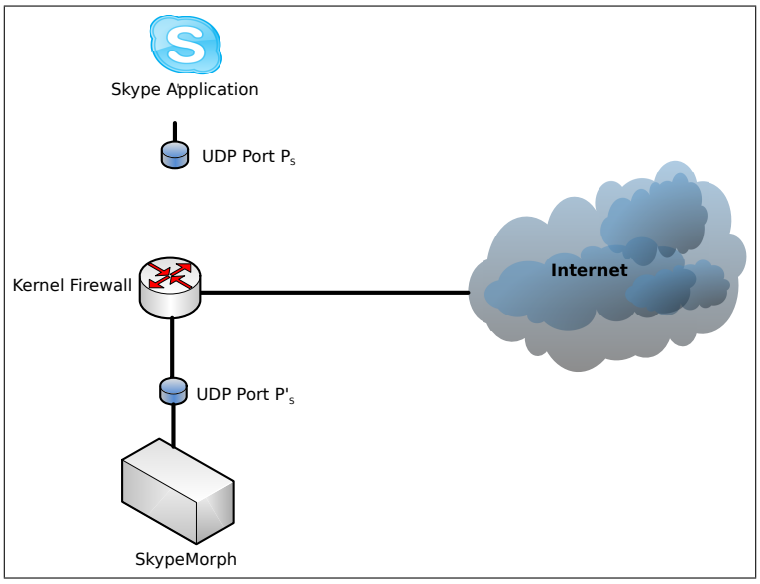


(b) TPROXY Reproducing Traffic

Figure 6.1: The TPROXY module starts by forwarding traffic to the Skype TCP port before and during the call initiation process (a); after the call is dropped, it reproduces the TCP packet patterns (b).



(a) UDP Forwarding



(b) UDP Redirected

Figure 6.2: UDP packets arrive at the Skype UDP port before and during the call initiation process (a); after the call is dropped, UDP packets are produced by SkypeMorph and UDP flows are redirected to its port (b).

It is worth mentioning that on recent Linux kernels, the `conntrack` tool, a part of the Linux stateful firewall system, assigns a connection (which is basically defined as a four-tuple of source/destination IP addresses and ports) to each UDP packet. The firewall adds an entry to the list of active UDP connections once a packet is received and considers that connection as "established" once a response is seen. Each entry in the table has a timeout period during which the connection is considered active. `conntrack` rules override `iptables` rules for existing connections and would not allow redirection of previously seen tuples. Therefore, before applying `iptables` rules one needs to make sure to remove its corresponding `conntrack` rules.

For the cryptographic features, we use the `curve25519-donna` [Lan08] library to generate elliptic-curve Diffie-Hellman keys shared between `smclient` and `smserver`. Curve25519 is chosen because of its short secret keys and its high performance [Ber06]. Our current prototype implementation sends public keys in plaintext over Skype and relies on Skype for confidentiality and authenticity. However the small sizes of messages in this phase make it possible to use any standard cryptographic or steganographic methods efficiently, in case other properties are needed. After this point, 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 [Cya], 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 6.3.

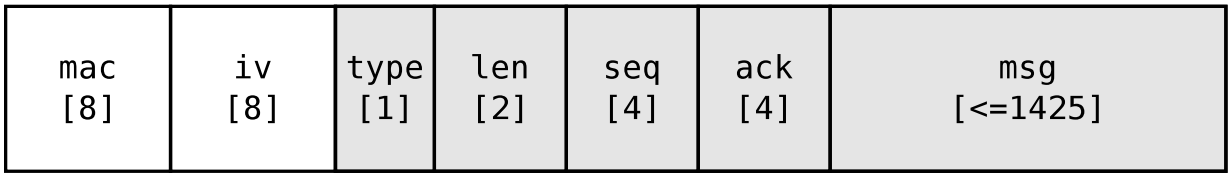


Figure 6.3: SkypeMorph UDP packet body layout, where the size (in bytes) is under the name for each field. The shaded parts are encrypted using 256-bit AES counter mode. All bytes after the mac field are included in the HMAC-SHA256-64 computation.

Traffic Shaping Oracle

We next discuss the traffic shaping oracle component, which controls the sizes and timings of each successive UDP packet to be sent. The oracle is the component that has access to the packet size and inter-packet delay distributions and its task is to sample from these distributions and output the result to our packetizer (described later in this section), to guarantee that the SkypeMorph traffic matches these distributions. 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 Section 4.4.

We currently have gathered data for up to $n = 3$ for a fixed bandwidth (see Section 7.1 for details on how this data is gathered), 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 probability distribution of packet sizes for both the source and the target traffic is gathered; for instance if the source is web browsing over Tor, we can get an empirical distribution from the previous web surfing instances. Having these distributions, we compute the morphing matrix. Next, the oracle first reads distributions of the packet sizes of the Tor traffic, the inter-packet delays of the Skype traffic, and the pre-computed morphing matrix. We use the `morpher` library from the Morpher Project [Kad11] 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.[1] 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 6.4. 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.

[1] Note that this might cause the Traffic Morphing method to produce dummy packets.

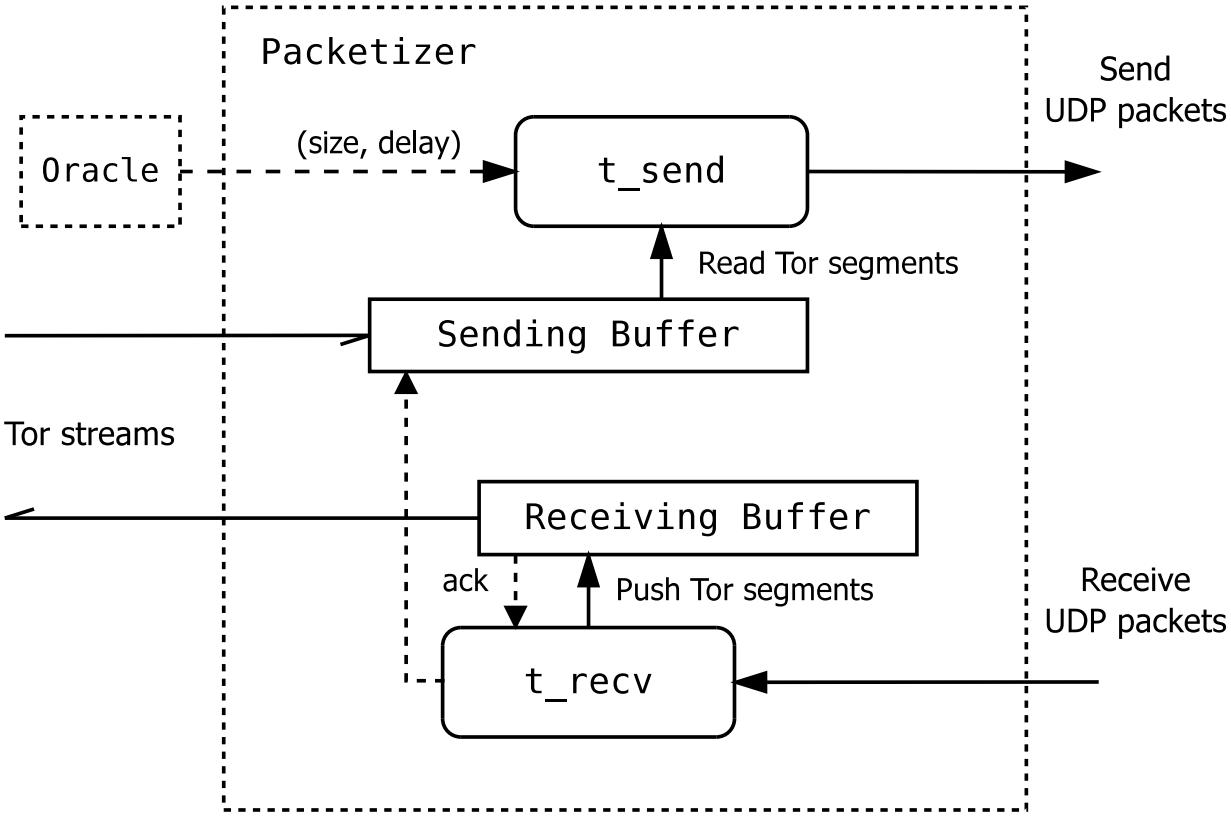


Figure 6.4: Structure of the packetizer.

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. Note that we might have to send

dummy packets — those that are entirely padding. 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, $\text{seq}_r + 1$ 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.

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 relevant 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.

6.3 OpenWrt Implementation and other Operating Systems

Changing `iptables` rules as described in Chapter 6.1 requires root privileges on both the client and the bridge. Although it is possible to decouple the portion of the code running `iptables` as a separate runnable with elevated access, for security purposes and ease of use, running as a normal user is more desirable. To provide this, we chose to implement a separate piece of software, for home routers running the Linux operating system, that preforms `iptables` manipulation in a way that is transparent to SkypeMorph users.

For the purpose of testing we used the OpenWrt [Ope] software. OpenWrt is an open-source Linux distribution with package management for embedded systems designed specif-

ically to allow customizable firmwares for router devices. The OpenWrt distribution comes with a list of pre-configured packages including `iptables` and `conntrack`. It also contains an SDK for cross-compiling to routers with different architectures.

Our OpenWrt implementation comes in two pieces: `iptbl_server`, which runs as a daemon on the router and acts as a server, responding to requests to add or remove `iptables` rules by nodes in the internal network and `iptbl_client`, which sends an `iptables` rule request to the router. During a SkypeMorph initial step, once the call is being dropped, a request is sent from the client to the server, containing information regarding the port redirection and the destination. The server receives this information, adds (or deletes depending on whether a new connection is being established or a connection is being torn down) the corresponding `iptables` rules and removes all the previous connections from the client to the destination on the same port in the `conntrack` tables. After all this is done, it notifies the client that it can start sending UDP packets. This way, the client does not need to have root privileges and need only create a TCP connection to `iptbl_server` module on the router, which can be done as a normal user.

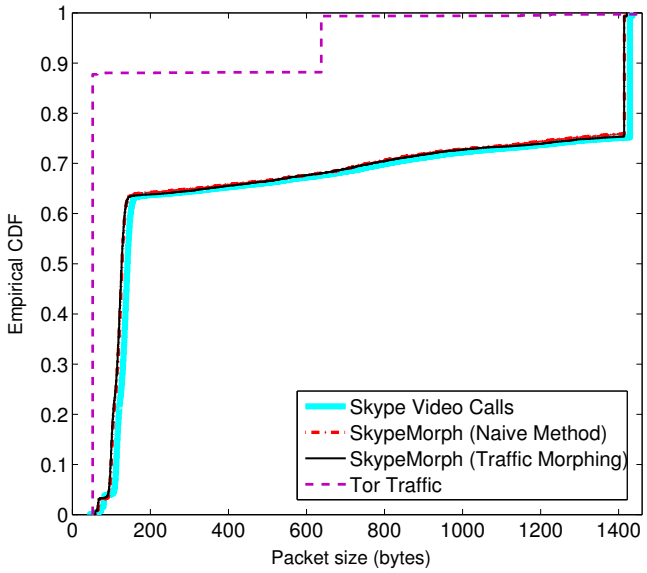
Chapter 7

Experiments

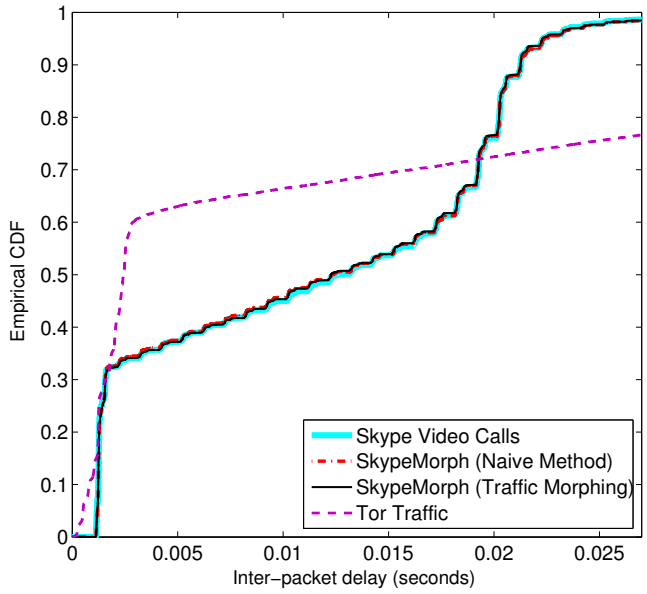
7.1 Linux Experiments

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. In particular, we have presented in this section our results for a call from a Linux station to a remote host connected to a wireless network for a duration of six minutes and resulting average bandwidth of around 45 KB/s. 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 as input to the SkypeMorph traffic-shaping oracle for drawing samples and generating the morphing matrix. The Morpher Project [Kad11] comes with a packet size distribution for a typical web browsing over Tor, which was used as the source distribution to produce the morphing matrix. Next, we used the distributions we gathered in the previous step to set up a SkypeMorph bridge on the same local network as the Tor client. We used this bridge as a proxy for browsing various webpages and downloading different files for five minutes.

Figure 7.1 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 [NIS03] for both the packet sizes and inter-packet delays shows no statistically



(a) Packet size distribution



(b) Inter-packet delays distribution

Figure 7.1: Experimental cumulative distributions for Skype video, SkypeMorph, and Tor. We show packet size and inter-packet delay distributions in (a) and (b) respectively.

Table 7.1: 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	$12\% \pm 1\%$	$28\% \pm 2\%$	$28\% \pm 3\%$

significant difference between the Skype video and SkypeMorph distributions, using either the naïve traffic-shaping or the enhanced Traffic Morphing methods and the same should apply to higher-order statistics. 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 [WCM09] 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[1] by connecting to the same bridge described above and running with the same setup. Using this 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 7.1.

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

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

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; therefore, combining Traffic Morphing with timing techniques does not have any significant effect in reducing the overhead.

The Traffic Morphing method requires the source distribution to remain static or else the morphing matrix needs to be recalculated (which takes more than 5 minutes on a commodity computer) in order to output the correct matrix, and also extending the morphing matrix to include higher order statistics does not seem to be feasible at this point. Therefore, we conclude that using the naïve method is more efficient and effective.

We also ran a Kolmogorov-Smirnov test on the output of both shaping methods and it reported 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.

7.2 OpenWrt Experiment

In this section we describe the method and the results of our experiment on our OpenWrt implementation. The test setting is shown in Figure 7.2 and is as follows: A laptop running the SkypeMorph client is connected to a SkypeMorph bridge (located at a remote location on different network) over a wireless network to a Buffalo WZR-HP-G300NH2 access point/router. The router runs an OpenWrt Linux system with kernel 3.3.8 and is connected to the Internet through a hub. It is also running the `iptbl_server` module described in Section 6.3. The hub is used to forward the packets while allowing the output of the router to be dumped to another terminal, connected to the same hub. Using this terminal, we gathered packets destined to SkypeMorph bridge and analyzed them. The experiment was repeated 10 times and Figure 7.3 shows the result of the experiments. The packet size distributions output by SkypeMorph and Skype (shown in Figure 7.3a) are almost identical; similarly, the packet timing distributions output by SkypeMorph and Skype (shown in Figure 7.3b) also match very closely.

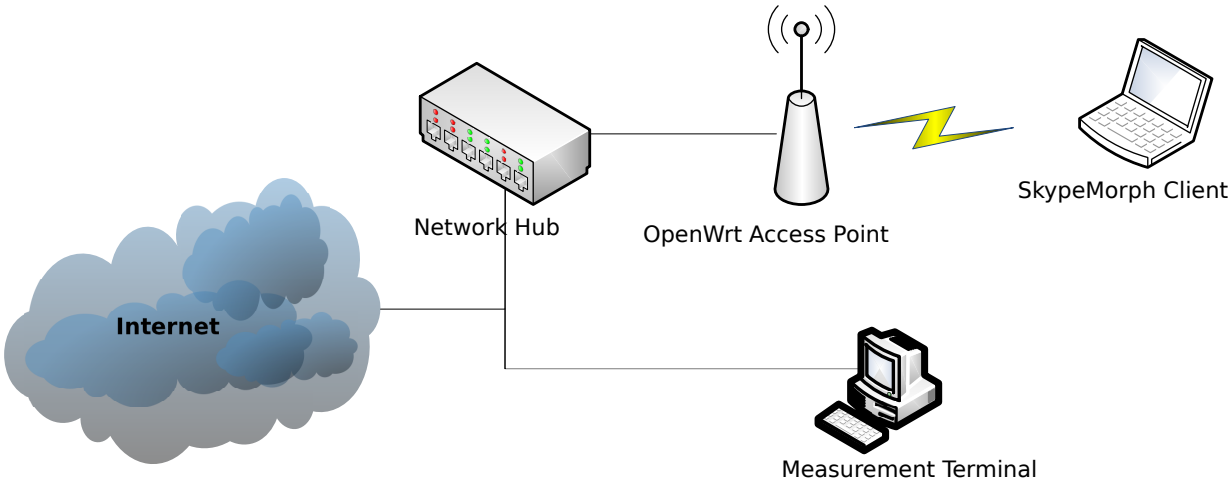
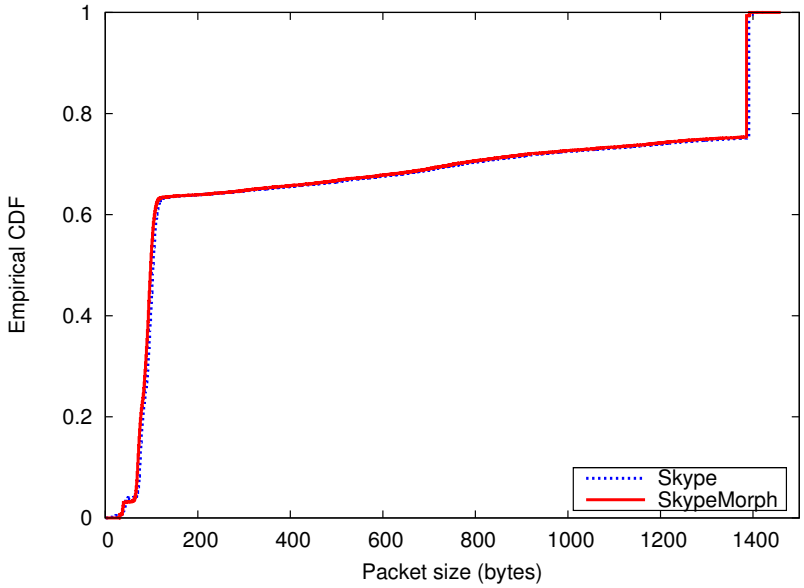
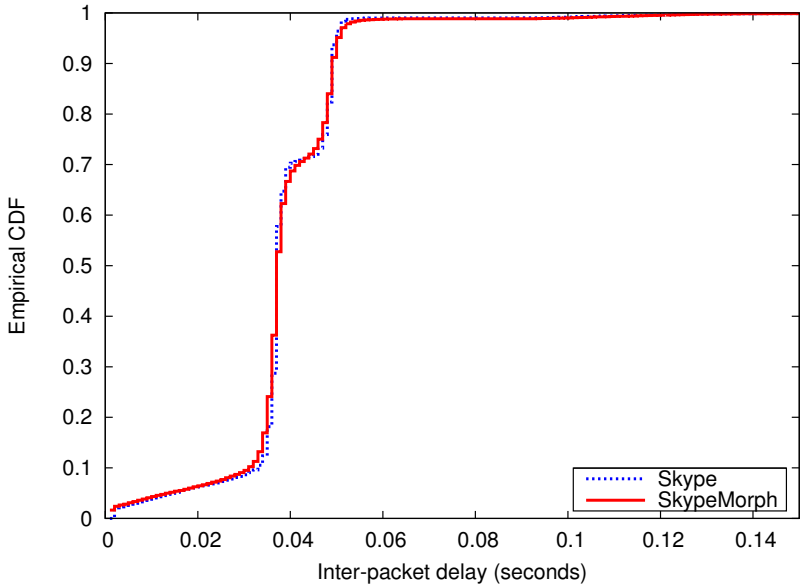


Figure 7.2: Our network setup for experimental run of SkypeMorph on OpenWrt. A node is connected through wireless to the OpenWrt router and a hub is used to intercept the outgoing packets to create a trace.



(a) Packet Size Distribution of OpenWrt Implementation



(b) Packet Timing Distribution of OpenWrt Implementation

Figure 7.3: Packet size and inter-packet delay distributions as produced by a home router running our SkypeMorph software on OpenWrt.

Chapter 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 in Chapter 3. 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. Also as discussed in our threat model, by having far more SkypeMorph bridges than those run by the censor, we can ameliorate the results of such an attack.

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.

More specifically, to replace Skype with a target protocol T, the following steps are required. First the session keys need to be exchanged either through an out-of-band mechanism or the target protocol itself similar to setup phase described in Section 5.1. Second, if naïve shaping is used, the Skype distribution for packet sizes and delays (or their joint distributions up to the desired order statistic), should be replaced by those of T. Otherwise, as stated above, the recalculated morphing matrix for T should be fed into the traffic shaping oracle as discussed in Section 6.2.

Higher-order Statistics and Skype Call Traces.

As discussed in Section 4.4, an interesting avenue for future work would be 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.

There is also another viable direction for outputting different distributions, while matching Skype video calls characteristics. The approach is to gradually construct a database of packet size and timings produced by Skype calls each time a user makes a call over Skype. A tool can gather this information and sort them by their bandwidth rate, and once a SkypeMorph connection is required, we can iterate through this database and use the portion that matches our desired rate.

SkypeMorph and Traffic Classification. A major task for protocol obfuscation is to be able to withstand different detection mechanisms. In our case, statistical analysis, including machine learning algorithms, seem proper avenues to explore. For instance, Wiley [Wil11a] discusses how one can employ Bayesian classification on features such as packet sizes and direction of a flow to predict the type of an obfuscated flow. Thus, testing our method against similar analysis can reveal valuable information about the ways we can improve our techniques.

8.1 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 [WCM09] 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.

References

[AM10] Jacob Appelbaum and Nick Mathewson. Pluggable Transports for Circumvention. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/180-pluggable-transport.txt`, 2010. [Online; accessed January 2013].

[BD06] Philippe Biondi and Fabrice 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 January 2013].

[Ber05] Tom Berson. Skype Security Evaluation. Technical Report ALR-2005-031, Anagram Laboratories, 2005.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Proceedings of the 9th International Conference on the Theory and Practice of Public Key Cryptography*, pages 207–228. 2006.

[BFV10] Sam Burnett, Nick Feamster, and Santosh Vempala. Chipping Away at Censorship Firewalls with User-generated Content. In *Proceedings of the 19th USENIX Security Symposium*, pages 29–29, 2010.

[BLJL06] George Bissias, Marc Liberatore, David Jensen, and Brian Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proceedings of the 6th Workshop on Privacy Enhancing Technologies*, pages 1–11, 2006.

[BMM+07] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing Skype Traffic: When Randomness Plays with You. In *Proceedings of the 2007 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 37–48, 2007.

[BMMdT08] Dario Bonfiglio, Marco Mellia, Michela Meo, and Nicolò Ritacca Dario Rossi Politecnico di Torino. Tracking Down Skype Traffic. In *Proceedings of the 27th IEEE Conference on Computer Communications*, pages 261–265, 2008.

[BS06] S. A. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–11, 2006.

[CM01] R. Chandramouli and N. Memon. Analysis of LSB Based Image Steganography Techniques. In *Proceedings of the International Conference on Image Processing*, volume 3, pages 1019–1022, 2001.

[Cya] CyaSSL Project. Embedded SSL Library for Applications, Devices, and the Cloud. `http://www.yassl.com/yaSSL/Home.html`. [Online; accessed January 2013].

[DCGS09] 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.

[Din11a] Roger Dingledine. Research Problems: Ten Ways to Discover Tor Bridges. `https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges`, 2011. [Online; accessed January 2013].

[Din11b] Roger Dingledine. Tor and Circumvention: Lessons Learned, 2011. [Online; accessed January 2013].

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[EG12] Tariq Elahi and Ian Goldberg. CORDON–A Taxonomy of Internet Censorship Resistance Strategies. Technical Report CACR 2012-33, University of Waterloo, 2012.

[EGKP11] Miro Enev, Sidhant Gupta, Tadayoshi Kohno, and Shwetak N. Patel. Televisions, Video Privacy, and Powerline Electromagnetic Interference. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 537–550, 2011.

[FBH+02] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing Web Censorship and Surveillance. In *Proceedings of the 11th USENIX Security Symposium*, pages 247–262, 2002.

[FHE+12] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phil Porras. Evading Censorship with Browser-based Proxies. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium*, pages 239–258, 2012.

[HM05] Stefan Hetzl and Petra Mutzel. A Graph-Theoretic Approach to Steganography. In *Proceedings of the 9th IFIP TC-6 TC-11 International Conference on Communications and Multimedia Security*, pages 119–128, 2005.

[HNCB11] Amir Houmansadr, Giang T.K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 187–200, 2011.

[HvAL09] N. Hopper, L. von Ahn, and J. Langford. Provably Secure Steganography. *IEEE Transactions on Computers*, 58(5):662 –676, 2009.

[JDJ00] Neil F. Johnson, Zoran Duric, and Sushil Jajodia. *Information Hiding: Steganography and Watermarking—Attacks and Countermeasures*. Kluwer Academic Publishers, 2000.

[Kad11] George Kadianakis. Morpher. `https://gitorious.org/morpher/morpher`, 2011.

[KEJ+11] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*, August 2011.

[KM11] George Kadianakis and Nick Mathewson. A Simple Obfuscating Proxy. `https://gitweb.torproject.org/obfsproxy.git`, 2011. [Online; accessed January 2013].

[Krz03] Martin Krzywinski. Port knocking: Network Authentication Across Closed Ports. *SysAdmin Magazine*, 12(6):12–17, 2003.

[Lan08] Adam Langley. curve25519-donna. http://code.google.com/p/curve25519-donna/, 2008. [Online; accessed January 2013].

[Lar12] Frederic Lardinois. 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].

[LCPW12] Christopher S. Leberknight, Mung Chiang, Harold Vincent Poor, and Felix Wong. A Taxonomy of Internet Censorship and Anti-Censorship. http://www.princeton.edu/~chiangm/anticensorship.pdf, 2012. [Online; accessed January 2013].

[LL06] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 255–263, 2006.

[MH09] Jon McLachlan and Nicholas Hopper. On the Risks of Serving Whenever You Surf: Vulnerabilities in Tor's Blocking Resistance Design. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, pages 31–40, 2009.

[MSS11] Wojciech Mazurczyk, Pawel Szaga, and Krzystof Szczypiorski. Using Transcoding for Hidden Communication in IP Telephony. *ArXiv e-prints*, November 2011.

[Mur11] Steven J. Murdoch. Moving Tor to a Datagram Transport. https://blog.torproject.org/blog/moving-tor-datagram-transport, 2011. [Online; accessed January 2013].

[Nan12] Nancy Messieh. 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].

[NIS03] NIST/SEMATECH. e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook/index.htm, 2003. [Online; accessed January 2013].

[Ope] OpenWrt Project. OpenWrt. https://openwrt.org/. [Online; accessed January 2013].

[PAK99] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information Hiding-A Survey. *Proceedings of the IEEE, Special Issue on Protection of Multimedia Content*, pages 1062 –1078, 1999.

[Pro01] Niels Provos. Defending Against Statistical Steganalysis. In *Proceedings of the 10th USENIX Security Symposium*, pages 323–336, 2001.

[Riv07] Rene Rivera. Boost C++ Libraries. `http://www.boost.org/`, 2007. [Online; accessed January 2013].

[SFKT06] Kyoungwon Suh, Daniel R. Figueiredo, Jim Kurose, and Don Towsley. Characterizing and Detecting Skype-Relayed Traffic. In *Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–12, 2006.

[Sim84] Gustavus J. Simmons. The Prisoners Problem and the Subliminal Channel. In *Advances in Cryptology: CRYPTO 1983*, pages 51–67, 1984.

[SJP+11] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. BridgeSPA: Improving Tor bridges with Single Packet Authorization. In *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society*, pages 93–102, 2011.

[Skya] Does Skype use encryption? `https://support.skype.com/en/faq/FA31/Does-Skype-use-encryption`. [Online; accessed January 2013].

[Skyb] Number of Skype users in 2010. `http://about.skype.com/press/2011/05/microsoft_to_acquire_skype.html`. [Online; accessed January 2013].

[Skyc] Skype Software. `http://skype.com`. [Online; accessed January 2013].

[Skyd] SkypeKit. `http://developer.skype.com/public/skypekit`. [Online; accessed January 2013].

[Skye] TOM Online: Skype Services for the Republic of China. `http://skype.tom.com/`. [Online; accessed January 2013].

[SLH+07] T. Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices That Tell On You: Privacy Trends in Consumer Ubiquitous Computing. In *Proceedings of 16th USENIX Security Symposium*, pages 5:1–5:16, 2007.

[Tor] The Tor Project. Tor BridgeDB. `https://gitweb.torproject.org/bridgedb.git/tree`. [Online; accessed January 2013].

[Tor12] The Tor Project. Tor Metrics Portal: Users. `https://metrics.torproject.org/users.html`, 2012. [Online, accessed July 2012].

[WBC+08] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, pages 35–49, 2008.

[WBMM07] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of the 16th USENIX Security Symposium*, pages 4:1–4:12, 2007.

[WCM09] Charles Wright, Scott Coull, and Fabian Monrose. Traffic Morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium-NDSS 09*, 2009.

[Wil11a] Brandon Wiley. Blocking-Resistant Protocol Classification Using Bayesian Model Selection. `http://http://blanu.net/BayesianClassification.pdf`, 2011. [Online; accessed January 2013].

[Wil11b] Brandon Wiley. Dust: A Blocking-Resistant Internet Transport Protocol. `http://blanu.net/Dust.pdf`, 2011. [Online; accessed January 2013].

[Wil12] Tim Wilde. Knock Knock Knockin' on Bridges' Doors. `https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors`, January 2012. [Online; accessed January 2013].

[WL12] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *Proceedings of the 2nd USENIX Workshop on Free and Open Communications on the Internet*, 2012.

[WMSM11] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, pages 3–18, 2011.

[WP00] Andreas Westfeld and Andreas Pfitzmann. Attacks on Steganographic Systems. In *Proceedings of the Third International Workshop on Information Hiding*, pages 61–76, 2000.

[WWGH11] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the Network Infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, pages 459–474, 2011.

[WWY+12] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a Camouflage Proxy for the Tor Anonymity System. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 109–120, 2012.