Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention

John Geddes
University of Minnesota
Minneapolis, MN 55404
geddes@cs.umn.edu

Max Schuchard
University of Minnesota
Minneapolis, MN 55404
schuch@cs.umn.edu

Nicholas Hopper
University of Minnesota
Minneapolis, MN 55404
hopper@cs.umn.edu

ABSTRACT

In response to increasingly sophisticated methods of blocking access to censorship circumvention schemes such as Tor, recently proposed systems such as Skypemorph, FreeWave, and CensorSpoofer have used voice and video conferencing protocols as "cover channels" to hide proxy connections. We demonstrate that even with perfect emulation of the cover channel, these systems can be vulnerable to attacks that detect or disrupt the covert communications while having no effect on legitimate cover traffic. Our attacks stem from differences in the channel requirements for the cover protocols, which are peer-to-peer and loss tolerant, and the covert traffic, which is client-proxy and loss intolerant. These differences represent significant limitations and suggest that such protocols are a poor choice of cover channel for general censorship circumvention schemes.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and Protection

General Terms

Security

Keywords

Censorship;Anonymity;CensorSpoofer;FreeWave;SkypeMorph

1. INTRODUCTION

Internet censorship has become a widespread issue as ISPs, often under control of repressive governments, filter their users' access to the Internet. In response, users have employed proxy-based circumvention schemes, where clients connect to proxies (or chains of proxies) that relay requests to the rest of the Internet, hiding the clients' true destinations. Censors have responded by employing increasingly sophisticated techniques, including protocol fingerprinting, deep packet inspection, and active probing, to identify and block access to circumvention systems as well.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS'13, November 4–8, 2013, Berlin, Germany.
Copyright 2013 ACM 978-1-4503-2477-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2508859.2516742.

Recently, several circumvention systems have emerged that attempt to use popular communication systems already in place as a *cover-protocol* for anonymous communication. This allows for encrypted communication under the guise of legitimate traffic, making detection, enumeration, and blocking difficult for potential censors. In this paper we examine three such systems with different approaches. SkypeMorph [18] mimics Skype communication in an effort to hide connections to Tor bridges, FreeWave [11] uses a modem to tunnel IP traffic through a voice over IP (VoIP) session, and CensorSpoofer [22] uses IP spoofing and asynchronous communication while mimicking VoIP.

Recently, Houmansadr, Brubaker and Shmatikov [9] have shown that accurately mimicking a modern cover protocol can be challenging, listing several methods a censor could use to distinguish CensorSpoofer and SkypeMorph traffic from legitimate traffic. It is not hard to imagine improved versions of these systems that interact directly with the cover-protocol system, replacing only encrypted contents with covert data, or using an approach similar to FreeWave.

In this paper we demonstrate that even in this case, inherent problems can arise when using covert channels for censorship circumvention. These issues arise due to mismatches between the cover protocol and the proxy protocol, and can violate both the *unobservability* and *unblockability* principles of anonymity systems, crucial for censorship-resistant communication. We highlight three types of mismatches:

- *Architectural mismatches* occur when the cover protocol and the circumvention scheme have different communication architectures, as when a peer-to-peer VoIP protocol is used as a cover for a client-proxy architecture. These mismatches can allow easy identification of proxies and connections, leading to dynamic blocking rules. An example is the FreeWave system, which uses Skype supernodes as relays in an attempt to hide the location of proxies; however, Skype clients use supernodes only as a last resort, *after* attempting direct contact with the destination of a call; thus attempting to contact a FreeWave proxy node will reveal the IP address. We document similar problems with SkypeMorph and CensorSpoofer in section 5.

- *Channel mismatches* occur when the cover protocol and the circumvention scheme have different requirements on the reliability of transmissions. For example, many VoIP and streaming video protocols use UDP and are designed to tolerate packet losses and duplication, whereas proxied TCP connections require reliable transmission. This can lead to attacks that allow a censor to drop or duplicate traffic at a level that is tolerable to the legitimate cover protocol but renders the covert channel useless. As an example, we show

how an adversary can indefinitely stall a Skypemorph transfer while having no effect on the quality of voice calls by dropping a handful of packets and delivering duplicate packets at a 5% rate, through *targeted dropping* of SkypeMorph acknowledgement (ACK) packets. Similar problems in CensorSpoofer and FreeWave are documented in section 4.

- *Content Mismatches* occur when significant differences between the content embedded in a covert channel and the expected contents cause noticeable differences in traffic patterns, even though the cover protocol is perfectly emulated. For example, in section 6 we show that the audio signal of a modem is sufficiently divergent from the audio signals of human speech that a VoIP protocol (such as Skype) using variable bit rate (VBR) encoding will generate easily distinguishable distributions on packet lengths for the two cases: FreeWave over Skype generates length sequences with a dramatically lower variance than spoken language.

The rest of the paper is organized as follows. In Section 2 we review previous anonymity systems and the issues surrounding them, followed by a more complete discussion of SkypeMorph, FreeWave, and CensorSpoofer. Section 3 covers our experimental setup for evaluating these schemes. In Section 4 we cover issues that all three systems have in dealing with using an error tolerant channel for reliable transmission. Following that Section 5 discusses the pitfalls of using a peer to peer cover-protocol system for anonymous communication that is inherently built around a client-proxy model. Section 6 focuses on FreeWave and problems that the system has with cross-content delivery of IP traffic over VoIP. Finally we wrap up with discussion and conclusions in Section 7.

2. BACKGROUND AND RELATED WORK

Censorship circumvention systems attempt to provide users unrestricted access to the Internet while staying hidden from a censoring ISP. The main goals of these censorship-resilient systems is to provide *unobservability*, whereby a censor is unable to tell whether or not a client is participating in the system, and *unblockability* meaning a censor cannot block access to the system without also blocking access to the entire Internet or a popular service. One common approach is to use encrypted tunnels through one or more proxies in an attempt to hide the actual destination from censors, as exemplified by Tor [8], Anonymizer [3], and JAP [1]. These systems are vulnerable to attacks that enumerate and block the proxy nodes, preventing users from accessing the system. To combat this Tor introduced private bridge relays [7] whose information is unlisted, and tightly access-controlled. Even under this restricted access, it has been shown that bridges are vulnerable to enumeration [17], and while methods were developed to prevent these techniques [21], there is documented evidence of nations such as China and Iran performing deep packet inspection on outgoing TLS connections and successfully identifying Tor bridges [2].

Another class of censorship circumvention systems is decoy routing [12] schemes, such as Telex [27] and Cirripede [10]. Instead of using proxies in an end-to-end manner, these systems use an end-to-middle approach using decoy routers along the path of routers forwarding packets to an unblocked end host. By using steganographic cryptography, a client can notify one of the intermediate decoy routers to deflect a TLS connection to a covert destination, providing an encrypted connection to the covert host unbeknownst to the censor. In principle, these schemes make enumeration more difficult, since censors that identify a proxied connection can only identify *paths* that contain decoy routers, and censors do not directly control the path a connection travels once it exits the censored ISP. However, as shown in [20], without extensive deployment, even small nations have enough resources to enumerate participating routers via path intersection, and are able to successfully route around and avoid decoy routers while cutting off a negligible amount of the overall Internet. Furthermore there exist several mechanisms that allow a censor to detect clients using these systems, and even in some circumstances identify the actual covert destination.

A third class of censorship-resilient systems attempt to use cover-protocols to hide anonymous communication. Using popular cover-protocols that censors would be hesitant to block gives these systems an avenue for embedding anonymous communication in apparently legitimate cover traffic. The remainder of this section will cover the details of the three systems, SkypeMorph, FreeWave, and CensorSpoofer, that are the focus of the remainder of this paper.

2.1 SkypeMorph

SkypeMorph [18] attempts to address the *unobservability* problem with Tor bridges, where large censors are able to utilize deep packet inspection to fingerprint TLS connections made to bridges [2]. In order to connect to a bridge running SkypeMorph, a client obtains the Skype ID of the bridge through some out of band channel. Once the identity is known, the SkypeMorph client uses the Skype infrastructure to locate proxies and conduct the session setup, utilizing chat functionality inside Skype to conduct key exchange and negotiate a unique port at which to contact the SkypeMorph bridge. Once this step has concluded, the client initiates a video call directly to the SkypeMorph node, which reveals the bridge's IP address to the user. The SkypeMorph node detects the incoming call, ignores it, and begins listening on the negotiated port for data traffic from the user. At this point the user ceases to use the actual Skype client, and simply attempts to mimic one by sending encrypted data through a traffic shaper to make packet sizes and sending times similar to traffic from a Skype video call.

2.2 FreeWave

FreeWave [11] is a standalone censorship circumvention system addressing both *unobservability* and *unblockability*. While similar in concept to SkypeMoprh, FreeWave takes the mimicry of VoIP calls a step further by actually sending IP traffic over voice using a virtual modem. In FreeWave, the user places a VoIP call to the proxy server using Skype, a popular VoIP client. FreeWave users call a publicly known Skype ID, relying on Skype's native encryption to hide which ID they are calling. Proxy nodes are configured to ignore direct incoming calls, forcing Skype to bridge the connection with the help of a super-node. From the censor's perspective, the user is either making a call to a random Skype node, or using the random Skype node to bridge a call. Blocking the proxies will not prevent the user from connecting to them, since they always connect via a random intermediary. Data sent to and from the proxy is modulated into sounds via a software modem, which in turn pipes those sounds to a virtual sound card, which transmits them to the proxy server via the Skype call.

2.3 CensorSpoofer

CensorSpoofer [22], proposed by Wang et al., attempts to avoid detection by mimicking VoIP calls placed over the Session Initiation Protocol (SIP) [19]. The key observation of CensorSpoofer is that traffic generated by web browsing is highly asymmetrical. A small amount of traffic, the HTTP request, is sent by the user and a large volume of traffic, the actual content, in turn flows from the server to the client. Thus in the CensorSpoofer architecture, the client uses a possibly low bandwidth channel, an email message for

example, to send a request to the proxy; the proxy then fetches the content on behalf of the user. Meanwhile the user initiates a VoIP call to a unique SIP ID which has been registered by the proxy for that user only.

SIP IDs include both a user and a domain. In order to locate the IP address which maps to a given SIP ID the user contacts the SIP server in charge of the given domain. The server in turn contacts the host running the given ID, which can respond with an IP address it would like to be contacted at. In CensorSpoofer the proxy does *not* respond with its own IP address, rather it selects a random host, referred to as the dummy host, on the Internet which `nmap` [16] does not show to be either off-line or have a closed SIP port. Since VoIP data and SIP messages are commonly sent via UDP, the proxy can spoof the IP address of the dummy host when sending subsequent SIP messages along with "call data" which will be delivered to the client. Since the user never discovers the IP address of the proxy, the censor cannot enumerate and block proxies by posing as a user. The proxy then sends the the content it fetched on behalf of the user by making it appear like encrypted VoIP data coming from the dummy user by spoofing IP and UDP headers accordingly. The client also sends junk data to the dummy host to complete the illusion of a VoIP call between the user and the dummy host. The dummy host ignores this junk data as it does not actually have a VoIP call between itself and the user.

2.4 Mimicry Issues

As described in [18, 22], SkypeMorph and CensorSpoofer only use the cover-protocol system for initialization, after which the client and proxy communicate directly, outside the cover-protocol. Houmansadr *et al.* [9] examined these schemes, along with StegoTorus [23], and found numerous inconsistencies and issues where they did not properly mimic some part of the cover-protocol correctly, leading to easy detection by a censor, violating the goal of *unobservability*. Common problems include failure to mimic side protocols such as control channels, not correctly reacting to errors like malformed or missing packets, and producing predictable patterns not reflective of actual cover-protocol traffic. Houmansadr *et al.* thus argue that mimicry of this form is fundamentally flawed, since there are too many interdependent protocols and subsystems to accurately mimic outside the cover-protocol system.

We argue that even if these flaws are overcome by re-engineering the systems to actively participate and embed data directly in the cover protocol, as FreeWave does, the resulting schemes would still be significantly flawed due to their use of cover protocols that do not support the requirements of generic client-proxy interaction. These flaws lead directly to attacks violating both *unobservability* and *unblockability*, as we describe in the remainder of this paper.

3. EXPERIMENTAL SETUP

This section briefly covers the experimental setup for SkypeMorph, FreeWave, and CensorSpoofer along with some of the metrics we're interested in.

3.1 SkypeMorph

For the SkypeMoprh experiments, we extracted the code for the packetizer module from SkypeMorph [1] and created a stand alone client and server that utilized this module. Included in the client and server were modules allowing fine grained control over network conditions that the censor might want to control, such as latency, jitter, packet loss, and packet replay. The client and server ran on separate machines running Ubuntu 12.10 over a 100 Mbps LAN.

3.2 FreeWave

Due to patent issues there is limited access to the modem specified in [11], so we rebuilt the modulator based on the specifications using a square root raised cosine filter pulse function to map symbols to the waveform which was then passed through a passband filter with center frequency f_C. The demodulator only consisted of shifting the received audio signal by the center frequency f_C and then running it through another square root raised cosine filter in order to extract the symbols. Since our experiments were only concerned with the transmission of symbols, the remaining parts of the modem, including the channel encoder, bit interleaver and QAM mapper, were omitted. These were implemented in Matlab using the Communications System Toolbox.

We focused on two main aspects of the functionality of the modem. First is traffic generation related to packet lengths produced by sending modem traffic over Skype. To collect this data two Ubuntu 12.10 virtual machines were started running Skype version 4.1.0.20, one machine we'll designate the client and the other the server. Once a Skype call was initiated between the client and server, the client was configured with pavucontrol [2] to redirect audio from the soundcard to the microphone. Then an audio file was played while running `tcpdump` to collect packet lengths sent from the client to server during transmission. Along with the audio generated by the modem, we used samples of speech data from the Oregon Graduate Institute Center for Speech Learning and Understanding's "22 Language" telephone speech corpus [15]. Samples longer than 10 seconds were played over Skype while recording the packet lengths, resulting in a total of 44 samples across 5 languages.

The other metric we are interested in is distortion of the raw symbols being sent by the modem (not to be confused with the actual IP traffic sent) along with the quality of the underlying audio communiciation. For this we used the SILK SDK [3] which includes tools to encode and decode audio files while varying parameters and simulating certain network conditions, such as packet loss. Using the codec tools instead of sending the data over Skype gives FreeWave the best case scenario, removing other noise added by network conditions. In addition this allowed us to collect a much larger sample size while varying different parameters, producing more accurate results. Then, to measure the quality of actual audio instead of the modem, we used a matlab script [4] to measure the MOS of audio compared to the output from the SILK encoder.

3.3 CensorSpoofer

The main focus of CensorSpoofer were the asynchronous transmission mechanisms. For these experiments a simple client and server were built where the server would transmit directly to the client with various forward error correction possibilities. CensorSpoofer mimics VoIP clients that use constant bitrate (CBR) codecs which dictates the traffic patterns. We chose to mimic G.711 which sends packets of size 190 bytes every 20 milliseconds. In addition it would simulate various types of packet loss, including burst or random, along with modules allowing more specific and targeted packet loss strategies.

[1] http://crysp.uwaterloo.ca/software/CodeTalkerTunnel.html

[2] http://freedesktop.org/software/pulseaudio/pavucontrol/

[3] http://cdn.dev.skype.com/upload/SILK_SDK_SRC_v1.0.9.zip

[4] http://www.mathworks.com/matlabcentral/fileexchange/33820-pesq-matlab-wrapper/

4. DIFFERENTIAL ERROR TOLERANCE

Anonymous communication must guarantee reliable transmission across its censorship-resilient system. While previous systems rely on the underlying mechanisms of TCP to guarantee reliable transmission, SkypeMorph, FreeWave and CensorSpoofer all transmit data over UDP using cover-protocols that have some amount of error tolerance. This forces those systems to implement their own mechanisms for reliable transmission, which a censor can interfere with and effectively block the underlying anonymous communication while still leaving a usable medium for legitimate traffic.

4.1 Packet Loss Thresholds

The main way a censor can attempt to interfere with anonymous communication traffic being sent over error tolerant channels is by inducing some amount of packet loss in an effort to disrupt the anonymous communication while still allowing legitimate usage of the cover-protocol. The majority of voice and video communication systems have various ways of dealing with packet loss, such as forward error correction and lowering the bitrate of the information being sent. While what amount of reduced quality due to packet loss is "acceptable" is an inherently subjective question, here we'll cover some work looking at the trade-offs.

To score the quality of a VoIP call the mean opinion score (MOS), which is the average of a set of subjective rankings on a scale from 1 to 5, where 5 represents the best quality possible. Ding and Goubran [6] looked at the impact of packet loss rates on the quality of VoIP communication and how it affected MOS.

Codec	0%	5%	10%	15%	20%
G.729	4.31	3.76	3.45	3.11	2.79
G.723.1 r63	4.25	3.74	3.41	3.10	2.76
G.723.1 r53	4.22	3.71	3.37	2.99	2.74

Table 1: Mean opinion scores of different VoIP codecs with varying levels of packet loss rates.

Table 1 shows the results based on their model with MOS ranging between 2.74 - 3.11 when packet loss rates get in the range of 15-20%. In terms of impairment 2 represents "annoying" and 3 is "slightly annoying". We posit this represents an upper bound of packet loss a censor could impose on VoIP connections while still maintaining usability, particular with some of the newer codecs.

Zhang *et al.* conducted similar research on Skype video calls [28], looking at how various network conditions effected video rates and quality. They found that past 10% packet loss rates Skype switches from normal to conservative state. Both the sending rate and video rate plummet after this threshold, where their model shows a drop in MOS from 3.5 to effectively 0. Based on this research we can assume a hard upper bound of 10% packet loss available to a censor, anything higher would effectively block access to Skype video communication.

4.2 FreeWave Modem

In order to directly use the VoIP network, FreeWave modulates raw data through a modem which transmits data frames through the VoIP client and over the network to the FreeWave proxy. Each data frame sent by the modem begins with a known preamble block used for synchronization, ensuring the demodulator is able to find the starting place of the actual data symbols sent. After the demodulator filters and samples the audio waveform it receives, it scans through and finds the point of maximum correlation and declares that to be the starting point of the data frame. If the wrong starting point is identified the demodulator will be unable to correctly decode the rest of the data, giving a censor a potential avenue to interrupt data transfer over FreeWave.

There are several different parameters and modes of operation that, while not explicitly mentioned in [11], are inherent in the design of the modem and will effect a censor attempting to cause the modem to desynchronize. First there is the length of the preamble, where the longer preambles give the modem a higher probability of correctly finding the starting point. Next there is the data frame length and how many data blocks are transfered. The preamble and signal parameter blocks could only be sent at the very beginning when the FreeWave client initiates the connection with the FreeWave server and remains in constant communication with null data being transfered during inactivity. Similarly the modem could be in constant communication with null data, however each data frame would be a fixed size of N data blocks. Another option is that the modem directly handles the data being transfered and sends batches of data at a time in the frames.

We first examined the effects of varying packet loss rates on correctly synchronizing with different preamble lengths. Using the modem we recorded 10 seconds of transmission of random symbols and encoded the output with FEC enabled. Then for each packet loss rate, the output from the encoder was decoded 100 times with the randomized packet loss rate so we could determine the percent of transfers that would succeed. We next fixed a starting point for the preamble, and varying the preamble length would scan through the decoded symbols, calculating 16000 points of correlation, and marking the preamble as the point with the highest correlation score. If this point was the actual start of the preamble the transfer was marked as a success, otherwise it was labeled as failed, even if the actual preamble had the 2^{nd} highest score.

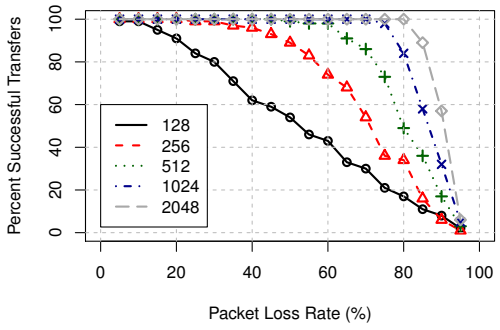


Figure 1: Percent of successfull transfers in FreeWave with different preamble lengths, while varying the packet loss rate.

Figure 1 shows the results of varying packet loss rates with respect to different preamble lengths. We see that even using fairly small lengths still results in a large percent of successful transfers with standard packet loss rates one would expect in a normal network. This is a combination of the fact that error correction in VoIP is able to compensate for small amounts of packet loss, and that correlation with random data produces very low scores, resulting in a fairly low threshold the preamble's score must be above. However we see that no matter how long the preamble, extremely high packet loss rates above 90% are able to prevent the modem from properly synchronizing, since with that high of packet loss the FEC codes are unable to recoup the loss data. While these packet loss rates would effectively kill all VoIP communication, the modem only embeds the preamble at the beginning of the connection. A preamble of length 2048 would take roughly 0.25 seconds to transmit, so a censor only needs to apply high packet loss rates for less

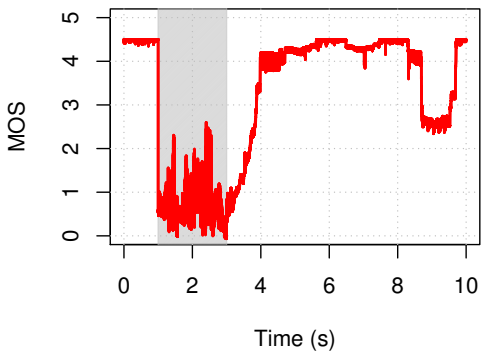


Figure 2: Targeted packet dropping to cause desynchronization in the shaded region with resulting effects on the Mean Opinion Score of the communication.

than a second to cause modem desynchronization while leaving the remainder of the connection untouched. Figure 2 shows a scenario where a censor applies 95% packet loss for a two second interval it believes contains the preamble, while leaving the remainder of the transmission in tact resulting in only a few seconds of incomprehensible communication.

In order to perform targeted packet dropping to prevent synchronization, a censor would need to know an approximation on when the preamble was sent in order to drop a large percentage of those packets. This is directly related to the different configurations of how data frames are composed and when they are sent. The first method mentioned where synchronization only occurs during the initial connection is easy to interrupt based on the evidence that high packet loss prevents synchronization with almost any preamble length. Even using a wide window for possible preamble packets, if a censor merely drops enough packets at the very beginning of the communication, it will prevent all the FreeWave modem from synchronizing while only being a minor inconvenience on actual VoIP communication. The logical extension from this protocol is to have fixed data frame sizes, allowing repeated chances at synchronization. Given predictable pattern of fixed data frame lengths and preamble positions, a censor can effectively target packets only containing preamble data. This allows a censor to completely desynchronize the modem with low packet rate loss, meaning any actual VoIP communication would be relatively unaffected allowing honest clients to still communicate. Another potential mode of operation is to have the modem send a data frame whenever it receives data either directly from the browser or off the network. In this case, the modem is only sending data frames when it actually has data to send and is quiet the rest of the time, only needing to synchronize over each frame. Since the preamble is always at the beginning of the data frame, a censor can just target the first burst of packets sent to drop in order to distort the preamble block.

4.3 SkypeMorph

In SkypeMorph the client and proxy directly communicate outside the cover-protocol system, so in order to handle data corruption and packet loss SkypeMorph implements its own retransmission mechanisms adapted from TCP to fit their traffic shaping model. The packetizer module in SkypeMorph has one thread for sending packets and one for receiving packets, both of which handle packet loss and retransmission. The sending thread ensures that packets get flagged as an ACK every 100 ms or so with the last sequence number seen. The receiving thread keeps track of the last ACK seen, and on seeing the same ACK 4 times in a row it resets the sending thread's head sequence number, causing the data starting from the new sequence number to all be retransmitted.

4.3.1 *Tracking ACKs*

A censor might want to target and interfere with packets containing acknowledgements in order to slow down or even halt the data transfer, so they need some way of accurately tracking ACK packets. Here we take advantage of the fairly deterministic way in which packets are flagged as ACK. The sending thread in the packetizer keeps track of the last time t_{ACK} an ACK packet was sent, and the first packet sent at time $t \geq t_{ACK} + 100$ms gets marked as an ACK packet. Therefore, the censor can keep track of the last time it knows an ACK packet was sent, and the first packet seen past the 100 ms window it knows is the next ACK packet. However, due to network jitter we cannot always assume the first packet seen after 100 ms is the ACK packet. In some cases either the actual ACK packet is received before the 100 ms interval and would be missed, or a non-ACK packet is delayed and appears to be the first packet after the 100 ms window, getting misclassified.

To accommodate this, the censor can keep track of different paths of probable ACK packets. First, we'll consider a *train* of packets as a group of packets in which the difference in the successive receive times for each packet is less than some ϵ [5], i.e. for any two consecutive packets received at times t_i, t_{i+1}, we have $t_{i+1} - t_i < \epsilon$. Then, when examining a window of probable ACK packets, consider the first packet in each packet train as possible ACK packets and branch a path off from there. We only consider the first packet because there is a higher probability of it being the ACK packet, and it also decreases any potential from skewing too far from the actual path (in that it is better to underestimate than overestimate).

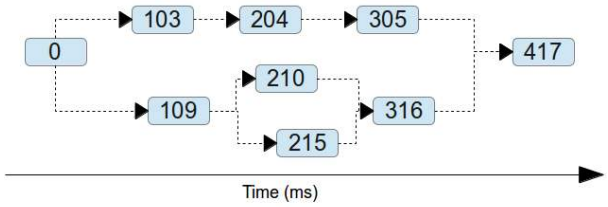
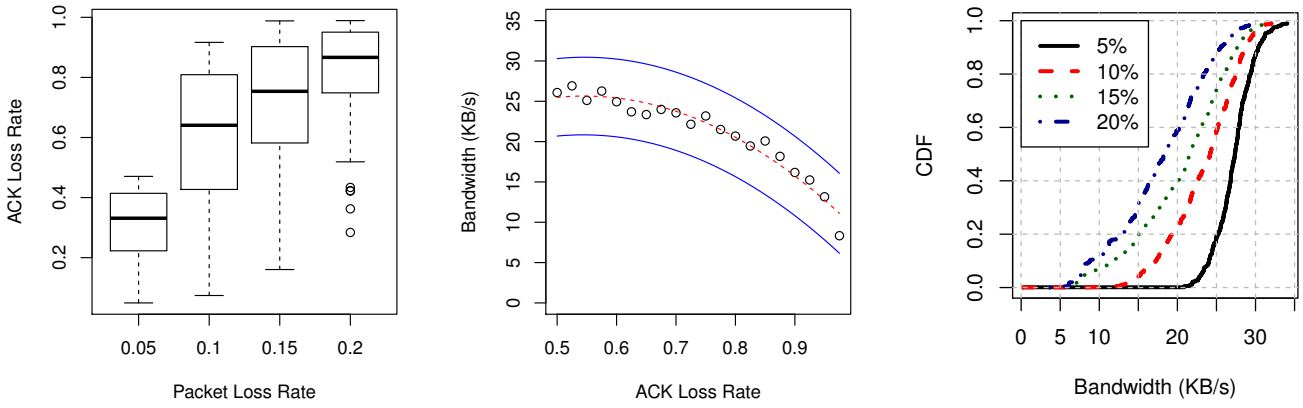


Figure 3: Example of a censor tracking paths of ACK packets in SkypeMorph, where the boxes correspond to potential ACKs flagged by the algorithm with the time they were recieved.

An example of this process can be seen in Figure 3, where multiple paths branch off from possible ACK packets. Note that it is entirely possible, and even likely, that paths will merge back to single possible ACK packets when there is only one possible packet train available in both windows. The problem with this technique is many times multiple packets could be potential ACK packets where a censor wants to select just one. To accomidate this, the censor computes a probability for each packet based on the probability distribution $p(x)$ of time differences between successive ACK packets. Since a packet may have multiple paths from some starting point, we first compute the probability of individual paths with times $\{t_1^i, t_2^i, \ldots, t_n^i\}$ as such:

$$P(path_i) = \frac{1}{n-1} \prod_{j=1}^{n-1} p(t_{j+1}^i - t_j^i)$$

[5] We used ϵ =1 ms in our experiments



(a) Packet vs ACK Loss Rate

(b) ACK Loss Rate vs Bandwidth

(c) Packet Loss vs Bandwidth

Figure 4: While the censor is attempting to drop ACK packets in SkypeMorph, (a) shows percent of ACKs dropped with different caps on packet loss rate (b) looks at how varying ACK loss rates effect overall bandwidth and (c) shows CDF of available bandwidth based on packet loss rates.

Then the probability assigned to the actual packet is just the average of the probabilities computed over all paths belonging to that packet.

4.3.2 *Dropping ACKs*

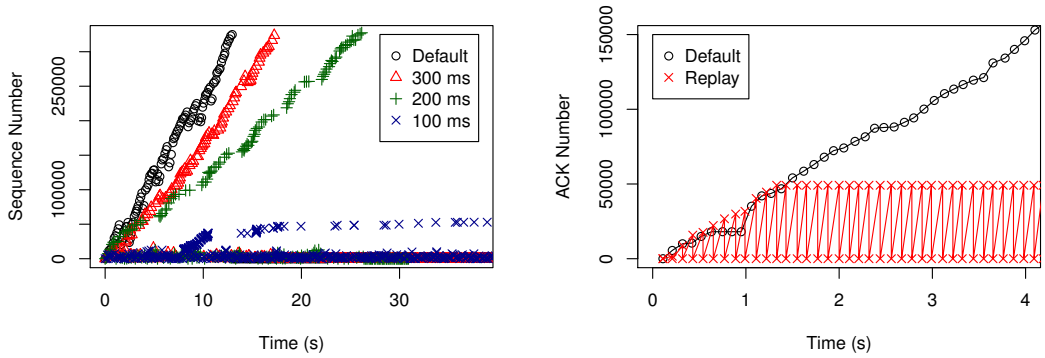
One avenue for distrupting the data transfer is for the censor to drop as many ACK packets as possible, increasing the delay in responding to packet loss causing throughput to drop. First we explored the range of ACK loss rates that can be achieved for varying packet loss rates. The packet dropping algorithm described was implemented on the client side with overall packet loss capped at a set rate. For each packet loss rate 100 downloads were performed to get the distribution of ACK loss rates achieved. Figure 4a shows these distributions for packet loss rates of 5, 10, 15 and 20%. Note that out of all packet sent, a full 10% of them are marked as ACK packets. This places a hard upper bound on packet loss rate of 5%, where at most 50% of all ACK packets will be able to be dropped with perfect accuracy, which did have a maximum ACK loss rate of 47%. Interestingly, packet loss rates of 10% and 15% had a very large span of possible ACK loss rates. These wide discrepancies are caused by the fact that occasionally in the ACK packet tracking algorithm you will see two distinct paths emerge. Dropping packets in both paths would result in packet loss rates exceeding the cap, forcing the algorithm to choose the probabilistically higher paths, which it will ultimately stick with the remainder of the transfer unless they happen to merge later. When it happens to choose the wrong path to follow, you see the very low ACK drop rates in Figure 4a. The distribution narrows considerably for packet loss rate of 20%, due to the fact that it's able to drop packets in both paths and still stay under the packet loss cap, something which could not be done with packet loss rates of 10 and 15%.

Dropping ACK packets from the client to server causes the server to take longer in responding to packet loss and start resending lost data. The end result from this is that it will decrease the throughput and increase the time it takes the server to transmit data, which is what the censor is ultimately interested in. To determine the effects of various ACK loss rates, the client was configured to directly drop ACK packets at a capped rate. The experiment would vary the ACK loss rate, performing 100 downloads for each chosen rate, while the server had a static packet loss rate of 10%. This was kept static because varying it had little effect on the overall bandwidth. Since the server rewinds the entire buffer back to the first packet lost and then retransmits everything beyond that, it doesn't matter how many packets past that initial lost packet were dropped. The experiments showed an initial drop from 34 KB/s to 27 KB/s even with the smallest amounts of ACK packet droping, and then the bandwidth held stable until ACK loss rates went past 50%. The results for rates past this threshold are in Figure 4b, showing the regression of the median bandwidth value along with the 95^{th} confidence interval. While there is a slow decrease in observed bandwidth, after 80% it accelerates downward reaching as low as 5-10 KB/s for ACK loss rates past 90%.

Given the large variation in both ACK loss rates and observed bandwidth, we want to see what the end result is when looking at packet loss rate from targeted packet dropping compared to the observed bandwidth, as these are the two parameters a censor is concerned about. Figure 4c shows a CDF of observed bandwidth for packet loss rates ranging from 5 to 20%. Since packet loss rate of 5% has an upper bound of a 50% ACK drop rate, it's not surprising that we don't see the bandwidth drop much below 25 KB/s as this was the constant rate seen for ACK loss rates ranging from 5 to 50%. For packet loss rates from 15-20%, considered to be a rough upper bound on the censor packet dropping capability, we see roughly about a quarter of transfers dropping below 15 KB/s bandwidth. While these rates may not seem like a large reduction compared to the initial 34 KB/s seen with SkypeMorph, the initial paper [18] noted that the normal bridge operated at 200 KB/s, implying that the combination means 90-95% reduction in available bandwidth.

4.3.3 *Replaying ACKs*

While the packet dropping techniques served more as a throttling mechanism than outright blocking, an active censor can also attempt to replay packets over the cover-protocol, while having little impact on legitimate traffic. SkypeMorph uses sequence numbers to keep track of what parts of the buffered data are being transfered, so upon receiving any overlapping or duplicate data, it's able to detect this and either drop the packet or discard the data.



(a) Time to transfer data with varying ACK replay intervals

(b) ACK number sent by client with and without single ACK injection

Figure 5: SkypeMorph ACK replay attacks with (a) showing the progression of the sent sequence number when replaying 4 ACK packets at a time at different intervals and (b) shows the effects on the received ACK number when injecting a single ACK packet every 100ms.

Before the check for duplicate data happens, SkypeMorph first checks the packet flag to determine whether the packet is an ACK or not, and if so processes the ACK number before the data duplication check is performed. This is done because dummy packets not transferring any real data can be flagged as ACK, so the ACK check needs to be done before the data processing. This opens up a potential packet replay attack to the censor, where they have the capability of replay old ACK packets seen and have them legitimately processed by the SkypeMorph client, even if the data is discarded. The censor runs the same ACK tracking algorithm as discussed previously and waits until all paths merge into a single possible ACK, increasing the probability of correctly selecting an ACK packet. This ACK packet can be replayed to the server in bursts of 4 at a time, so they will be received simultaneously causing the server to reset its current sequence number and roll back the buffer.

For the experiment we configured the client to record the first packet that appeared alone in the possible ACK window, and replay it 4 times in a row at a fixed interval, while the server was configured with a static 10% packet loss rate. We ran an experiment in default mode with no ACK replays, and then replaying ACKs in intervals of 300 ms, 200 ms, and 100 ms. The results for this experiment are shown in Figure 5a. Replaying ACKs at intervals of 200 and 300 ms does significantly increase the amount of time it takes to transfer data, but it doesn't completely kill the transfer. This is because after the train of replays are sent, the next valid ACK that the server receives will reset the current ACK value and will roll forward the buffer to this position, as it assumes it has received everything before the ACK counter. For the interval of 100 ms, we see a very dramatic reduction in how much data is able to be sent, having it take around 40 seconds to send as much data normally sent in the first 5 seconds.

This result leads to a lower resource and more efficient replay attack. The server needs to receive 4 packets with the same ACK number before it rolls back and starts retransmitting data. Therefore, while inducing a small amount of packet loss from the server to the client, a censor can simply inject a recorded ACK packets once every 100 ms, continually causing the server to reset the current ACK value and preventing the ACK counter from ever reaching high enough to rewind the buffer. Figure 5b shows the ACK number received by the server under default operation and while performing the single ACK injection. We can see the received ACK number jumping between the legitimate ACK value and the replay ACK being injected. Once a packet is dropped from the server to the client, the client keeps sending the server the same ACK number in an attempt to notify of data loss, but since the injected packet keeps reseting the current ACK counter the server never resends the data, stalling out the transfer and completely killing the connection.

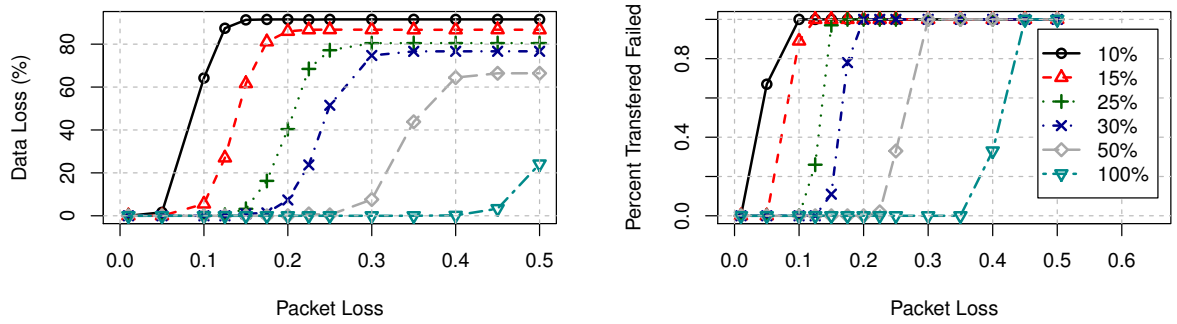
4.4 CensorSpoofer

CensorSpoofer uses asynchronous communication in order to hide the identify of the proxy, having data flow only from the proxy to the client. This means the client cannot notify the proxy of lost packets or missing data like SkypeMorph was able to do. So in order to reliably transmit data they suggest two methods of error correction, one using an XOR encoder/decoder to send redundant packets and the other to use forward error correction codes such as Reed-Solomon. We explore these methods and how they hold up to a censor attempting to disrupt communication between a CensorSpoofer client and server.

4.4.1 XOR Encoding

The XOR encoder/decoder has a redundant packet sent for every n packets transmitted, so given packets $(p_1, p_2, \cdots, p_n)$, the packet $r = p_1 \oplus p_2 \oplus \cdots \oplus p_n$ is constructed and sent after packet p_n. By including a simple packet counter at the beginning of each packet, the client will be able to detect if a packet was lost, and as long as $n-1$ of the remaining packets in the n packet window were received, the client will be able to reconstruct the missing packet simply by XORing the packets together. Assuming uniform packet loss, this type of system will be able to handle a rate of $\frac{1}{n+1}$ packet loss. There are, however, problems with making this assumption.

The first problem is that one of the common causes of packet loss is due to buffers filling up and having to drop packets, which in turn causes bursts of packets to be dropped [4]. This means that there will be a very high chance that packet loss will cause more than one packet to be dropped from the n packet window, meaning that the lost data will not be able to be recovered. The more troublesome problem is that an active censor could prevent any client from attempting to use CensorSpoofer by dropping a few consecutive packets, ensuring that data being transfered over CensorSpoofer is lost while barely effecting the usage of legitimate VoIP communication. As we can see in Figure 7, even using the smallest possible value of $n = 2$ a censor can still cause a large amount of data loss, ensuring all data transfers would fail when using XOR encoding.



(a) Percent Data Loss

(b) Percent Transfered Failed

Figure 6: Effects of packet loss in CensorSpoofer while producing varying overhead values based on Reed-Solomon parameters, looking at both (a) percent of data loss and (b) percent of failed transfers while using Reed-Solomon with different overhead.

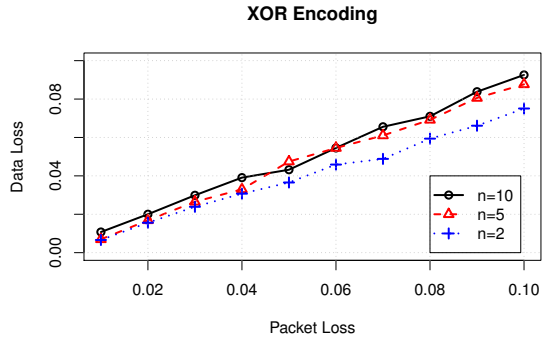


Figure 7: Percent of data loss in CensorSpoofer compared to packet loss rates when using XOR encoding with different n packet windows for redundant data.

4.4.2 *Reed-Solomon*

The other method mentioned that could be used by CensorSpoofer was a forward error correcting code such as Reed-Solomon, a popular code used in many error-prone mediums. Reed-Solomon codes are parameterized by encoding messages of k symbols, where each symbol is m-bits and the resulting encoded block is $n = 2^m - 1$ symbols. Such a code can correct up to $n - k$ errors if the location of the erroneous symbols are known, otherwise it can only correct up to $\frac{n-k}{2}$ errors. While this offers more reliable error correction in the presence of packet loss, we can still run into some of the same problems that we previously did. The naive way to perform the encoding is to split the data into blocks b_i of size k symbols, encode each block $b'_i = \texttt{Reed-Solomon}(b_i, n, k)$ resulting in n symbols, then transmit the encoded blocks $b'_1, b'_2, b'_3, \ldots$ to the client. Since reliable transmission means that *all* data encoded in the blocks must be recovered, a censor that can cause enough packet loss to corrupt a single block would cause the entire transmission to fail. For example, a common Reed-Solomon code has $m = 8$ bit symbols with $(n, k) = (255, 223)$ message and block lengths. So in order to send a 320 KB file, a server would need to transmit 1470 encoded blocks, only one of which needs to be corrupted in order to stop the transmission.

From examining both these schemes, we can identify the real problem is that for a server sending an averaged sized web file, a censor only has to drop a small amount of continuous packets to cause a portion of the data to be corrupted, thus disrupting the entire transmission. In order to fix this problem, we need to introduce a bit interleaver which causes a single block to be dispersed throughput multiple packets instead of being concentrated in a single packet. Using a random bit interleaver, we experimented with different Reed-Solomon parameters, fixing $n = 255$ and varying k while expressing the combination in terms of overhead of extra data transfered, equivalent to $\frac{n-k}{k}$. For each set of parameters we performed 100 downloads of a 320 KB file and recorded the average percent of data loss during the downloads and the percent of failed transfers. Figure 6 shows the packet loss thresholds for overhead values ranging from 10 to 50%. With the combination of a random bit interleaver and forward error correction, the server would still need to encode with 50% overhead to to avoid losing data below the 20% packet loss rate threshold.

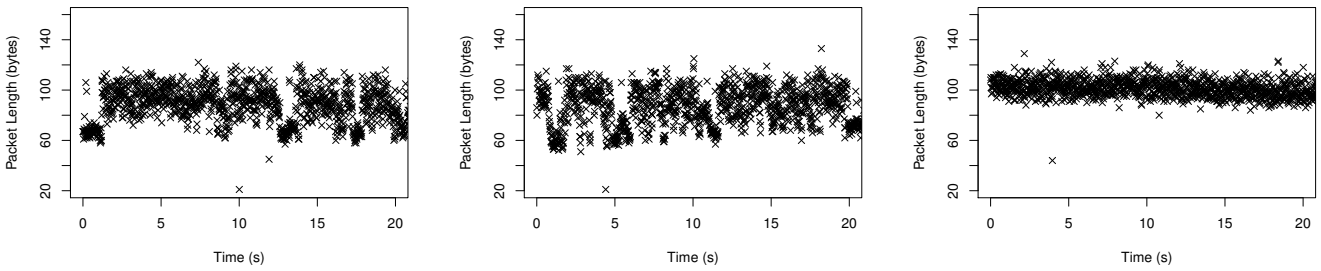
5. PROXY MODEL IN P2P SYSTEMS

Many of these censorship circumvention systems attempt to apply a client-proxy system of anonymous communication to a dynamic peer-to-peer system outside their control. For example, SkypeMorph and FreeWave both use Skype or other peer-to-peer VoIP systems as their cover-protocol. This section explores some of the difficulties that arises when forcing a proxy model into a peer-to-peer system.

5.1 Tor Over Skype

The main problem SkypeMorph attempts to solve is the ability of censors to enumerate Tor bridges and blacklist them. By using Skype as the protocol to mimic, it makes it hard for censors to detect and enumerate these bridges, and outright blocking popular services like Skype is something many censors would like to avoid. However, attempting to use the Skype network to tunnel Tor connections through is problematic due to fundamental different use cases for Tor verses VoIP. The Skype system is peer to peer with sporadic short lived connections made between unique clients, while the client-proxy model in Tor sees long lived connections between many clients and a single proxy. These differing use cases open up potential avenues for a censor to fingerprint SkypeMorph.

Figure 8 shows the ratio of daily bridge users from Syria, Iran, and China compared to the total number of bridge relays in the Tor network. Syria and Iran peaked at around 8-12 times as many users as available bridges, and while China currently fluctuates below that, they at one point saw ratio as high as 120 users to bridges be-



(a) English Packet Lengths (b) Portuguese Packet Lengths (c) Modem Packet Lengths

Figure 9: Skype packet lengths of English and Portuguese speech samples from the OPI corpus compared to modem audio sent over Skype.

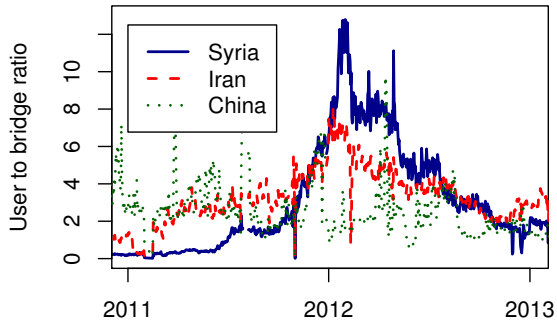


Figure 8: Ratio of daily Tor bridge users from Syria, Iran, and China compared to the overall number of available Tor bridges.

fore China started blocking access. Having so many unique clients connecting through each bridge daily is going to create traffic patterns quite different than one would expect from a normal Skype user. For example, long lived connections spanning many hours will be much longer than the most video calls where only 5% last longer than and hour [5], having multiple simultaneous connections overlapping at different time windows, and clients that seemingly communicate non-stop with a large number of unique clients around the clock. These characteristics do not even need to be able to definitively identify SkypeMorph connections, they can tolerate some small amount of false positives and only need to be able to exclude a large number of actual Skype connections. Once a connection is flagged a censor can attempt to create a SkypeMorph connection to the client under suspicion in order to verify that it's a SkypeMorph bridge.

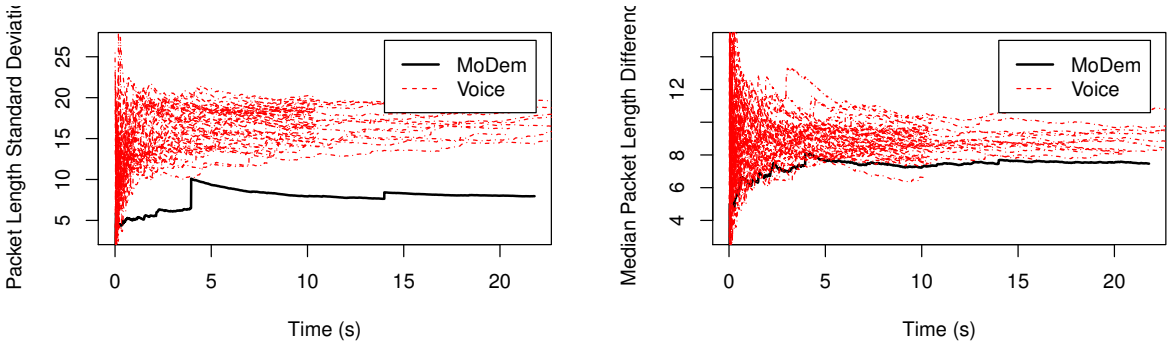
This also points to a problem that would be faced when deploying SkypeMoprh bridges into the Tor network. A large number of Tor bridges are already blacklisted by censoring countries, only new bridges unknown by these countries deploying SkypeMorph would be able to be reached by the clients. This implies an even smaller base of reachable SkypeMoprh bridges available to these countries, which could be potentially swarmed as clients attempt to use them amplifying the statistics mentioned previously, making it even easier for the SkypeMorph bridges to be flagged.

5.2 FreeWave Proxies

FreeWave prevents this type of fingerprinting by using supernodes in the Skype network as proxies, which then forward the connection to the actual FreeWave server and off to the end destination. However, there are other issues for relying on an external system for proxying connections. At the end of the day, software developers building a product that runs on top of an existing system have to make assumptions about how that base system will behave. One of the largest problems with directly utilizing an existing VoIP client for censorship circumvention is that these assumptions, which security guarantees can depend on, do not always hold up over time. Changes made without announcement can result in failures of the system to uphold its security goals. An excellent example of this is FreeWave's assumptions about how and when Skype clients attempt to use supernodes as a proxy for VoIP connections. FreeWave assumes that by placing a Skype node behind a NAT box it will cause Skype to automatically use a supernode to bridge VoIP connections. This was shown by prior research [13] to be the case. However Skype is a constantly changing system and its behavior evolves over time, in many cases without announcement.

While at the time FreeWave was designed Skype might have been more willing to use supernodes, currently Skype clients attempt, no matter the details of the network they reside in, to directly connect to each other. A Skype client sitting behind a NAT box will send multiple packets to the client it is calling, even if that client also resides behind a NAT box. Only if those packets fail to reach their destination will the caller fall back to a supernode. Additionally, the callee will attempt to send packets in response directly to the caller, even if the caller is actively using a supernode to reach the callee. The issue for FreeWave users is that this allows the censor to flag connections that are using Skype to connect to FreeWave servers. The censor can easily enumerate the IP addresses of the FreeWave servers by placing calls to the FreeWave VoIP IDs and examining which IP addresses Skype attempts to directly connect to. The censor can then flag hosts that send UDP packets to, or receive UDP packets from, those IP addresses. This issue can, to some extent, be fixed by adding the FreeWave server to a firewall on the user's machine, and user's IP address to a firewall on the server side. However, it still serves as an excellent illustration of the issues with using an active system that is in constant flux. In order to not rely on the underlying architecture of Skype, FreeWave would have to resort to a framework similar to SkypeMorph, where proxy IDs are secret and distributed out of band. This would allow FreeWave clients to directly connect to proxies using the Skype ID. But then we are operating in the same environment presented in Section 5.1 with all the same problems of using a proxy model in a peer to peer system.

The peer to peer system used by many VoIP based systems present alternative methods for the censor to detect FreeWave users. Continuing to focus on Skype, using supernodes as proxies exposes the



(a) Packet length standard deviation over time

(b) Average Packet Differences Over Time

Figure 10: While tracking connections over Skype, these show how the modem in FreeWave compares to actual voice data when comparing the (a) standard deviation of observed packet lengths and (b) the average difference between difference between packet lengths

user to identification should the censor observe the supernode. As shown previously, the censor can use Skype to enumerate the IP addresses of FreeWave servers. Since the supernode is a simple one hop proxy, the censor can then defeat the unobservability of FreeWave simply by observing traffic through supernodes. While many censors might find this out of their scope of capabilities, others would find the task trivial. For example, China currently forces users to run its own version of Skype called TomSkype [14]. This is currently done so Chinese censors can view instant messaging chat between users. It would be a simple change to force TomSkype to only select supernodes that reside inside China, making linkage between the two IP addresses carrying out a call possible.

6. CROSS-CONTENT DELIVERY

Extensive research has been done on information leakage in encrypted VoIP calls [26, 25, 24], showing how a censor can identify languages spoken, recognize key phrases, and even reconstruct parts of the conversation. This is possible due to the fact that when using variable bitrate (VBR) codecs, common in applications like Skype and Google Talk, there is a strong correlation between how much voice data there is to send and the packet length (in contrast to constant bitrate codecs which have fixed lengths). Previous methods that take advantage of this information leakage are generally fairly complex, relying on Hidden Markov Models or classifiers such as $k-$Nearest Neighbor or χ^2. While these techniques might work for analyzing a targeted VoIP call, implementing these methods on a large scale would be very computationally intensive and impractical or even impossible to perform across all calls. To this end, Houmansadr *et al.* demonstrate that FreeWave over Skype has similar traffic statistics to actual Skype communication, comparing minimum, maximum, and average packet sizes.

While the very basic traffic statistics are similar, there still might exist an intermediate test geared specifically towards FreeWave traffic that can still be performed with ease. The simplest model used for exploiting the information leakage was the χ^2 in [26] for language classification. In this case the model had to differentiate between 22 different possible languages, our task is considerably easier only needing a binary classification. Using the packet lengths generated by the samples from Oregon Graduate Institute corpus [15] mentioned in Section 3, we compared the distribution of packet lengths to those generated by the modem. Figure 11 shows the probability density functions of the minimum, maximum, and average packet lengths seen over the OGI sample set, with the corresponding value for the modem marked on each density.

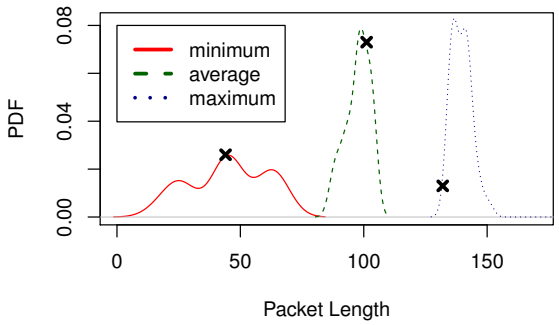


Figure 11: Distribution of minimum/average/maximum packet lengths of actual voice data, with **x** marking values computed for the FreeWave modem packet lengths

The minimum and average packet length for modem overlaps almost perfectly with the densities for actual speech, and while the maximum value is close to the left tail, it still overlaps implying that you cannot distinguish modem from potential legitimate VoIP traffic using only these metrics, confirming the original claim. However, as seen in Figure 10, there is still a very noticeable difference in stream of packet lengths seen with the modem compared to actual speech. Modem packet lengths seen in Figure 9c have a much tighter distribution then either the English or Portuguese samples.

Using this intuition, we can build computationally simplistic binary classifiers that can detect modem traffic from actual speech. Figure 10 shows the divergence of modem verses speech while tracking either standard deviation of the observed packet lengths over time, or simply keeping track of the average inter packet length difference. The average inter packet length difference starts to diverge for the modem connection after around 10 seconds, but while the average is lower than all but one of the samples, it's still fairly close to the other averages potentially making it harder to differentiate and lead to false positives. Tracking the standard deviation produces much better results, starting to significantly diverge after only 5 seconds. Furthermore, the standard deviation on packet length for the modem ends up being 2.0 - 2.5 times lower than all of the speech standard deviation, making it much easier to classify

without the risk of false positives. Being able to use such a low resource classifier lets a censor conduct widespread monitoring of encrypted VoIP calls instead of being limited to targeting specific connections.

7. CONCLUSION

With the increase in technologies available to censoring regimes, achieving unobservability in censorship-resilient systems is becoming increasingly difficult. In response there have been a new wave of systems that attempt to use popular services such as VoIP as a cover-protocol for anonymous communication. While the use of mimicry has been shown to be a flawed approach, these systems can be improved to actively participate in the cover-protocol, removing their reliance on mimicry techniques. We show that even removing the dependence on mimicry is not enough to achieve unobservability alone, as there are significant problems in delivering anonymous communication over these cover protocols.

First, we show there are issues in differential error tolerance between the anonymous communication and legitimate cover-protocol traffic. By using an unreliable channel to tunnel data, these systems must implement their own mechanisms for ensuring reliable data transmission. Due to the fact that the cover-protocols are error tolerant and can still operate while incurring packet loss rates from 10-15%, a censor is able to interfere with the mechanisms attempting to ensure reliable transmission, limiting the use of anonymous communication while retaining usability of legitimate use of the cover-protocol. Next, we demonstrate the inherent problems in forcing a client-proxy model on to a peer-to-peer system. Diverging use cases for the different systems allow for noticeable traffic patterns that can be flagged by censors, easily breaking unobservability. Additionally, we explore the pitfalls of relying on a dynamic external network for proxies, as changes can quickly break censorship circumvention. Finally we look at issues in cross-content delivery, showing that even when fully participating in the cover-protocol system, issues arise in attempting to deliver content outside the normal scope of operation, leading to low cost methods of detection.

Acknowledgments We thank the anonymous reviewers for their valuable feedback and suggestions. This research was supported by NSF grants CNS-0917154, CNS-1223421, and CNS-1314637.

8. REFERENCES

[1] JAP: The JAP anonymity & privacy homepage. http://anon.inf.tu-dresden.de/.

[2] Knock Knock Knockin' on Bridges' Doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[3] The Anonymizer. https://www.anonymizer.com/.

[4] BOLOT, J.-C. End-to-end packet delay and loss behavior in the internet. In *Conference proceedings on Communications architectures, protocols and applications* (New York, NY, USA, 1993), SIGCOMM '93, ACM, pp. 289–298.

[5] BONFIGLIO, D., MELLIA, M., MEO, M., AND ROSSI, D. Detailed analysis of skype traffic. *Multimedia, IEEE Transactions on 11*, 1 (2009).

[6] DING, L., AND GOUBRAN, R. A. Speech quality prediction in voip using the extended e-model. In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE* (2003), vol. 7, IEEE, pp. 3974–3978.

[7] DINGLEDINE, R., AND MATHEWSON, N. Design of a blocking-resistant anonymity system. Tech. rep., Tor Project, November 2006.

[8] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium* (2004), USENIX Association.

[9] HOUMANSADR, A., BRUBAKER, C., AND SHMATIKOV, V. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2011 IEEE Symposium on* (2013), IEEE.

[10] HOUMANSADR, A., NGUYEN, G. T., CAESAR, M., AND BORISOV, N. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)* (2011).

[11] HOUMANSADR, A., RIEDL, T., BORISOV, N., AND SINGER, A. I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention. NDSS.

[12] KARLIN, J., ELLARD, D., JACKSON, A. W., JONES, C. E., LAUER, G., MANKINS, D. P., AND STRAYER, W. T. Decoy routing: Toward unblockable internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI)* (2011).

[13] KHO, W., BASET, S. A., AND SCHULZRINNE, H. Skype relay calls: Measurements and experiments. In *INFOCOM Workshops 2008, IEEE* (2008), IEEE.

[14] KNOCKEL, J., CRANDALL, J. R., AND SAIA, J. Three researchers, five conjectures: An empirical analysis of tom-skype censorship and surveillance. In *USENIX Workshop Free and Open Communications on the Internet (FOCI)* (2011).

[15] LANDER, T., COLE, R. A., OSHIKA, B., AND NOEL, M. The ogi 22 language telephone speech corpus. In *Proc. Eurospeech* (1995), vol. 95.

[16] LYON, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[17] MCLACHLAN, J., AND HOPPER, N. 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* (2009), ACM.

[18] MOHAJERI MOGHADDAM, H., LI, B., DERAKHSHANI, M., AND GOLDBERG, I. Skypemorph: protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 97–108.

[19] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.

[20] SCHUCHARD, M., GEDDES, J., THOMPSON, C., AND HOPPER, N. Routing around decoys. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM.

[21] SMITS, R., JAIN, D., PIDCOCK, S., GOLDBERG, I., AND HENGARTNER, U. Bridgespa: improving tor bridges with single packet authorization. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society* (2011), ACM.

[22] WANG, Q., GONG, X., NGUYEN, G. T., HOUMANSADR, A., AND BORISOV, N. Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 121–132.

[23] WEINBERG, Z., WANG, J., YEGNESWARAN, V., BRIESEMEISTER, L., CHEUNG, S., WANG, F., AND BONEH, D. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM.

[24] WHITE, A. M., MATTHEWS, A. R., SNOW, K. Z., AND MONROSE, F. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Security and Privacy (SP), 2011 IEEE Symposium on* (2011), IEEE.

[25] WRIGHT, C. V., BALLARD, L., COULL, S. E., MONROSE, F., AND MASSON, G. M. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on* (2008), IEEE.

[26] WRIGHT, C. V., BALLARD, L., MONROSE, F., AND MASSON, G. M. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob. In *Proceedings of the 16th USENIX Security Symposium* (2007).

[27] WUSTROW, E., WOLCHOK, S., GOLDBERG, I., AND HALDERMAN, J. A. Telex: anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Conference on Security (SEC)* (2011).

[28] ZHANG, X., XU, Y., HU, H., LIU, Y., GUO, Z., AND WANG, Y. Profiling skype video calls: Rate control and video quality. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE.