TÉCNICO LISBOA

Unobservable Covert Streaming for Internet Censorship Circumvention

Diogo Miguel Barrinha Barradas

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors:
Prof. Luís Eduardo Teixeira Rodrigues
Prof. Nuno Miguel Carvalho dos Santos

Examination Committee

Chairperson:	Prof. José Carlos Alves Pereira Monteiro
Supervisor:	Prof. Nuno Miguel Carvalho dos Santos
Member of the Committee:	Prof. Henrique João Lopes Domingos

September 2016

Acknowledgements

Firstly, I thank my advisors, Professor Luís Rodrigues and Professor Nuno Santos, for welcoming me as their student and for their enthusiastic guidance throughout this work. I thank my family for the endless support and encouragement throughout my studies. Lastly, I thank all my friends who have turned the last few years into a delightful journey.

Lisbon, September 2016
Diogo Miguel Barrinha Barradas

For my parents,

Resumo

Vários governos aplicam técnicas de vigilância e censura em larga escala na Internet, de forma a impedir os seus cidadãos de aceder ou publicar informação. Para evadir o controlo dos censores no acesso à informação, sistemas recentes possibilitam a criação de canais de comunicação encobertos ao encapsular dados em protocolos de transmissão de conteúdo multimédia. Infelizmente, a introdução de um canal encoberto pode introduzir discrepâncias entre os padrões que caracterizam o fluxo original de pacotes e o fluxo que transporta o canal. Se estas discrepâncias forem estatisticamente significativas, estas podem ser observadas por um censor. Assim, um dos principais desafios que se levantam na concretização desta técnica consiste em garantir que o canal encoberto não é observável pelo censor.

Tendo em conta este desafio, esta tese visa o estudo da codificação de informação arbitrária no canal vídeo de aplicações de vídeo-conferência, de forma a que as características do tráfego resultante se assemelhem às características do tráfego de vídeo-chamadas legítimas. Particularmente, a tese propõe e avalia diferentes alternativas para a codificação de informação no canal, tendo como objectivo a maximização da taxa de transferência e a preservação das características de tráfego de vídeo-chamadas legítimas. O protótipo desenvolvido encapsula pacotes da camada de rede, suportando o encaminhamento de qualquer protocolo transmitido sobre TCP/IP. Os resultados da avaliação do sistema mostram que é possível atingir uma taxa de transferência de 0.4 KB/s ao manter a não-observabilidade do canal encoberto, permitindo a execução de aplicações comuns tais como FTP, Telnet ou Wget.

Abstract

Repressive regimes are known to deploy large-scale surveillance and censorship mechanisms in order to deter their citizens from accessing or publishing rightful information in the Internet. To evade the censors' control over the access to information, recent systems enable the creation of covert channels by tunneling data through popular media streaming protocols. Unfortunately, the covert channel may induce patterns in the resulting packet stream that distinguish themselves from the packet patterns that characterize regular streams. If the differences are statistically significant, the censor may be able to unveil, and subsequently block, the covert channel. Therefore, one of the main challenges in implementing this technique is to ensure that the covert channel remains unobservable from censors.

Considering this challenge, this thesis studies the encoding of arbitrary data on the video channel of widely used video-conferencing applications, such that the traffic characteristics of legitimate video-conferencing calls are preserved. Particularly, the thesis proposes and evaluates different alternatives to encode information within a video-stream, in order to maximize the available throughput while remaining indistinguishable from legitimate video-conferencing traffic. A prototype of the system, named DeltaShaper, tunnels network layer packets and supports the forwarding of any protocol that runs over TCP/IP. The experimental evaluation's results show that it is possible to achieve a throughput of 0.4 KB/s while maintaining unobservability, which allows to run standard applications such as FTP, Telnet, or Wget.

Palavras Chave
Keywords

Palavras Chave

Análise de Tráfego

Encapsulamento de Tráfego

Evasão de Censura na Internet

Privacidade

Síntese de Fluxos de Vídeo

Vídeo-conferência

Keywords

Traffic Analysis

Traffic Encapsulation

Internet Censorship Circumvention

Privacy

Video Stream Synthesis

Video-conferencing

Contents

List of Figures

List of Tables

Acronyms

CBR Constant Bit Rate

CDF Cumulative Distribution Function

DPI Deep Packet Inspection

ECC Error-Correcting Code

EMD Earth-Mover's Distance

FTE Format-Transforming Encryption

ISPs Internet Service Providers

RTS Real-Time Strategy

SIP Session Initiation Protocol

SNI Server Name Indication

TLS Transport Layer Security

VETH Virtual Ethernet

VBR Variable Bit Rate

VoIP Voice over IP

VM Virtual Machine

1 Introduction

This thesis addresses the problem of Internet censorship circumvention. It studies the use of popular video-conferencing applications such as Skype as vehicles for bi-directional covert channels, allowing for censorship-resistant communication. The idea of using different forms of media, including video-conferencing, to encode covert channels has been recently explored in the literature in several ways. However, ensuring the *unobservability* of the covert channel remains a significant challenge. Informally, a covert channel is deemed unobservable if there are no noticeable differences between a regular stream and a stream carrying covert messages. This thesis presents a solution to the problem of enabling covert communication through video-conferencing applications, while preserving the characteristics of unmodified video-streams. Covert communication is provided by tunneling TCP/IP packets, allowing for arbitrary traffic to be transparently forwarded between the parties engaged in circumvention.

1.1 Motivation

Today, most electronic communication over the Internet can be controlled by governments and/or by a few corporations. This allows repressive regimes to monitor and control the access to the Internet by employing several censorship techniques, such as blocking specific IP addresses or specific content (for example, web sites) (Aryan, Aryan, and Halderman 2013). As a result, citizens may be prevented from exercising their civil rights, for instance they may not be allowed to access information, communicate, or express opinions freely (Chaabane, Chen, Cunche, De Cristofaro, Friedman, and Kaafar 2014).

Fortunately, not even the most oppressive regimes can afford to block all electronic communication channels with the outside world, as these may be instrumental to preserve the economy of the country and the sustainability of the regime itself. In particular, there is evidence that several countries do restrict access to information but maintain operational widely used services such as Skype (Zittrain and Palfrey 2007). In fact, only in extreme scenarios, governments can afford to completely block the access to the Internet (Dainotti, Squarcella, Aben, Claffy, Chiesa, Russo, and Pescapé 2011).

In certain cases, to prevent access to specific sources of information, the censor instructs ISPs to block direct connections to these sources. To circumvent such restrictions, a typical strategy consists in accessing this information via a trusted proxy. However, this approach is only viable as long as the proxy address is not public and the connections to it cannot be easily flagged as suspicious. In order to prevent the public exposure of proxy addresses, systems such as Tor (Dingledine, Mathewson, and Syverson 2004) employ proxies (named *bridges*) whose addresses are not publicly distributed to access its overlay network. Even so, if no explicit effort is made to obfuscate the traffic towards a bridge, the traffic may exhibit patterns that make the bridge easily recognizable (David Fifield 2014). Unfortunately, the task of obfuscating the traffic is a challenge by itself: if the resulting pattern does not match any known protocol, the flow can also be flagged as suspicious and possibly blocked by a censor. Therefore, for the obfuscation to succeed, the resulting traffic should mimic existing protocols, ideally

protocols that a censor may not be willing to block. But protocol mimicking can be extremely hard to implement (Houmansadr, Brubaker, and Shmatikov 2013). Several factors concur to this difficulty, in particular: (1) the complexity of protocols' specification in terms of the number of possible states that must be mimicked, (2) the specificity of their respective implementations which may be used by a censor to distinguish legitimate from simulated protocol implementations, (3) the fact that proprietary protocols are often closed to the general public, and (4) the need to constantly update the mimicked protocol whenever a new release is put out.

A more recent and promising approach consists in providing access to rightful information using a covert channel by *tunneling* the data stealthily through protocols that are unlikely to be blocked by the censor. Here, the assumption is that it is possible to encode payload messages in such a way that the traffic pattern of the carrier protocol is not modified in a significant way. Examples of this line of work include FreeWave (Houmansadr, Riedl, Borisov, and Singer 2013), which encodes network traffic into acoustic signals sent over Voice over IP (VoIP) connections, and Facet (Li, Schliep, and Hopper 2014), which enables clients to stream censored videos over video calls.

These results are very promising but have important limitations. FreeWave is vulnerable to attacks that leverage perturbations in the network. Dropping selected packets may render the system useless, as it will corrupt the negotiation of parameters for the audio data demodulation. Also, its covert channel can be uncovered when established over a Variable Bit Rate (VBR) VoIP connection, by analyzing the generated network traces. In its turn, Facet employs a technique named *video morphing*, which consists of embedding the censored video into some carrier video such that the resulting transmission cannot be distinguished from the original by a censor, even if the agent can listen to the network and perform traffic analysis or active packet manipulation. However, Facet can only be used to transfer video content. This restriction may severely limit Facet's broader applicability to other important types of communication, namely web browsing and bulk file transfers. Providing such functionality may be fundamental to enable critical communication in the presence of a state-level censor.

The goal of this thesis is to leverage the basic principle of video morphing and extend it in order to support the transfer of arbitrary data content and not only video streaming, in a way that preserves *unobservability* (i.e, a censor must not be able to distinguish regular call streams from streams carrying covert data).

1.2 Contributions

This thesis analyzes, implements and evaluates techniques to establish a bi-directional covert channel over the video layer of a video-conferencing call, such that the generated traffic patterns are indistinguishable from those generated by regular video-conferencing calls. As a result, the thesis makes the following contributions:

- Proposes a novel Internet censorship circumvention system named DeltaShaper which establishes a covert channel over the video layer of popular video-conferencing applications. This channel enables the tunneling of network layer packets and the use of high-level application protocols.
- Delivers a comprehensive analysis over several data encoding alternatives, including the trade-offs involved between the maximization of the system's throughput and resilience against detection from a censor.

1.3 Results

The results produced by this thesis can be enumerated as follows:

- An implementation of the system, named DeltaShaper, using Skype as the carrier video-conferencing application.
- An experimental study to determine valid subsets of encoding parameters that allow the transfer of covert information, while retaining the properties of the traffic generated by regular Skype streams.
- An experimental evaluation of the implemented system's performance, comparing its adoption in opposition to the use of overt communication channels.

Research History

This work benefited from the fruitful comments of Professor Fernando Pereira, regarding the H.264 codec inner workings. We also benefited from Professor Bruno Martins' suggestions on evaluation metrics.

A paper that describes parts of this work has been accepted for publication in Actas do Oitavo Simpósio de Informática, INForum 16 (Barradas, Santos, and Rodrigues 2016).

This work was performed at INESC-ID and was partially funded by FCT and PIDDAC as part of the UID/CEC/50021/2013 project.

1.4 Structure of the Document

The remaining of this document is organized as follows. Chapter 2 provides an introduction to real-world censorship apparatus, dwelling on techniques aimed at both censorship enforcing and circumvention. Chapter 3 describes the design of DeltaShaper and Chapter 4 details the implementation of the system's prototype. Chapter 5 presents the results of the experimental evaluation of DeltaShaper. Lastly, Chapter 6 concludes this document by summarizing our main findings and discussing a few directions for future work.

2 Related Work

This chapter starts by addressing the different levels of competence exhibited by censors around the world (Section 2.1). Then, it provides a survey over Internet censorship circumvention techniques described in the literature, in an increasing order of sophistication (Section 2.2). Lastly, this chapter focuses on the survey of recent protocol tunneling systems, discussing the main advantages and limitations of such systems (Section 2.3).

2.1 Censor Operation

Oppressive regimes have prevented their citizens to make full use of their civil rights long before the Internet. Traditional media, such as books or newspapers, would be scrutinized by regime authorities in order to ensure that no offensive information would be published and accessible to the population. However, the advent of the Internet has allowed citizens to express their opinions freely and access information hosted all over the world (Al-Saqaf 2016).

In order to prevent the dissemination of information that can jeopardize the legitimacy of the regime itself, state-level censors are known to stage different Internet censorship initiatives, aimed at minimizing the leakage of information from within the nation and the access to external information, which may be contradictory to the regime ideals. Notably aggressive censorship apparatus can be observed during especially sensitive time-spans, such as election periods (Esfandiari 2013).

However, censorship comes at a cost, even for the most repressive regimes. Not all communication channels can be ultimately blocked due to social and economical motives, opening the opportunity for the devise of covert channels which can then be used to transfer rightful information. In this setting, the censor must be able to identify and block abusing connections, under the penalty of incurring in collateral damage by blocking legitimate connections. Real-world censors are known to act in a multitude of domains so as to efficiently detect and/or block selected information flows. Generally, censorship initiatives depend both on the amount of resources a censor has available and on its know-how and proficiency. The following paragraphs offer a high-level overview over real-world censors' capabilities aimed at preventing the access or distribution of rightful information.

Traffic Collection, Analysis and Interference: The surveillance capabilities introduced by the collection and further analysis of Internet traffic flowing through a censor's borders are instrumental for the devise of refined censorship enforcing mechanisms. There has been a proposal of a three level-based classification for censors exhibiting such capability (Houmansadr, Brubaker, and Shmatikov 2013), which relies on the sheer amount of raw network traffic a censor is able to observe, analyze or interfere with at any given point in time: *local adversary*, *state-level oblivious adversary* and *state-level omniscient adversary*. A local adversary (such as a corporation) is only able to observe a limited number of connections. On the contrary, state-level adversaries can observe connections on a larger scale, often with state-sponsored Internet Service Providers (ISPs) cooperation. A state-level oblivious adversary is expected

to perform Deep Packet Inspection (DPI) only to short observations of network traffic, considering that its infrastructure does not allow to keep connection records in long-term. Instead, a state-level omniscient adversary has the required infrastructure to collect, store and analyze a large set of network traces.

Broadly speaking, there are two major categories of attacks that a censor may perform to detect or to prevent the access to rightful information, when in control of the traffic that flows across its borders:

- *Passive attacks* are tied to the analysis of observed and collected network traces. Typical passive attacks are conducted with statistical traffic analysis resorting to DPI mechanisms, usually deployed within ISPs' premises.
- *Active attacks* are perpetrated by perturbing network traffic. A censor may be able to delay, drop, modify or inject content into selected network packets.

To perform these attacks the censor may employ a large spectrum of techniques aimed at identifying, interfering with, or blocking information flows. Such techniques range from simple IP address blocking to actual monitoring and manipulation of served content. DPI allows for the gain of information about several characteristics of the packets that cross a network, such as their length and content. This enables a censor either to take immediate action based on the content of each observed packet or to employ statistical traffic analysis techniques, which usually require the ability to store a large number of network traces (Wagner 2008). Such statistical analysis may be able to unveil unusual network traffic patterns that may be linked to the use of censorship circumvention mechanisms.

Rogue Software Deployment: Censorship actions may not directly depend upon the inspection and control of the network traffic crossing the censor's borders. In particular, there is evidence of government initiatives directed at increasing the reach of Internet censorship to the edges of the network. As a matter of fact, China has explicitly requested the modification of software products in the past so as to introduce surveillance and censorship components. For instance, Microsoft has partnered with Tom Online in order to provide an altered Skype version to chinese users (Knockel, R. Crandall, and Saia 2011). This modified version would actively record and censor the text messages produced by users. Another example is China's Green Dam (OpenNet Initiative 2009), a filtering software product that would allegedly protect children from browsing specific Internet content. However, the software's functionality reached far deeper into the operating system itself, actively monitoring a wide range of applications and disrupting their use, should they deal with information deemed prohibited by the censor. Despite its use has been condemned by the general population and the government had officially backed down on Green Dam's mandatory installation, the possible spreading of government-sponsored malware, aimed at enforcing such restrictions or at the identification of citizens engaging in active censorship circumvention, must be acknowledged.

Foreign Services Disruption: Rather than filtering and/or blocking the access to an information source, a censor may attempt to render that very source incapable of serving requests to its users. In a recent episode which took place in March 2015, China has staged a distributed denial of service attack against two GitHub repositories maintained by GreatFire.org, through an offensive system dubbed "GreatCannon" (Marczak, Weaver, Dalek, Ensafi, Fifield, McKune, Rey, Scott-Railton, Deibert, and Paxson 2015). The system worked by intercepting foreign traffic aimed at the domestic service Baidu, serving a malicious script into the visiting users' browser, who would then unknowingly participate in the attack. The deployment of such system for censorship purposes hinders users all over the world, departing from the typical approach of preventing information to be accessed by users co-located with the censor.

Lawful Interception Policies: The data of some widely used services, such as instant-messaging applications, may be hosted outside a censor's border, making it difficult for authorities to access the content generated by its citizens. In order to avoid the bureaucratic hassle of requesting logs overseas and make it easier for state entities to access such data, some censors have already begun to regulate the operation of foreign services within its borders. These are subject to place the servers containing the citizens' data within the country (Reuters 2016). In some settings, censors are known to replace popular foreign services, such as social networks, with domestic variants (Bowen and Marchant 2015).

Proactive Circumvention Mechanisms Disruption: Apart from the detection and further blocking of attempts to circumvent censorship carried out by their citizens, censors may try to proactively find and thwart systems offering censorship evasion capabilities. The idea is rather simple: a censor interacts with random or suspicious hosts as if it was a client interested in triggering a particular circumvention mechanism. The contacted endpoint can then be blocked shall it offer a way to engage in covert communication. For instance, China is known to proactively probe Internet addresses while looking for Tor bridges (Ensafi, Fifield, Winter, Feamster, Weaver, and Paxson 2015).

As described, state-level censors seek efficient surveillance and censorship mechanisms, driven by the need of expanding their control over the information handled by citizens. Particularly, China is recognized for possessing the most extensive and advanced Internet censorship infrastructure in the world. In practice, there is no evidence that the remaining known censors possess such advanced capabilities. According to the data of recent reports, the vast majority of censors mainly concentrate their efforts in: traffic collection, analysis and interference; pro-actively disrupting deployed circumvention mechanisms. The next section addresses the evolution of censorship-enforcing mechanisms tied to such capabilities, surveying concurrently evolving censorship-evasion techniques as well.

2.2 Censorship Arms Race

Censorship circumvention systems are continuously at an arms race against increasingly sophisticated censorship techniques. Any circumvention technique aims at achieving both "unobservability" and "unblockability":

- A censorship circumvention system lacks "unobservability" if it can be identified by the censor. To achieve "unobservability", circumvention techniques typically rely on the existence of a pre-shared secret (Feamster, Balazinska, Harfst, Balakrishnan, and Karger 2002; Burnett, Feamster, and Vempala 2010; Wustrow, Wolchok, Goldberg, and Halderman 2011; Houmansadr, Nguyen, Caesar, and Borisov 2011) and / or on the ability to obfuscate the communication (Moghaddam, Li, Derakhshani, and Goldberg 2012; Weinberg, Wang, Yegneswaran, Briesemeister, Cheung, Wang, and Boneh 2012; Wang, Gong, Nguyen, Houmansadr, and Borisov 2012). These two approaches can make it harder for the censor to detect covert communications in the network (among other innocuous traffic).
- A circumvention system is said to be "unblockable" if the flows it protects cannot be blocked without substantial loss for the censor. One of the most promising strategies to achieve this property is to tunnel flows through widely used protocols, such as those employed on cloud services, email, VoIP or video-conferencing (Brubaker, Houmansadr, and Shmatikov 2014; Zhou, Houmansadr, Caesar, and Borisov 2013; Houmansadr, Riedl, Borisov, and Singer 2013; Li, Schliep, and Hopper 2014),

in such a way that the only remaining strategy for the censor is to apply indiscriminate attacks to all flows, even to those that the censor would like to preserve.

Clearly, a system that is observable becomes more vulnerable to be blocked, given that the censor is then able to stage a targeted attack instead of an indiscriminate attack. It is therefore no surprise that achieving "unobservability" is referred as the biggest challenge facing the existing censorship circumvention systems (Houmansadr, Riedl, Borisov, and Singer 2013).

The following paragraphs present an evolution of censorship-enforcing techniques as well as systems designed to circumvent them.

2.2.1 Content Withholding

Early state-sponsored censorship techniques relied on the ability of the censor to control publication mediums, such as newspapers or TV shows. Before being published, all content would be previously scrutinized to ensure that harmful information to the regime would never be divulged. Recently, the Internet has become an excellent medium to exchange information and therefore an obvious target for censors. Although many Internet services are neither hosted nor controlled by repressive regimes, they may still offer censorship-enforcing platforms to prevent offensive content from being distributed within a country's borders.

Techniques Available to the Censor: Some popular Internet services, such as Twitter, enable governments to formally request the withholding of content within a country's boundaries (Tanash, Chen, Thakur, Wallach, and Subramanian 2015). This is mostly due to business decisions, enabling the service to remain available within several countries, even those whose governments actively perform Internet censorship.

Circumvention Strategies: In order to stealthily transmit information that would otherwise be censored, traditional steganographic techniques aimed at concealing the existence of a message by embedding it into some cover medium (Cheddad, Condell, Curran, and Kevitt 2010). Examples of such techniques are the concealment of Morse code in the length of grass in a drawing or Microdots, which allow for the representation of printed material in small dots. Variations of these steganographic techniques are still employed in the digital age, where data may be encoded within different layers, such as text or media files (Djebbar, Ayad, Meraim, and Hamam 2012; Li, He, Huang, and Shi 2011). Such techniques may be used by citizens living in countries ruled by oppressive regimes in order to publish and access rightful information by evading simple content filtering mechanisms and not having their messages targeted by content withholding requests. This may force the censor to block the access to the Internet service itself, as it may not be able to distinguish innocuous from steganographically-marked communication.

2.2.2 Simple Internet Destination Blocking

Destination blocking is one of the simplest strategies that can be used by a censor to prevent users from accessing rightful information in the Internet. It consists in identifying the endpoints that serve the content to be censored and block the access to those endpoints. In many cases, this strategy can be implemented even by a local adversary possessing few computational resources.

Techniques Available to the Censor: Consider the case where the information sources can be identified by an IP address. In this case, a censor may choose to enforce either a blacklist or a whitelist of

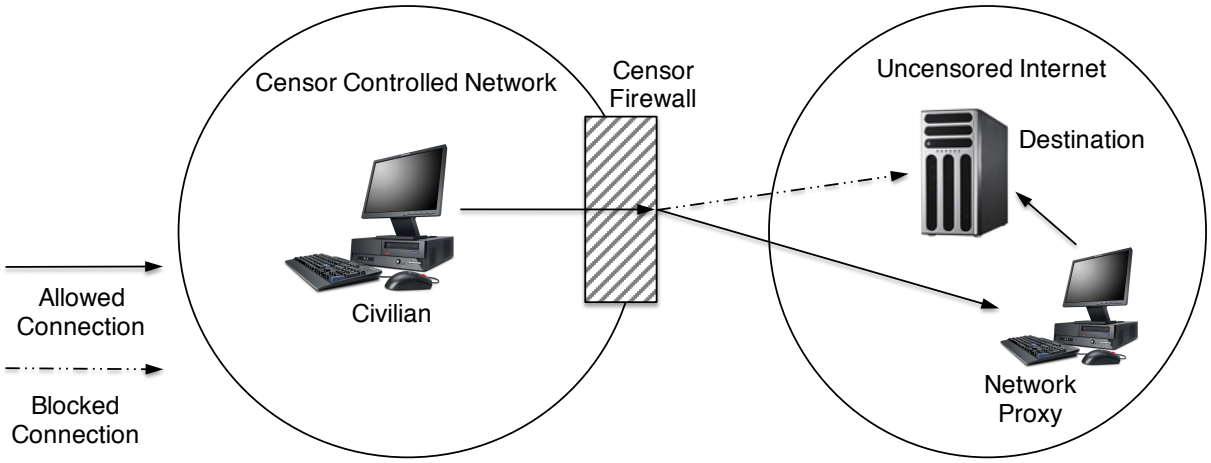

Figure 2.1: Proxy-based censorship circumvention overview.

IP addresses (Aryan, Aryan, and Halderman 2013). While being more permissive, a blacklist is easier to manage than a whitelist, considering that in the latter each new host on the Internet would need to require an authorization from the censor in order to become accessible within its network.

The interception of DNS Lookup requests can also be used by the censor in order to prevent the translation of a given hostname to its IP address. The censor responds with an IP address serving a different page, typically under its control, rather than the originally requested one (Aryan, Aryan, and Halderman 2013).

A censor may also block connections in a more selective fashion. Providing that the content is served over HTTP, the censor may filter connections based on the headers of HTTP requests. This kind of attack allows for the blocking of content at a finer granularity, restricting access only to certain pages of a given domain. To carry out this attack, the censor analyzes a page's full URL looking for specific keywords tied to potentially compromising information it wishes to block.

Circumvention Strategies: The described censorship techniques can be evaded by leveraging proxy-based traffic re-routing, which is often combined with digital steganography techniques. Instead of directly establishing a connection with the host serving blocked content, a user could direct its request to a network proxy, which would instead be accessible, enabling seemingly uncensored web-browsing. Such a mechanism is implemented by systems such as Ultrasurf (Ultrareach Internet Corp. 2002). A simple depiction of this censorship circumvention technique can be observed in Figure 2.1.

A different approach to traffic re-routing consists in relaying traffic to blocked destinations through routers, rather than end hosts. To this end, Decoy routing systems (Wustrow, Wolchok, Goldberg, and Halderman 2011; Houmansadr, Nguyen, Caesar, and Borisov 2011; Karlin, Ellard, Jackson, Jones, Lauer, Mankins, and Strayer 2011) aim to deploy special routers within cooperating ISPs' networks. A client issues an HTTPS steganographically marked request to an overt destination whose path crosses a decoy router deployed by an ISP. Decoy routers recognize such mark (which reveals a client's true desired destination) and act as a man-in-the-middle, diverting traffic to blocked destinations. A censor is only able to observe the innocent looking destination in HTTPS requests.

A different approach employing steganography relies on volunteer relay servers which secretly serve censored content. An example of such a system, named Infranet (Feamster, Balazinska, Harfst, Balakrishnan, and Karger 2002), allows clients to place visible, seemingly innocuous HTTP requests with associated additional semantics, regarding the sequence of requests performed. Infranet servers inter-

pret such sequence, steganographically embedding the requested content into uncensored images that are returned to the client. Infranet's threat model explicitly considers the blocking of HTTPS connections to be a reality.

2.2.3 Network Proxies Detection and Blocking

As already discussed in Section 2.2.2, the use of network proxies may be leveraged to evade censorship techniques that rely on the identification of hosts serving prohibited content. However, it is often difficult to hide the proxies from the censor and, once proxies are detected, they can be blocked in a similar manner as the original source.

Techniques Available to the Censor: A censor may be able to block connections to / from a given source of information, even when users re-route their traffic through network proxies. By gathering the addresses of publicly distributed proxies, a censor is able to blacklist connections to such hosts.

A censor may also be able to detect connections to unlisted network proxies. If a client connects to a network proxy through HTTP, the censor may be able to inspect the network packets' contents and flag the connection for prohibited content. Not only that, the censor may actually be able to modify a packet's content itself, preventing prohibited information to be delivered. The same strategies can be used to detect an Infranet server. If the website serving the covert content does not have a legitimate reason to change its cover content frequently, a censor may notice that the same cover content structure appears to be different when embedding different covert data.

An attack where censors may proactively probe and detect routes containing decoy routers has already been proposed (Schuchard, Geddes, Thompson, and Hopper 2012). Thus, censors capable of making routing decisions could avoid to send traffic through such routes. However, the outcome of a study about such attack suggests that a strategic placement of decoy routers would render it ineffective (Houmansadr, Wong, Inc, and Shmatikov 2014).

Providing that HTTPS is allowed and the client uses an HTTPS proxy to circumvent censorship, a censor may employ known website fingerprinting (Herrmann, Wendolsky, and Federrath 2009) techniques to look for statistical deviations of the censor's regulated traffic, identifying ciphered connections that are being used to circumvent content filtering. The same applies for connections between a client and decoy routers, where the traffic fingerprint of the covertly requested website does not match the one of the seemingly requested website.

To flag the aforementioned HTTP connections, a censor must at least be able to perform line-speed DPI in order to analyze the properties of single packets flowing through the network. This type of censor falls into the category of state-level oblivious adversary. Conversely, website fingerprinting techniques require the analysis of large data sets of network traces in order to yield more accurate results. Therefore, only a state-level omniscient adversary owns this capability.

Circumvention Strategies: A censor may still be able to detect connections to unlisted proxies through a connection's characteristics and content.

A new class of systems helping in circumvention aims to obfuscate connections' traffic, transforming it in such a way that it cannot be linked to the underlying application layer protocol. Such technique is named *traffic morphing* (Wright, Coull, and Monrose 2009). An instance of this approach consists in *protocol randomization*, where all the traffic from a regular connection is ciphered to make it seem random from a network's observer point of view.

Protocol randomization is used in Tor (Dingledine, Mathewson, and Syverson 2004), an overlay network which uses a proxy approach in order to support censorship circumvention. This procedure is necessary since connections to Tor proxies can be discovered through passive attacks by observing Tor connections' characteristic traffic. Such attacks may require line-speed DPI (David Fifield 2014) or the storage and posterior analysis of network flows (Panchenko, Niessen, Zinnen, and Engel 2011; Cai, Zhang, Joshi, and Johnson 2012).

An interesting insight over protocol randomization is that the use of length-preserving encryption does not prevent detection from traffic analysis techniques which take into account the distribution of packets' length and inter-arrival times. To avoid simple Tor traffic fingerprinting at line-speed, Obfsproxy (Kadianakis and Mathewson) uses a stream cipher to encrypt regular Tor traffic. This system hides the Transport Layer Security (TLS) cipher suite list, server name and TLS extensions, which could give the connection away, by making the traffic effectively look like random bytes. However, a censor may leverage the aforementioned traffic analysis techniques to identify Tor connections. To protect against this kind of attack and to enhance the randomization mechanism, ScrambleSuit (Winter, Pulls, and Fuss 2013) is used in tandem with Obfsproxy, manipulating packets' length and inter-arrival times.

2.2.4 Protocol Randomization Detection and Blocking

While protocol randomization can evade detection from simple protocol classifiers, a censor may take a different approach, by whitelisting approved protocols and throttling the connection for unknown or undesirable protocols.

Techniques Available to the Censor: While protocol randomization systems are able to provide fast and efficient traffic morphing, the network traffic they generate is distinguishable as it does not resemble any known protocol. Therefore, such systems can be defeated by a censor which only authorizes known and approved protocols to be used across the network.

In particular, a censor may attempt to distinguish between connections performed through protocol randomization mechanisms and regular TLS connections. While Obfsproxy encrypts every flow message, TLS uses fixed plaintext headers when establishing a connection, as part of the TLS handshake (RFC 6246 - TLSHandshake). Entropy tests on initial message headers may be able to distinguish regular TLS traffic from Obfsproxy encrypted traffic. Concretely, such entropy tests may indicate a uniform byte distribution where the plaintext TLS headers should be located (Wang, Dyer, Akella, Ristenpart, and Shrimpton 2015). This analysis does not require comparison and correlation with large data sets of stored packet samples, enabling a state-level oblivious adversary to block an unknown or undesirable protocol.

Circumvention Strategies: A different strategy for protocol obfuscation is *protocol imitation*, which aims at mimicking known and popular protocols permitted by a censor, evading the possible throttling of randomly morphed ones. The advantage lies in the unwillingness of a censor to indiscriminately block popular and fundamental services to the region's economical development. As such, existing systems try to mimic widely used protocols like HTTP and several VoIP systems' implementations.

To avoid traffic analysis of a connection's packets distribution, StegoTorus (Weinberg, Wang, Yegneswaran, Briesemeister, Cheung, Wang, and Boneh 2012) steganographically conceals chops of Tor traffic on a cover protocol messages. The chopped traffic is split over multiple connections and can be delivered out of order. Each connection is served by a steganography module, which may act like an HTTP or VoIP client/server.

SkypeMorph (Moghaddam, Li, Derakhshani, and Goldberg 2012) was designed to obfuscate connections to Tor bridge nodes by mimicking the statistical properties of Skype video calls. Unlike StegoTorus, a SkypeMorph client initiates a connection by calling a bridge, quickly dropping the call thereafter. However, the call is seemingly uninterrupted as the communication with the bridge continues by exchanging the morphed network packets carrying Tor's traffic.

CensorSpoofer (Wang, Gong, Nguyen, Houmansadr, and Borisov 2012) leverages a similar approach to both StegoTorus and SkypeMorph, while attempting to keep its server location hidden from the censor. A client can communicate to the CensorSpoofer server the page it wishes to visit through a low-bandwidth channel, such as a steganographically marked email. The server fetches the requested web page and sends it back to the client by embedding it into the network packets of a spoofed VoIP session imitation.

A different approach at protocol imitation is to use Format-Transforming Encryption (FTE) (Dyer, Coull, Ristenpart, and Shrimpton 2013) in order to produce ciphertexts that are able to match regular expressions from a user's choosing. FTE can then be used to foil regex-based DPI systems by producing ciphertexts that match the content definition of a protocol allowed across the censor's network.

By mimicking the traffic characteristics of different network protocols, protocol imitation obfuscation techniques are able to defeat the early discussed traffic analysis techniques based on the network packets' length and inter-arrival times distributions. Moreover, since these techniques aim at mimicking popular protocols allowed within the censor's controlled network, protocol imitation systems are able to evade blacklisting due to potential collateral damage.

2.2.5 Protocol Imitation Detection and Blocking

Achieving "unobservability" through protocol imitation is considered a hard challenge (Houmansadr, Brubaker, and Shmatikov 2013). This is because a system must completely mimic the target protocol behavior, including error conditions and implementation specific bugs or features. Additionally, protocols may show dynamic dependencies among several established connections, adding to the complexity of a correct imitation.

Techniques Available to the Censor: To flag or thwart a covert channel over protocol imitation only requires the censor to spot a single discrepancy when comparing with the real protocol. Favoring the censor, a large part of good candidates for imitation are also complex protocols with many of the aforementioned dynamic dependencies. Thus, even a local adversary may be able to distinguish between a legitimate protocol and an imitation by performing active or proactive attacks. Other imitation techniques may only be detected by leveraging DPI mechanisms that are feasible to be implemented even by low resourceful censors. Several attacks that a censor may apply to the previously described protocol imitation systems are presented below.

A censor may pro-actively send HTTP requests targeted at StegoTorus servers' HTTP module so as to determine if the servers' responses to both correct and malformed requests is consistent with the software fingerprint of a real HTTP server. The StegoTorus server may be flagged by a censor, shall it fail to exhibit a usual fingerprint found among real HTTP servers.

SkypeMorph lacks the correct implementation of an accompanying TCP control channel which directly reflects perturbations in the UDP stream and vice-versa. A censor may perform an active attack by dropping UDP packets and then check that such dynamic dependence is not enforced as in a legitimate Skype call.

Connections to CensorSpoofer, another protocol imitation system, can also be flagged by a censor. In particular, the censor may identify a connection to a CensorSpoofer server by detecting inconsistencies between the actual and the spoofed server. The Session Initiation Protocol (SIP) connection between a client and a CensorSpoofer server is relayed through a registrar outside the censor borders. As such, the censor is unable to verify the server's true IP address. However, a censor may send a SIP INVITE request to a dummy SIP ID on the server's alleged IP address. While a genuine SIP client would return a status message, the spoofed server may not have a running SIP client, therefore not responding to the censor's probe.

The use of FTE may be detected at line-speed by applying entropy tests on the first message of an FTE flow, an HTTP GET request. Such tests have yield results with relatively low false-positive rates (Wang, Dyer, Akella, Ristenpart, and Shrimpton 2015), allowing a censor to flag the connection with a high degree of confidence.

Circumvention Strategies: Although it does not offer a definitive solution for protocol imitation, a recent system, named Marionette (Dyer, Coull, and Shrimpton 2015), tackles some of the previous mimicking limitations. Marionette allows for the construction of a hierarchical composition of probabilistic automata, capable of controlling several aspects of protocol mimicry. Automata composition is used to control fine-grained aspects of mimicry, such as dynamic dependencies over channels, statistical properties of generated traffic and error conditions. For instance, Marionette makes it is possible to fully model an HTTP server specification so that a server leveraging protocol imitation resists active probing attacks.

Unlike the previously described imitation systems, Marionette is fully programmable through a domain-specific language, making it easy to adjust the obfuscation strategy, ranging from randomized obfuscation to full protocol imitation, while avoiding the need to rewrite the whole system. However, good candidates for imitation may be proprietary software, demanding its reverse engineering in order to build a model for imitation. Not only this is a tedious effort, it may involve its repetition once a new version of the software is made available.

To raise the difficulty of detection by a censor, *protocol tunneling* systems attempt to evade censorship by tunneling blocked content through an implementation of a protocol. Such as protocol imitation systems, these rely on the unwillingness of a censor to block popular protocols. However, systems that work by tunneling take advantage of directly coping with the chosen implementation intrinsic characteristics. As an example, FreeWave (Houmansadr, Riedl, Borisov, and Singer 2013) takes advantage of VoIP channels to tunnel modulated acoustic signals which encode Internet traffic data.

2.2.6 Protocol Tunneling Detection and Blocking

Protocol tunneling systems avoid the need to faithfully mimic a popular cover protocol by effectively using it to piggyback covert data. To prevent covert communication without hurting legitimate traffic, a censor can try to pinpoint an abusing connection, perturbing it after some degree of certainty it is being used to evade censorship.

Techniques Available to the Censor: Protocol tunneling systems do not explicitly change its cover protocol behavior. However, the embedding of covert data may induce subtle changes on a cover protocol's typical network traffic characteristics. In order to detect a covert channel over a popular protocol, a censor may try to analyze such changes and look for the following mismatches between the covert and carrier protocols (Geddes, Schuchard, and Hopper 2013):

- *Architectural mismatches* may allow the identification of information flows when the cover protocol communication architecture differs from that of the covert protocol. For instance, if protocol tunneling is performed through a centralized system, the server relay is consistent with a client-proxy architecture. Decentralized architectures may turn the endpoint acting as a proxy into a hotspot, raising suspicion from the censor.
- *Content mismatches* may allow for the detection of covert information flows, by observing that the traffic patterns generated by a given protocol are different than expected when it is used to tunnel the covert protocol data.
- *Channel mismatches* may pose a risk for the correct functioning of the covert protocol. A covert protocol which needs reliable transmission may be affected by the design of its cover protocol, which may tolerate the loss, delay or duplication of packets. For instance, the loss of selected packets may prevent the transmission of messages which are crucial for the establishment of the covert channel, while having little to no impact on the cover protocol functioning.
- *Utilization mismatches* may set off a response from a censor which possesses simple traffic analysis capabilities at its disposal. Since a protocol not directly intended to access blocked content may be used for tunneling, the user of such system must avoid seemingly abusive uses so that circumvention can remain undetectable (Zhou, Houmansadr, Caesar, and Borisov 2013). For instance, a long web-browsing session over FreeWave may take longer than the average Skype call time (Statistic Brain Research Institute 2015). Abnormal connection times through the cover protocol may look suspicious and trigger further investigation from the censor.

The detection of covert channels over carrier protocols requires a passive attack to be performed. So as to detect content or utilization mismatches, the censor must be able to keep a record of past network traces. For instance, connections from users must be recorded in order to establish common utilization rates, such as the average duration of VoIP calls. Traffic analysis aiming to find content mismatches also requires a large data set of network flows to be available. A censor must first be able to build signatures for different kinds of legitimate traffic prior to start looking for deviations on said traffic. As it stands, a more accurate detection of a covert channel is in the hands of a state-level omniscient adversary. However, even a local adversary may be able to thwart a covert channel while incurring in minor performance issues for legitimate connections, shall it be able to exploit a channel mismatch between the carrier and covert protocols.

Circumvention Strategies: To prevent a local adversary from thwarting covert protocols, protocol tunneling systems should be implemented in such a way they can cope with the chosen cover protocol intricacies. These systems pose as a good approach to censorship circumvention as they may be able to effectively evade simple or line-speed censorship mechanisms, forcing the censor to devise advanced censorship techniques based on traffic analysis. A more detailed description of such systems is presented in the next section.

2.3 Protocol Tunneling Revisited

By avoiding to mimic complex protocol behaviors and relying on the unwillingness of a censor to block popular protocols, protocol tunneling surges as one of the best approaches to censorship circumvention. Providing that a protocol tunneling implementation avoids mismatches between the covert and

cover protocol, not even a state-level omniscient adversary may be able to infer discrepancies between a regular execution of the legitimate protocol and one being used for evading censorship. The next paragraphs present a survey of existing protocol tunneling systems, concluding with a brief discussion.

2.3.1 Protocol Tunneling Using Staged Network Applications

The following systems stand upon widely used protocols which use an oblivious server relay to stage both client requests and covert data prior to delivery. They are presented in increasing order of security guarantees.

2.3.1.1 SWEET

SWEET (Zhou, Houmansadr, Caesar, and Borisov 2013) leverages email communication to tunnel covert traffic through email messages.

Description: SWEET operates by having the client exchange emails that include the covert traffic with a dedicated server. When it receives an email, the SWEET server is able to extract the covert information, process it (for instance, obtain a requested web page) and produce a reply message that tunnels the response back to the client (for instance, the content of the requested web page). The client may contact the SWEET server directly or, if the destination domain needs to be hidden, it can also be accessed indirectly: emails are sent to an account of a widely used service and this account is regularly accessed by the SWEET server (in this case, the credentials for that account need to be agreed using some out-of-band channel).

The way to tunnel information in email messages depends on the type of email services that are permitted by the censor. If the client can access directly trusted servers using encrypted emails, the information is simply transferred in the body of the email message. If the client is forced to use an outgoing email server controlled by the censor, steganography techniques must be used to encode the covert information in what appears to be a regular email message.

Advantages: A significant advantage of SWEET is that it uses a service which is unlikely to be completely blocked by a censor, given the importance that email has obtained in the daily life of citizens and business. SWEET is also able to provide an interactive communication scheme with a sufficiently small latency for supporting web-browsing.

Limitations: The traffic patterns imposed by SWEET's covert channel may be significantly different from the traffic patterns induced by regular email. For instance, under normal use a client may only exchange a few email messages per day, with relatively long inter-message intervals. On the other hand, while browsing the web using SWEET, the client may deal with many (incoming and outgoing) messages in a short interval. Thus, traffic analysis tools are likely to detect an utilization mismatch and unveil the existence of the covert channel. A better approach would be to make use of a cover service with more flexible traffic and utilization patterns, such as cloud storage, to tunnel covert data.

2.3.1.2 CloudTransport

CloudTransport (Brubaker, Houmansadr, and Shmatikov 2014) leverages cloud storage services to tunnel covert traffic through storage read and write requests.

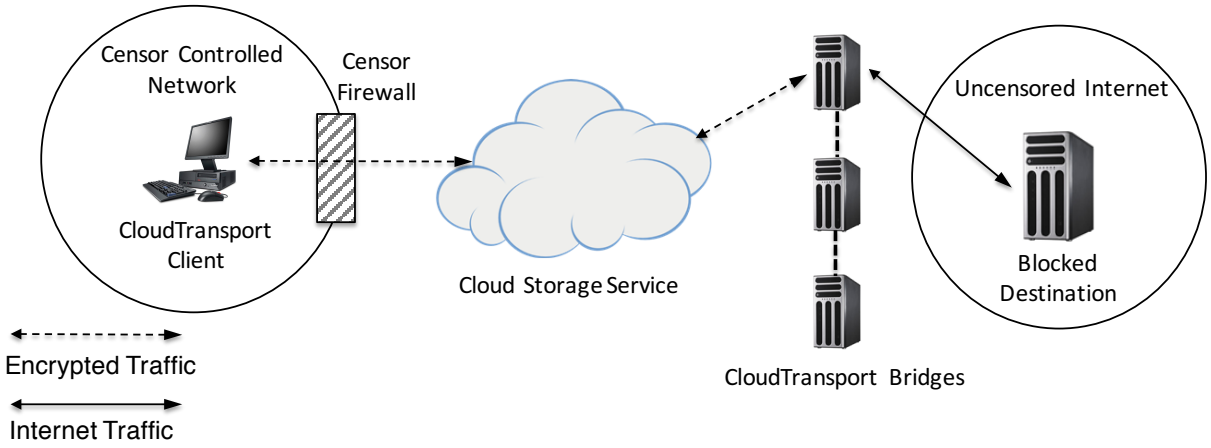

Figure 2.2: CloudTransport overview.

Description: CloudTransport operates by having the client exchange information with a server that serves as a bridge to the censored service via an intermediate cloud storage service, as depicted in Figure 2.2. The client writes a request in the cloud storage. In turn, the bridge periodically polls the cloud storage for new files; it fetches the request, serves it and writes the response back to the storage service. Finally, the client reads the storage service to get the reply. For this scheme to work, the client and the bridge must pre-agree on the storage service to use and on location of the files to be exchanged on the storage namespace.

CloudTransport also aims at helping clients to find bridges and negotiate which storage account to use for exchanging the files. For this purpose, each bridge is required to maintain a dedicated cloud storage account that is only used to rendezvous with potential clients. In turn, a client can set up its own cloud storage account for supporting the cover channel and then share the account credentials with the selected bridge, making use of the rendezvous account of that bridge. Furthermore, CloudTransport's authors suggest that some directory service could be created for bridges to advertise their services.

Advantages: As cloud storage services become more and more popular, it may be hard for a censor to completely block cloud storage without significant loss. Also, as the set of applications that use cloud storage is diverse, many read/write patterns may exist in practice and this diversity may help in making the covert channel hard to detect.

Limitations: The traffic patterns imposed by the covert channel may still be different from the traffic patterns induced by typical applications that use cloud-storage. Although the task of monitoring traffic aimed at a cloud-storage service is arguably harder than monitoring email traffic, a censor may still be able to detect content mismatches and unveil the covert channel. For instance, the censor may conduct website fingerprinting attacks in order to correlate traffic from a CloudTransport client with traffic patterns that emerge when web-browsing blocked destinations. A way to mitigate this vulnerability is to employ some of the traffic morphing techniques previously discussed in Section 2.2.4. However, as stated before, such countermeasures may also be detected by a censor.

Another shortcoming of CloudTransport is that the techniques proposed to help clients to discover and connect to bridges are prone to several attacks. First, the censor may create false bridges to divert traffic to itself and then match the observed traffic with traffic produced by clients to identify the client of a given request. In fact, by performing a Sybil attack (Douceur 2002) an adversary may easily dominate the number of advertised bridges in the system, making this attack very powerful. Also, shall it act like a client, the censor can monitor all accesses to its cloud storage account to obtain the IP addresses of

the bridges it interacts with. This enables the censor to neutralize the bridges by staging targeted denial of service attacks.

Further staged protocol tunneling approaches may aim at the mitigation of attacks that leverage the analysis of traffic patterns, by exhibiting patterns common to regular utilization of the cover system.

2.3.1.3 Castle

Castle (Hahn, Nithyanand, Gill, and Johnson 2016) leverages common commands issued in Real-Time Strategy (RTS) games to encode and tunnel covert traffic.

Description: The system employs desktop automation software to issue game commands that carry covert data. Castle works by selecting structures or immobilized units, respectively setting rally-points and attempting unit displacement. The system's authors propose a combinatorial scheme where a move/rally-point command is issued to different subsets of the available units, giving the possibility to encode more data than if a fixed number of units was chosen.

The receiving Castle client must have a way to interpret the covert data in order to retrieve the original data. Usually, RTS games keep a log of the issued commands in order to save or replay a game. This allows the receiving Castle client to fetch and decode Castle's specially crafted commands.

The authors of Castle have shown that the system's implementation may be tuned to leverage a particular game's features to improve throughput. While a general implementation of Castle was able to achieve a maximum throughput comprised between 42 and 320 Bps when deployed over three different games, a game-specific Castle's implementation achieved a throughput of 435 Bps.

As it stands, a Castle's client may be used to transmit textual data such as emails or other short messages. The authors of Castle propose a different mode of operation where a client can place a request for a web page which would be served by parallel data transfers provided by the remaining clients, acting like a web proxy. This is made possible because several RTS games allow multiple players to join a session.

Advantages: Castle's design may be adjusted to make use of a panoply of games available to the general public, such as free games like 0 A.D., Aeons or some of the best-selling RTS games ever. Since many of such games share elements inherent to the RTS genre, a censor gains little by banning a specific game, since another similar one may be used as cover. Thus, the only censor's alternative is to blanket ban all RTS games, incurring in social discontentment.

The games which Castle relies upon typically encrypt and authenticate their network communication channels to prevent cheating, thus preventing the ability of a censor to observe the commands issued by the clients. Such properties also thwart active attacks relying on rogue packet injection. Moreover, reliable data transmission channels are implemented in the application layer as the majority of RTS games transfer game data through UDP. Therefore, the system tolerates active attacks comprising the drop and delay of packets.

Although a censor cannot infer the commands placed by analyzing the encrypted game packets, the traffic flow variance caused by several game factors may be analyzed by the censor. However, by varying the number of units selected per command and limiting the rate at which game commands are issued, the traffic flow generated by Castle is indistinguishable from the traffic flow produced by an actual human player. Thus, Castle is resilient against passive attacks.

Limitations: Castle's performance is limited by two factors: i) throughput is highly dependent on the time that the desktop automation tool takes to perform the unit selection and to select a coordinate; ii) different games have different limits on the amount of units that can be selected at once, varying the amount of data that is possible to encode in a given command. Such limitations may prevent a given Castle's implementation to sustain a sufficient throughput for bulk file transfers.

2.3.1.4 *meek*

meek (Fifield, Lan, Hynes, Wegmann, and Paxson 2015) leverages domain fronting, the use of different domain names at different communication layers, to tunnel traffic over HTTPS connections to allowed hosts while establishing a covert connection to a prohibited host.

Description: Domain fronting is applied by using different domain names at different layers of the HTTPS request. In such requests, the destination's domain name may be found in three locations: the DNS query, the TLS Server Name Indication (SNI) extension and in the HTTP Host header. While a censor may observe the domain name associated with the DNS query and SNI, it is unable to check the value of the HTTP Host header, since it is hidden by the HTTPS request encryption. This allows for a domain-fronted request to use an allowed domain name on the layers that a censor is able to observe, while hiding the true desired destination in the HTTP Host header. When the frontend server receives such a request, it routes it to the covert destination indicated in the HTTP Host header. Content Distribution Networks (CDNs) are good candidates for deploying frontend servers. As part of their normal operation, they already forward requests to the domain found in HTTP Host header whenever they are unable to serve a request from their local cache.

The performance evaluation conducted over the system's deployment on Tor shows that bulk-download times are increased by about a factor of 3, when compared to downloads performed through Tor without *meek*.

Advantages: CDNs are widely used today and it may be impossible for a censor to block its usage. By using this strategy, a client may perform a request to a CDN's apparently inoffensive front domain, which will resolve to a frontend server the censor is not able to observe.

Limitations: The patterns observed on *meek*'s TCP ACK traffic are distinct from regular TLS connections. This is due to the fact that *meek*'s clients periodically poll the *meek* server, checking whether there is data to be received. Moreover, a censor may attempt to detect irregularities in the latency observed in connections to blocked destinations when compared to that expected when connecting to the advertised front domain. While the destination IP and URL of a connection may seem legitimate, *meek* does not attempt to match the traffic patterns of real CDN usage. Although the traffic flowing through CDNs can be highly diverse, machine learning techniques applied on traffic classification can be useful in distinguishing *meek*'s implementation over Tor on particular settings, like a campus network and home wireless networks (Wang, Dyer, Akella, Ristenpart, and Shrimpton 2015). Thus, such traffic classifiers are able to detect content mismatches when deployed in the environments where they were trained, rendering it an interesting approach to be deployed in localized environments.

2.3.2 Protocol Tunneling Using Multimedia Streaming Applications

The protocol tunneling systems described up to this point leverage a staged communication approach, where an oblivious server stores and forwards the communication between the client and server

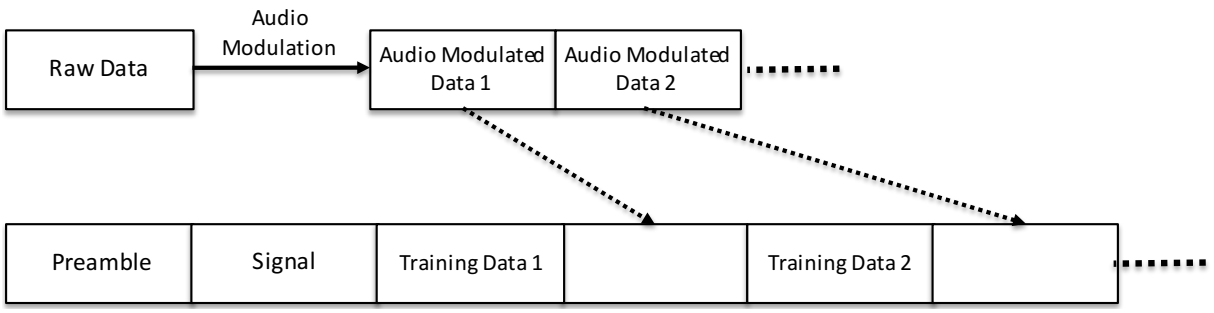

Figure 2.3: FreeWave's data frame layout.

of a given censorship circumvention system. A different tunneling approach leveraging multimedia streaming protocols can make use of an oblivious server to relay the communication or take advantage of peer-to-peer communication between the endpoints engaging in circumvention. This approach has been adopted in different systems which exhibit remarkable differences between their security properties and use cases. Such systems are presented on the next paragraphs, where both their advantages and limitations are highlighted.

2.3.2.1 FreeWave

FreeWave (Houmansadr, Riedl, Borisov, and Singer 2013) leverages VoIP connections to tunnel Internet traffic, allowing for uncensored web browsing.

Description: FreeWave's audio data is generated by a bit-interleaved coded modulation. The system uses a wrapper protocol to carry the modulated data. This wrapper protocol allows for the demodulator to synchronize with the modulator and to negotiate the modulation parameters necessary to demodulate the acoustic signal. The modulated bit stream is split in data frames. Each data frame has a preamble block which is used for synchronization. The preamble is followed by a signal block which contains the parameters used for the modulation of the data frame. Lastly, the data frame contains interleaved blocks of training data and actual data, as depicted in Figure 2.3.

The experimental evaluation conducted over FreeWave's covert data transfer rates have revealed that the system achieves a throughput in the range of 2.0 - 2.4 KBps.

Advantages: The use of FreeWave over a Constant Bit Rate (CBR) VoIP system avoids the unveiling of the system's covert channel through traffic analysis techniques. The vast number of available VoIP providers makes it hard for a censor to ban all instances of FreeWave unless it performs a potentially undesirable blanket ban over all CBR VoIP services within its borders.

Limitations: When FreeWave is used over VBR codecs it exhibits a content mismatch which can be detected by passive attacks launched by a censor. Despite that packet payloads are ciphered, in-depth analysis of the packets length distribution over time is known to be able to distinguish between the language being spoken in a conversation (Wright, Ballard, Monrose, and Masson 2007) and even to reconstruct spoken phrases (White, Matthews, Snow, and Monrose 2011). FreeWave's detection can be achieved based on the observation that the generated network packets' length distribution is nothing similar to that of a recognizable language, expected to be found in an actual conversation (Geddes, Schuchard, and Hopper 2013).

FreeWave is also vulnerable to active attacks that capitalize on channel mismatches. In particular, FreeWave's covert channel may be thwarted by preventing the synchronization of its modem compo-

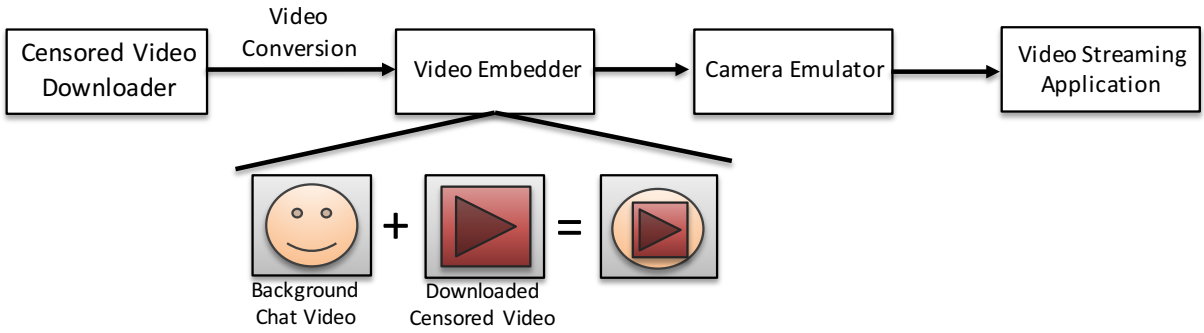

Figure 2.4: Facet pipeline overview.

nents. As previously stated, data frames must contain a known preamble block so that the demodulator can synchronize and interpret the data being issued by the counterpart endpoint. Providing that the preamble is not found by the demodulator, it will not be able to decode the actual data. One can picture several modes of operation, where both the preamble and signal blocks are included: i) uniquely at the beginning of the connection; ii) at the beginning of scheduled data frames with a fixed data blocks number; iii) at each data frame sent whenever the modem has data to send. While the VoIP channel resists to packet dropping, this may prevent the FreeWave's modem from synchronizing. Indeed, the censor may be able to block any of the three modes of operation described, shall it be able to selectively drop the packets which carry the preamble.

2.3.2.2 Facet

Facet (Li, Schliep, and Hopper 2014) leverages video-conferencing calls to tunnel censored videos.

Description: Facet enables clients to place a request for the transmission of a censored video through a Skype video-conferencing call. To this end, the client must tell the Facet server the URL of the video he wishes to watch through an instant-message of the video-conferencing service or through a steganographically-marked email. Once the Facet server receives instructions from the client, it downloads and converts the desired video, feeding it to audio and video emulators. Instead of using a real microphone and camera, Skype reads input data from the emulators and places a call to the client, which is then able to watch the chosen video.

Facet employs *video morphing*, a technique developed to ensure that the network packets generated by the video conferencing software do not directly reflect the characteristics of the censored video, but approximate those of regular video calls instead. To this end, Facet embeds the censored video in a portion of each frame, filling the remaining space with a chat video. This procedure is depicted in Figure 2.4. A user will then watch a scaled-down censored video over a background chat video. Naturally, the more scaled-down the censored video is, the better the resilience to traffic analysis. This happens because the background video characteristics dominate over those of the embedded censored video. To foil a censor's traffic classifier results, audio morphing is also required. The audio layer of the censored videos is re-sampled to simulate the lower quality of chat audio. Another interesting insight on Facet is that it avoids channel mismatches by tunneling videos over a video transmission channel. This provides an active attack resistance by design, since any perturbation in the network will cause exactly the same effect on a regular or covert video transmission.

Since a censor may block video search services, the Facet server can be used as a proxy for obtaining video URL lists. The user may send an encrypted or steganographically marked email directed

at the server (a similar approach to the one discussed in SWEET (Zhou, Houmansadr, Caesar, and Borisov 2013)) to which the server will answer.

Advantages: The described approach yields good enough results to prevent a censor from easily blocking Facet's sessions. A censor who tries to completely disrupt Facet from functioning may need to block 20% legitimate video-conferencing calls, shall the censored video occupy a total of 12,5% of the video-conferencing call display area. This area may be adjusted upwards, enabling a user to watch a censored video with better quality, while still representing possible prohibitive collateral damage to the censor.

Limitations: By design, Facet is only able to serve video content. This limits the system's applicability to other types of communication such as web browsing. This is because different kinds of data would need to be properly encoded to be transmitted, and a reliability layer would need to be in place to resist against active attacks perpetrated by a censor.

Moreover, in order to guarantee a prohibitive collateral damage for the censor, the covert video's scale factor must be small overall, damaging the quality perceived by the client. While video quality may be enough for watching a film's trailer, small scale factors may hinder the viewing experience for several videos. For instance, tutorial-like videos often require the viewer to follow steps shown on-screen. These would be hard, if not impossible, to perceive under high traffic analysis resistance guarantees.

2.3.2.3 CovertCast

CovertCast (McPherson, Houmansadr, and Shmatikov 2016) leverages the video layer of live-streaming feeds to transmit the content of blocked websites via modulated images.

Description: In order to set up the system, CovertCast servers' operators must select blocked websites which the server will scrape, modulating the respective data into images. These images are aggregated in order to be transmitted in a loop fashion through live video feeds. Then, CovertCast operators set up live video channels in video-streaming websites like Youtube, using the modulated data as input for the stream's video layer. In this way, live feeds will repeatedly broadcast a small choice of web pages. Internally, a CovertCast server downloads the selected page, encoding it into the RGB components of each frame's available pixel, up to a maximum of 6 bits per pixel. In order to make the decoding process resilient against errors introduced by the compression applied by video encoders, the system's implementation employs a fixed a square of 8x8 pixels where all its pertaining pixels encode the same color, and thus the same data. CovertCast streams 2 frames containing modulated data each second. This transmission scheme enables CovertCast to achieve a covert data transfer rate of 21.12KBps when streaming data over a 1280x720 high-definition video.

For using CovertCast, users must know the URL of the live video feed which is currently transmitting the content they wish to access. The client component demodulates images served through the live stream, extracting and saving the correspondent web content.

Since the video content is ciphered prior to transmission, CovertCast must make sure the traffic patterns it generates blend among the traffic flows generated by legitimate users. The results obtained from the system's experimental evaluation show that a censor is unable to accurately identify CovertCast's covert channel. Thus, a censor wishing to block the system incurs in significant collateral damage by erroneously classifying and blocking legitimate streams.

Advantages: Video-conferencing applications aim at enabling devices with heterogeneous computational power and bandwidth to communicate. As such, the video-compression algorithms present in

these applications are optimized for mostly static video. In contrast, video-streaming platforms take different approaches to multimedia data compression, enabling for the delivery of high-quality videos. CovertCast takes advantage of this fact to mix among legitimate traffic, being able to use all of a video frame's area to encode data. Since a censor is unable to distinguish CovertCast transmissions from legitimate transmissions, active attacks aimed at disrupting a video stream would have the same effect on both kinds of streams, potentially causing severe collateral damage.

Limitations: Due to its design, CovertCast is only able to provide a satisfactory experience by implementing a unidirectional multicast communication channel. This means that it is unable to offer an interactive communication scheme and that it can exclusively be used to consume censored content. Such a scheme implies that clients are unable to dynamically choose the content they wish to access. This fact forces CovertCast servers' operators to set up a multitude of live-feeds to serve different content and distribute the associated URLs to clients.

Additionally, the kind of media platforms CovertCast depends on can be used to rapidly disseminate potentially prohibited information. Unlike several video-conferencing applications, which have suffered from more severe blocks during times of political unrest, video-streaming platforms have been targeted and long-term blocked by real-world censors. This fact may encumber CovertCast's deployment.

2.3.2.4 SkypeLine

SkypeLine (Kohls, Holz, Kolossa, and Pöpper 2016) modulates the background noise found in VoIP connections to transmit covert data.

Description: SkypeLine introduces a new data modulation technique based on Direct-Sequence Spread Spectrum (DSSS) based steganography to hide information in the audio layer of Skype VoIP calls. The signal resulting from this modulation consists in an apparent noise signal. In order to attach covert information to the carrier signal, the system generates pseudo-noise sequences from a secret seed, shared between the caller and callee, enabling both parties to synchronize the respective demodulation operation.

Due to the nature of the data modulation technique employed by SkypeLine, the system's throughput peaks at 64bps in the presented prototype.

Advantages: SkypeLine faces a strong threat model which assumes a censor to have the capability to eavesdrop on the contents of an ongoing VoIP call. The background noise modulation approach introduced by SkypeLine is robust both to attacks that leverage traffic analysis and to those taking advantage of accessing the actual audible content of the VoIP call.

Limitations: The accuracy of the demodulation must ensure a correct recovery of the embedded covert data. SkypeLine faces two issues with regard to this aspect. Firstly, the original call audio signal may interfere with the modulated noise sequences, diminishing the distinguishability of the data to be demodulated. To enhance both signals distinguishability, SkypeLine leverages active gain control, which adjusts the volume of the noise sequences to the different volumes of the lead audio signal. Also, since the covert information is attached to the audio media itself, the noise reduction applied by the audio codecs in existing VoIP applications may filter out noise sequences carrying covert data. To account for such losses, SkypeLine employs error-correcting codes in order to achieve a greater reliability over the recovery of the original data.

By employing an heavy focus on steganographic security, SkypeLine's throughput is quite limited.

Table 2.1: Protocol tunneling systems comparison.

System	Active Attack Resistance	Passive Attack Resistance	Arbitrary Data Transmission	Interactive Communication	Reasonable Throughput ($>=$ 320Bps)
SWEET	Yes	No (limited by utilization mismatch)	Yes	No (limited by utilization mismatch)	Yes
CloudTransport	Yes	No	Yes	Yes	Yes
Castle	Yes	Yes	Yes	Yes	Yes
meek	Yes	Yes	Yes	Yes	Yes
FreeWave	No	No (w/ VBR codecs)	Yes	Yes	Yes
Facet	Yes	Yes	No	No	Yes
CovertCast	Yes	Yes	Yes	No	Yes
SkypeLine	Yes	Yes	Yes	Yes	No

This may encumber the system's applicability to several applications that demand higher covert data transfer rates. However, it remains an interesting approach for providing access to low-bandwidth hungry services such as instant messaging.

2.3.3 Discussion

The surveyed protocol tunneling-based circumvention systems are depicted in Table 2.1. This table summarizes the security and performance properties closely addressed in this section: resistance against active and passive attacks; capability to transfer arbitrary data frames; capability to engage in interactive communication; and the ability to deliver a reasonable throughput (measured with respect to the throughput obtained by the general Castle implementation, the first protocol tunneling system under analysis which is able to offer a combination of the aforementioned properties). A more comprehensive study over a broader range of security and performance properties exhibited by existing censorship circumvention systems has been presented in the past (Elahi, M. Swanson, and Goldberg 2015).

Protocol tunneling systems that create covert channels through staged network applications present good approaches at censorship circumvention. Although the majority of the systems surveyed do not implement explicit mechanisms aimed at preventing detection from traffic analysis techniques, they leverage widely used services with flexible traffic and utilization patterns to hide covert transmissions among legitimate traffic. For instance, *meek* is at the forefront of currently deployed protocol tunneling systems by mixing its traffic among diverse CDN traffic. Due to this fact, blocking *meek* could represent severe collateral damage for a censor. However, its traffic analysis resistance assumptions rest on the simple fact that it may be hard to distinguish from legitimate CDN traffic.

Although the encoding of data in multimedia streaming protocols has been addressed in the past, it is still relatively unexplored in the context of Internet censorship circumvention. Novel techniques are currently being devised in order to allow the reliable transmission of arbitrary types of data, while being resilient to active and passive attacks perpetrated by a censor. FreeWave provides the possibility for arbitrary data transmission over a loss tolerant channel. However, it is vulnerable to active attacks perpetrated by a censor, rendering such transmission unreliable. Furthermore, FreeWave is unable to resist against traffic analysis when used over applications employing VBR codecs, since the network traffic generated by its modulated data is distinguishable from that generated with real speech. Contrary to FreeWave, Facet is not intended for interactive communication. Although the system's approach seems promising from a traffic analysis resistance point of view, it is limited on the transmission of video content, assuming a non-interactive communication model. In its turn, CovertCast enables for the exchange

of other types of data over the video layer of live-streaming feeds. However, it favors scalability at the expense of interactive communication. SkypeLine assumes a stronger threat model than previous works on multimedia streaming tunneling approaches by envisioning the collusion of the streaming service provider with the censor. Its ability to provide interactive communication while remaining undetectable comes at the cost of the throughput offered by its covert channel.

As it stands, existing systems that tunnel covert data through multimedia streaming protocols fail to offer a combination of all the properties depicted in Table 2.1. In fact, although systems such as Facet or CovertCast are resilient against passive and active attacks launched by a censor, these systems fail to provide an interactive covert communication channel that allows for the transmission of arbitrary data. In its turn, due to its strong threat model and data modulation technique, SkypeLine lacks sufficient throughput for the usage of many traditional TCP/IP applications, such as a web client, focusing on low-bandwidth hungry services. Considering the limitations of existing systems, the next chapter introduces the design of DeltaShaper, a censorship-resistant communication system which aims to offer a full combination of all of the aforementioned properties. In particular, DeltaShaper aims to significantly extend Facet's video morphing technique, allowing for the unobservable and interactive transmission of arbitrary data through the video layer of existing video-conferencing systems.

Summary

This chapter introduced a survey of the main capabilities of real-world censors, as well as an overview over several deployed censorship circumvention mechanisms. As it can be observed, China is at the forefront of deployed Internet censorship mechanisms, while several other countries are slowly keeping up, deploying technical measures and passing new legislation.

While rudimentary censorship circumvention techniques reveal to be effortlessly blocked by censors, recent tunneling approaches are not easily defeated, showing to be resilient even in the face of censors which may have the infrastructure to keep large records of data and the computational power to analyze them. Thus, such censorship circumvention approaches seem the most promising and are the base for the further work introduced in the next chapter.

3 DeltaShaper

This chapter introduces DeltaShaper, a novel Internet censorship circumvention system which leverages the video layer of video-conferencing applications to embed covert data. Section 3.1 presents the fundamental goals that drive the system's design and Section 3.2 describes the threat model faced by DeltaShaper. Lastly, the design of DeltaShaper is thoroughly detailed in Section 3.3.

3.1 Design Goals

The goal of DeltaShaper is to embed a bi-directional covert channel in a regular video stream such that the network flow resulting from its transmission cannot be identified by a censor. The design of DeltaShaper is driven by five goals:

Unblockability: The censor must not be able to block the transmission of covert messages without significant degradation of the Skype service for legitimate users.

Unobservability: A censor must not be able to distinguish regular call streams from streams carrying covert data.

Video-carrier independence: The system's data encoding scheme must be able to allow for covert communication without depending on a specific video-conferencing application.

Reasonable network performance: The performance of the covert channel in terms of latency and throughput must allow for traditional TCP/IP applications to work.

Portability: The system shall work without the need to change the binary image of the video-conferencing application operating at the channel endpoints. DeltaShaper assumes that the network packets generated by the video-conferencing application are ciphered prior to transmission. Without loss of generality, this thesis focuses on Skype, a popular video-conferencing application.

3.2 Threat Model

The goal of the adversary is to detect and block Skype communication flows that carry covert messages. We assume an omniscient state-level adversary, which is able to observe, store, interfere with, and analyze all the network flows between the parties that are engaged in the communication, namely Skype connections. The censor is able to deploy proactive probing mechanisms aimed at discovering end hosts offering DeltaShaper servers. The adversary has the power to perform deep packet inspection but is computationally bounded, such that it cannot break the underlying cryptographic primitives used to cipher the content of Skype-generated network packets. Therefore, the censor is unable to observe packet contents and to reconstruct the video stream between the endpoints engaging in circumvention. The adversary has no control over the software installed on end-users computers and has not the

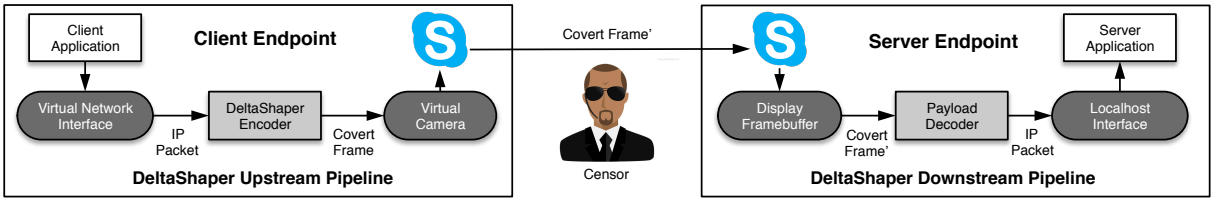

Figure 3.1: Architecture of DeltaShaper.

power to deploy rogue software with the purpose of monitoring systems on network edges. Thus, the communication endpoints where the Skype clients are executed are assumed to run trusted software. The adversary considers Skype traffic as allowed due to economical and social reasons. Therefore, the censor is not willing to arbitrarily break Skype calls, avoiding severe collateral damages. However, it may introduce controlled perturbations on the network connection in order to cause the breakdown or disclosure of the covert channel. The censor shall only mark video-conferencing calls as disallowed providing that there is strong evidence those are being used to convey some covert channel. Lastly, the videoconferencing provider (i.e., the Skype service provider) will not collude with the adversary, allowing it to inspect the video content rendered at the communication endpoints.

3.3 Design

DeltaShaper aims at establishing an unobservable covert channel over the video layer of popular video-conferencing applications. To make the system as general as possible, the architecture allows for the forwarding of network-layer packets between the endpoints engaging in circumvention. As a result, the system can support any TCP/IP application that can tolerate low throughput / high latency links.

The operation of DeltaShaper depends on an upstream and a downstream pipeline, illustrated in Figure 3.1. Traditional TCP/IP applications, such as Telnet or FTP, can be used as client / server application. When an application sends data, the DeltaShaper Encoder component receives the payload and encodes it in a video stream that is fed to Skype using a virtual camera interface. Skype transmits this video to the remote Skype instance and the received stream is captured from the Skype video buffer. A Decoder component is then responsible by extracting the payload from the video stream, transparently delivering it to the server application. The same procedure is applied at both endpoints of a Skype call, thus supporting a bi-directional channel between the client and the server applications.

3.3.1 Design Challenges

Although the principles behind the development of DeltaShaper are relatively simple, there are many design challenges, described below, that will be addressed in the next sections.

1. Conflicting data encoding requirements: Intuitively, it is desirable to encode as many covert payload bits per frame in order to achieve a high throughput. Unfortunately, this may not be possible due to two fundamental reasons. On the one hand, the video stream is re-encoded and compressed by Skype during the transmission, using lossy algorithms. One needs to ensure that the payload is encoded with additional redundancy so that it can still be retrieved at the receiver, regardless of the transformations performed by Skype. On the other hand, a careless encoding scheme will likely generate network streams whose characteristics differ significantly from a typical Skype call, making them prone

to be detected by a censor. Hence, there is the need to define a video encoding decoding scheme that is flexible enough to produce unobservable streams while offering an acceptable performance, despite the video quality degradation induced by Skype's video compression algorithms.

2. Characterization of unobservable streams. The traffic signature of a Skype call which carries an embedded covert channel should be indistinguishable from a "normal" Skype call. The challenge, however, is to define what a "normal" Skype call is. There is a need to establish objective metrics that allow for the identification of such streams through traffic analysis and to generate covert traffic that retains a similar signature.

3. Adaptation to network conditions. The properties of a "normal" Skype call may change depending on the network conditions upon which a call takes place. Moreover, video-conferencing applications' codecs are typically programmed to adjust the respective output bitstreams to the current network conditions in order to offer a good visual experience. As a result, DeltaShaper must be able to adapt its data encoding algorithm according to the current network conditions, under the penalty that a censor leverages discrepancies in the generated traffic to compromise the unobservability of covert streams.

4. Synchronization of covert channel endpoints. A consequence of DeltaShaper's encoder adaptation to network conditions is that channel endpoints must synchronize each other so that the receiver is able to interpret covert data according to the encoding scheme used by the sender. However, such a mechanism must be resilient to active attacks issued by the censor in order to avoid disruptions on the exchange of new encoding parameters.

3.3.2 Data Encoding and Decoding

This section starts by providing some basic notions on the functioning of the H.264 video codec, which will prove to be detrimental in guiding the choice of encoding parameters for DeltaShaper. Then, the description of DeltaShaper's encoding scheme is presented.

3.3.2.1 How Skype Encodes Video Streams

Skype uses the H.264 video codec to enable the transmission of live video content. Advanced video codecs like H.264 take a set of images - called frames - and explore the similarities among those to produce a compressed representation of the data (Wiegand, Sullivan, Bjontegaard, and Luthra 2003; Chen, Kao, and Lin 2006).

In order to achieve high rates of compression, video codecs depart from the traditional RGB color representation of images, employing the YUV model in their image pipelines. The YUV model is a color space defined in terms of one luminance (Y) and two chrominance (UV) components. While the luminance component conveys the brightness of an image, chrominance represents its color information. As the human color perception is more sensitive to luminance than chrominance, the YUV model allows for a reduced bandwidth regarding chrominance components. This enables video transmission errors or compression artifacts to be more efficiently masked by the human perception than using a direct RGB color representation.

In H.264, a frame is split into macroblocks, each consisting of a small matrix of pixels. To enhance compression, H.264 uses two types of macroblock prediction: intra prediction and motion compensated prediction. Intra prediction leverages information of already transmitted macroblocks for a given video

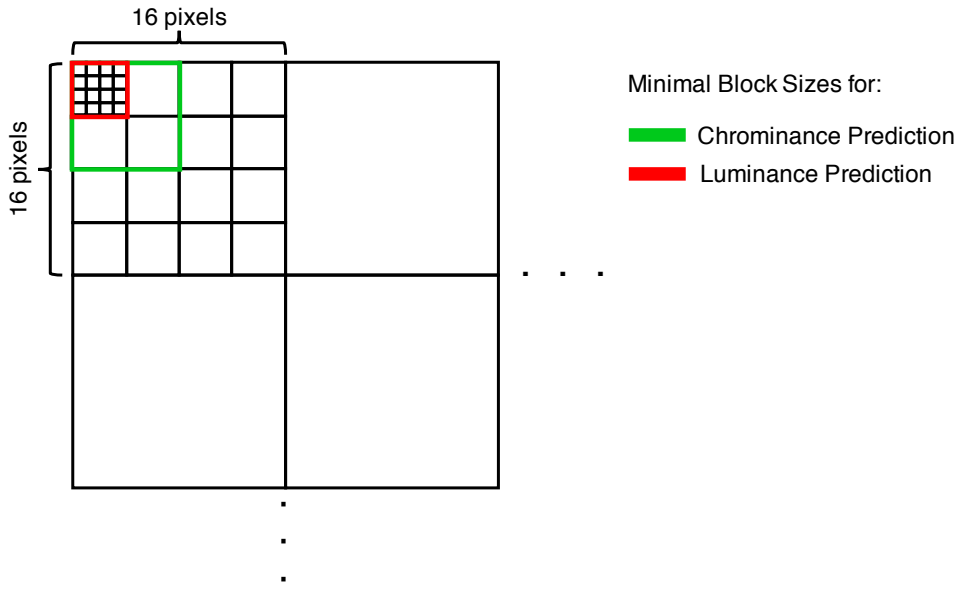

Figure 3.2: Maximum sub-block divisions on H.264 prediction modes.

frame. It attempts to predict the current macroblock by extrapolating the neighboring pixels from adjacent blocks, in a defined set of different directions. The difference between the predicted block and the actual block is then coded to be transmitted. This approach is particularly useful in flat backgrounds where spatial redundancies often exist.

H.264 provides two types of intra prediction that can be applied to the luminance component. The first type is called INTRA 4x4. Using INTRA 4x4, the macroblock, which is of the size 16 by 16 pixels, is divided into sixteen 4x4 subblocks. Then, a prediction for each 4x4 subblock of the luminance signal is applied individually. This concept is illustrated in Figure 3.2. There are nine different prediction modes, and a different one may be applied to each subblock. The second intra prediction type is called INTRA 16x16, where one out of four different prediction modes is applied to the whole macroblock. Unlike the luminance component, the intra prediction of the chrominance components of a macroblock is always performed on four 8x8 subblocks.

In its turn, motion compensation prediction leverages information from fully transferred reference frames. In an attempt to better isolate motion, macroblocks can be decomposed up to a maximum of sixteen 4x4 partitions, turning these into the minimum area used for overall predictions. Usually, for a given area, the algorithm will be able to find a matching block with little prediction error so that the overall size of the discovered motion vector plus a prediction error is lower than the size of a raw encoding for that specific block.

Next, the results from the prediction stages are transformed from the spatial domain into the frequency domain. Lastly, the resulting coefficients are quantized. This process causes a lossy compression, reducing the precision of the integer coefficients, while maintaining perceptual quality. The quantization parameter is often adjusted on the fly to allow for more or less compression, depending on the available bandwidth. DeltaShaper must deal with such video compression procedures in order to encode covert data in such a way that it does not severely interfere with Skype's video encoder.

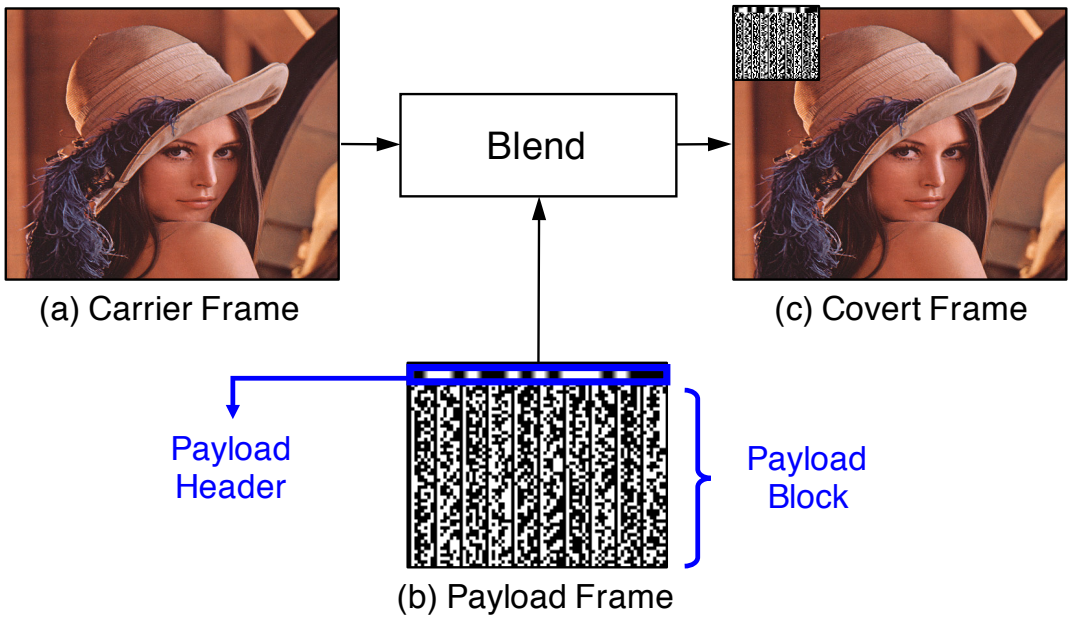

Figure 3.3: Blending payload into carrier frame.

3.3.2.2 DeltaShaper's Covert Data Encoding Scheme

Given that a video stream is a sequence of frames, that each frame is composed by a set of pixels, and that each pixel can be defined by RGB components, one needs to find the best way to encode the data bits in the available pixels. According to the XWD format specification that is used to store screen dumps created by the X Window System, an RGB encoded pixel takes 24 bits, allowing the encoding of 16,777,216 different colors. Thus, in theory, one could encode 24 bits in each pixel. Assuming a 640x480 frame (VGA resolution), it would be possible to encode, at max, 7372800 bits in a single frame.

However, there are some reasons that prevent such an encoding scheme from functioning in practice. Firstly, video processing may modify a frame's pixels in multiple ways: change the colors of each pixel, thus altering the information being transferred; omit differences among adjacent pixels, loosing all information encoded in those pixels. Secondly, it is necessary to preserve unobservability. If all pixels of a frame are used to encode data to the maximum capacity, the resulting image complexity would be significantly different from a typical image transferred in Skype, where many pixels are similar. This would cause the traffic signature of the resulting Skype stream to be extremely different from a that of a "normal" Skype stream.

To deal with such conflicting trade-offs, we propose a data encoding scheme based on two basic ideas:

1. Blend synthetic payload video into "normal" Skype video: The covert data encoding scheme generates transmitted video frames (*covert frames*) from the combination of two components: *carrier frames* and *payload frames*. Carrier frames are taken up from a pre-recorded Skype call. Payload frames consist of synthetic video frames that encode the application data to be transmitted to the receiver. Payload and carrier frames are then blended together into covert frames and passed over to Skype. Figure 3.3 shows an example of how a (a) carrier frame and a (b) payload frame are blended into a (c) covert frame. The payload frame is overlapped to top-left corner of the carrier frame. The goal of carrier frames is to mimic a realistic Skype call by modulating the network stream observed by the censor thus preserving unobservability.

Name	Description	Example
s_p	payload frame area (pixel×pixel)	160×120
s_c	cell size (pixel×pixel)	4×4
b_c	color encoding (bits)	1
r_p	payload frame rate (frame/sec)	3

Table 3.1: Payload frame encoding parameters.

2. Support tunable payload frame encoding: The payload encoding scheme depends on several parameters. Each payload frame encodes N bits of the covert message on a *payload block*. Each payload block is a synthetic image that consists of a grid of *cells*. Each cell consists of a fix-sized area of contiguous pixels featuring the same color. The color code is used to encode b_c bits of information of the payload block. The total amount of bits that can be encoded per frame N is then defined by the geometry of the payload frame and given by: $N = b_c \times n_c$, where n_c is the number of cells per frame. The communication throughput T is given by $N \times r_p$, where r_p is the rate of payload frames sent per unit of time. The encoding scheme is then defined by the parameters in Table 3.1: size of payload frame in pixels (s_p), size of cells in pixels (s_c), color encoding in bits (b_c), and payload frame rate (r_p). A data encoder is represented by the tuple $S : \langle s_p, s_c, b_c, r_p \rangle$, for example $\langle 160 \times 120, 4 \times 4, 1, 3 \rangle$. In this example, a cell takes 4x4 pixels and the payload frame size is 160x120 pixels, totaling $n_c = 1200$ cells. Given that the payload data is encoded with 1 bit, yielding a binary black-white image, the payload block is $N = 1200$ bits. Since the payload frame rate is 3 frames per second, the maximum communication throughput T is 3600 bit/sec. To decode the data from payload frames, the receiver must collect covert frames at rate r_p, extract the payload area s_p from the frame, average out the color of each pixel of each cell, and streamline the b_c bits of each cell. Consequently, to decode a payload block, the receiver must know which encoding parameters were used. For this reason, the sender appends these parameters into a fixed-format band atop the payload frame (payload header).

Together, these techniques enable DeltaShaper to handle with conflicting data encoding requirements by providing multiple degrees of freedom. Reducing the number of bits to represent color codes from 24 bit makes the system more resilient to per-pixel color change introduced by video encoding pipelines. Increasing the cell size above 1x1 helps tolerate loss of information between adjacent pixels as a result of video compression, as more pixels will be used to encode the same data. Moreover, rather than using every frame to encode new payload data, the scheme allows for a reduction in the payload frame rate, which is important to mitigate the video encoding effects that can cause loss of unobservability. Thus, by properly tuning DeltaShaper encoding parameters one can control the amount of information blended into the carrier video which will determine how close from a “normal” video-conferencing call the resulting covert video will be. Furthermore, the different video compression procedures used by popular video-conferencing applications are based in similar algorithms (Feller, Wuenschmann, Roll, and Rothermel 2011). The exploration of the space of encoding selectors provides DeltaShaper a general approach to compute valid encoding selectors without being tied to a specific video-conferencing application’s codec implementation.

Since there is an inherent trade-off between the amount of information that can be sent through the covert channel (*throughput*) and the deviation of the resulting network traffic (*observability*), there is the need to find a configuration that satisfies the best from both worlds. Section 5 provides more details on the determination of encoding parameters for Skype video-conferencing calls, specifying the conditions in which these hold. The encoding parameters can be determined not only statically through a set of offline experiments, but can also be negotiated on the fly between the communicating endpoints.

3.3.3 Preserving the Skype Traffic Signature

Ideally, it would be desirable to devise an encoding procedure that could select optimal parameters for the embedding of covert data onto a given carrier video, even before its transmission. Unfortunately, due to the complexity of video encoding algorithms, there is no trivial manner to estimate, from some easy to capture features of the original carrier video, how an encoded Skype stream will look like. In fact, the H.264 encoding process is non-normative, which means that each service provider can tune the encoding process to its like, only needing to assure that the produced bit stream adheres to a standard. In particular, H.264 prediction modes can be configured in multiple ways, making it very hard, if not impossible, to create an analytic model that guides the encoding of the covert channel such that the resulting stream produced by Skype is indistinguishable from a "normal" Skype stream. Ultimately, DeltaShaper must generate covert videos that preserve the signature of a "normal" Skype stream. To that end, typical Skype streams must be characterized in order to devise a technique that allows the system to generate covert streams that follow similar traffic patterns.

"Normal" Skype streams are designated *regular streams*. A Skype stream is *regular* it if results from a legitimate video-conferencing call between Skype users carrying no covert messages. In such cases, users normally stand in front of the camera and move sparingly as they speak. In contrast, the resulting traffic pattern is expected to be quite different if Skype is used for streaming an action movie, for instance. In such cases, frames will change more frequently and extensively causing Skype's video encoding to reflect such changes. To express this intuition that regular calls tend to follow common pattern, while inevitably having some differences, a stream is considered to be *irregular* if it differs by more than a given threshold Δ from known regular streams, in which Δ is obtained by a given *similarity function* σ. Put more formally, considering s_R to be a set of known regular streams, f a *feature function* of the stream (e.g., packet length distribution), and s_C an arbitrary stream (that may contain a covert channel), s_C is said to be indistinguishable from s_R if:

$$\sigma(f(s_C[P]), f(s_R)) \leq \Delta$$

Therefore, the frame encoding parameters P for s_C must be chosen in such a way that the resulting covert stream obeys this condition; to meet this goal, the following steps are taken:

1. Find an effective feature function (f): A feature function extracts some relevant quantitative attribute out of the packet traces that constitute a stream. Through experimental evaluation, the *frequency distribution of packet lengths* (f_l) was found to be effective at characterizing a given stream pattern in Skype. Similar reasoning was proven to be successful at differentiating Skype streams from Tor streams (Moghaddam, Li, Derakhshani, and Goldberg 2012). The packet length of the stream depends on both the input video and compression applied by Skype. Therefore, blending payload frames into the carrier frames will alter the packet length distribution. An alternative function based on the 2-gram distribution of packet lengths has enabled to differentiate regular Skype calls from the transmission of YouTube videos over Skype (Li, Schliep, and Hopper 2014). In the context of DeltaShaper, this function produces similar results as f_l. Alternative feature functions based on the inter-packets' arrival time are also considered in the system's evaluation and are described in Section 5.6. Feature functions based on the packets' content were not considered since Skype-generated packets are encrypted.

2. Find an effective similarity function (σ): A similarity function aims to calculate the difference between two feature functions. Given that f_l, which outputs the frequency distribution of a stream's packet length, was adopted, there is a need to find metrics that calculate the similarity between two probabil-

ity distributions. Previous work has adopted the 2-sample Kolmogorov–Smirnov test (Moghaddam, Li, Derakhshani, and Goldberg 2012; Vines and Kohno 2015). Informally, this test quantifies the maximum vertical distance between the empirical cumulative distribution functions of two given samples.However, the Earth Mover's Distance (EMD) (Rubner, Tomasi, and Guibas 2000) was found to yield better classification results (as it is further shown in Section 5.6) and was selected as similarity function. Intuitively, EMD$(f_l(s_R), f_l(s_C))$ represents the total amount of work that is necessary to undertake in order to transform the packet length frequency distribution of a regular stream s_{Ri} into the packet length frequency distribution of stream s_C.

3. Define a set of reference streams (s_R): Now that f and σ have been defined, a set of known regular streams must be fixed to serve as *reference streams* around which DeltaShaper's generated covert streams will compare against. Such regular streams will correspond to streaming several carrier videos that may be used by DeltaShaper in the payload blending process (as shown in Figure 3.3-a), and can be obtained by recording the packet trace of real video-conferencing Skype calls, for example.

4. Compute the similarity threshold (Δ): The similarity threshold Δ aims to set a bound to the differences that one can expect to find between legitimate regular Skype calls. To determine this value, an empirical approach is undertaken. This approach consists of creating a training set of N legitimate Skype call videos and record the packet length distribution of the resulting test stream s_i, where $0 \leq i < N$. Then, the average similarity between each test stream and every other regular stream is calculated. The threshold value Δ can then be assigned in several ways, for instance: the largest difference verified between multiple reference streams; the average similarity between all regular streams, plus a multiplicative factor based on the standard deviation. Furthermore, Δ must be assigned dynamically, as Skype's video encoder will adapt its bit rate to the current network conditions.

5. Obtain a valid encoding selector (P): The final step consists of determining valid sets of parameter instances (P) to the payload encoding scheme. A specific instance of P is called *encoding selector*. To be valid, an encoding selector must produce unobservable streams. Encoding selectors that satisfy such condition can be found by exploring the space of P generating a training stream $s_C[P]$ and verify that s_C is indistinguishable from s_R. For checking whether s_C can be identified as a regular stream, its similarity value δ can be obtained by computing the average similarity between s_C and all regular test streams in s_R. More precisely:

$$\frac{1}{N} \sum_{i=0}^{N-1} \text{EMD}(f_l(s_C[P]), f_l(s_{Ri})) = \delta, \; P \text{ is } \textit{valid} \text{ if } \delta < \Delta$$

The set of valid encoding selectors must be obtained experimentally and provided to DeltaShaper as possible encoding selectors to be adopted. If multiple encoding selectors are valid, DeltaShaper selects the one that delivers highest throughput, which is also determined experimentally. Section 5 presents the results of the empirical analysis conducted over DeltaShaper.

3.3.4 Adaptation to Network Conditions

As it turns out, a single reference stream set s_R and respective threshold Δ cannot be permanently fixed and hard-coded in DeltaShaper. In fact, according to the experimental evaluation's results (described in Section 5), the Skype stream distributions that result from playing a given (carrier) video greatly depend on the specific network conditions under which the transmission has occurred, such as bandwidth or packet loss rate. This observation brings two consequences:

The reference stream set and the similarity threshold must be set dynamically: In order to preserve the properties of unobservability on a given connection, it is necessary to adopt a reference stream set (s_R) and threshold value (Δ) according to the specific network conditions. Furthermore, it must be taken into account that network conditions may change over time, either due to contingencies of the network infrastructure or to active attacks launched by the censor. In fact, by altering the network conditions (e.g., throttling bandwidth or dropping packets) a censor may try to induce changes in the streams' properties in order to reveal suspicious discrepancies in the generated traffic.

The encoder selector must be set dynamically: In responses to changes in network conditions, it may be necessary to change the frame encoding parameters in order to preserve a stream's indistinguishability. Moreover, the new parameters must be agreed upon by both endpoints. The negotiation of parameters between parties must also be resilient to active attacks issued by the censor aimed to prevent the agreement and cause denial of service.

To make DeltaShaper adapt to the network conditions, the client endpoint performs the following operations: 1) before starting data transmission, determines the closest reference stream set for the current network conditions, 2) from that set, obtains the ideal encoding parameters, 3) starts encoding data frames according to the determined parameters and embeds the encoding parameters directly into each frame, 4) periodically, readjust to network conditions, by repeating this procedure starting from 1. Next, these steps are addressed in detail:

1. Finding the reference stream set: The main cause for the change of the threshold value is a modification of the reference streams' packet distributions upon changes in network conditions. To accommodate to those changes, before the client starts encoding payload data into carrier frames, the client performs a calibration operation in which it transmits the carrier video alone without embedding any payload blocks. The client transmits this video for a certain calibration time T_C and in this process collects relevant features about the stream packets. This sample will allow the client to obtain a fingerprint of the carrier video stream for those particular network conditions. The reference stream set shall then be chosen by computing the similarity metric σ between the captured stream and the different reference sets. The client takes as reference the set that results in the lowest σ value. As these reference sets are captured in different network conditions, it is expected that the current stream shall be closer to the reference set captured under the same conditions.

2. Determining the encoding parameters: Based on the previously obtained reference streams set, the ideal parameters' selection can be performed twofold. On one hand, DeltaShaper can obtain the ideal parameters based on a reference table, which consists in a map that tells for each reference stream set and pre-defined cover video combination which parameters could be used for achieving better throughput, on a given network setting. On the other hand, DeltaShaper can transmit several covert videos produced with different encoding parameters and carrier videos, compute the similarity for each one and choose the one which provides the largest throughput while maintaining unobservability. Both reference streams and reference tables can be determined empirically.

3. Agreeing upon frame encoding parameters: Since the frame encoding parameters can change dynamically negotiated, the client must be able to tell the server which parameters to adopt when decoding a given payload frame. To this end, each covert frame has a small metadata block which indicates the parameter values used in the associated payload block. The metadata block contains: a data type indicator which states the frame's function (calibration, dummy, payload); the total payload area used to encode covert data; the size of each payload cell; the number of bits encoded per cell; and a parameter used for error-correcting purposes. This makes each frame self-contained and the parameter agreement resilient to denial-of-service attacks launched by the censor. The metadata block, represented

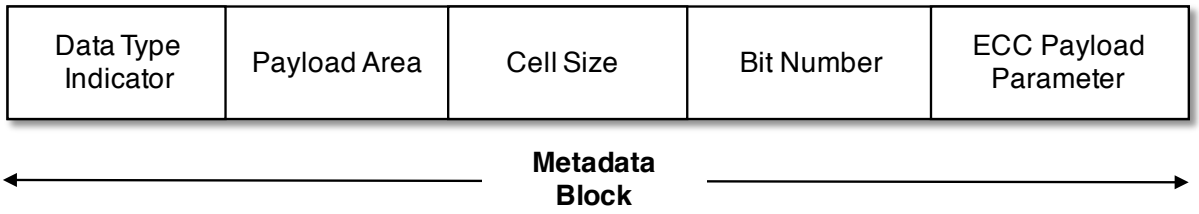

Figure 3.4: Format of covert frames' metadata block.

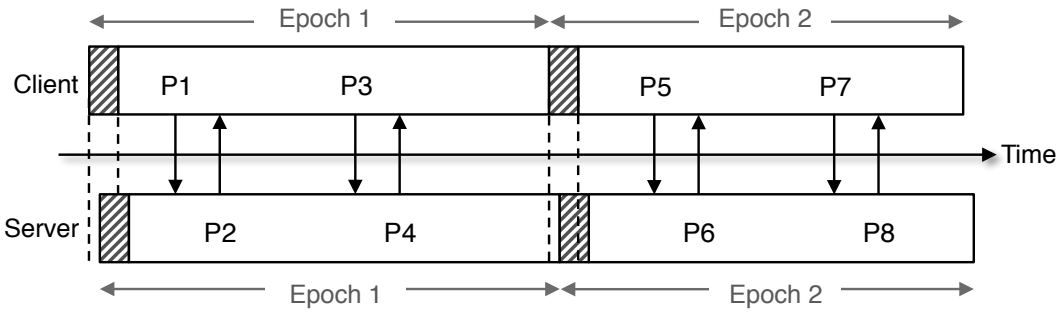

Figure 3.5: Epochs in a bidirectional covert channel. IP packets are exchanged between client and server applications during the data transmission phases.

in Figure 3.4, uses a conservative fixed encoding scheme so as to be resilient to decoding errors and prevent observability anomalies.

4. Readjusting the frame encoding parameters: Since the network conditions may change during a communication session, DeltaShaper allows for the readjustment of the encoding parameters. Essentially, this is achieved by enabling the client to issue repetitions of the calibration process to determine new parameters and then reuse such parameters to encode the ensuing data frames. Each period comprising a calibration phase and a data transmission phase is called an *epoch*. Each session can comprise multiple epochs. The time span dedicated to the calibration phase and to the data transmission phase are defined by configuration. To support epochs, the metadata header includes a *data type* block that indicates whether the frame carries dummy data, payload data or is used for calibration purposes only. Figure 3.5 illustrates the concept of epochs; it depicts a timeline diagram that represents a two-way communication channel established by DeltaShaper. There are two independent communication flows, used to send IP packets in both directions: from client to server (C→S) and from server to client (S→C).

Summary

This chapter has detailed the design of DeltaShaper, a novel Internet censorship circumvention system that leverages the video channel of existing video-conferencing applications to tunnel covert data. The design of DeltaShaper's encoding scheme enables for the creation of an interactive covert channel which can be used to transmit arbitrary data between the endpoints. One of the primary concerns on the system's design is tied to its flexibility, allowing DeltaShaper to adapt in the occurrence of changing network conditions that may or be not caused by a censor. This offers an extra degree of resilience against traffic analysis and active attacks perpetrated by a censor. The next chapter thoroughly describes the implementation of the system's prototype.

4 Implementation

This chapter addresses the implementation details of DeltaShaper's prototype. Section 4.1 presents an overview over the implementation of DeltaShaper's upstream / downstream pipelines. Section 4.2 details the system's setup and operation procedures. The format of the covert channel's messages is specified in Section 4.3. The mechanisms that were employed to implement error recovery are described in Section 4.4. Network layer packets' fragmentation and reassembly mechanisms are addressed in Section 4.5. Section 4.6 addresses the management of DeltaShaper's covert channel. Lastly, Section 4.7 details the encoding selector algorithm leveraged by DeltaShaper to dynamically determine covert data encoding parameters.

4.1 Implementation Overview

We have implemented a DeltaShaper prototype for Linux, as represented in Figure 4.1. The prototype comprises several components that implement the client and server pipelines of DeltaShaper (see Figure 3.1). Some components were built from scratch in C++ and Python; others are based on existing tools. To build DeltaShaper, several challenges were faced at different levels: network interfacing, video processing, and Skype interfacing.

4.1.1 Network Interfacing

The network interfacing between DeltaShaper and the client / server application must be performed without changes to the application. For this reason, complementary techniques are adopted on the client and server endpoints. In particular, Linux's network namespaces offer separate instances of network interfaces and routing tables that operate independent of each other. A client application executing inside a network namespace can have its traffic transparently forwarded by DeltaShaper.

The prototype takes advantage of both Linux's network namespaces and the netfilter packet filtering framework (Netfilter 1998) in order to build the backend of DeltaShaper's network packets tunneling mechanism. Netfilter allows for selected packets to be analyzed and modified by a user-space program, which can then re-inject, drop or steal a packet from the current network stack.

Each endpoint instantiates a pair of Virtual Ethernet (VETH) interfaces which allow the communication of a network namespace with the "global" namespace where physical interfaces exist. The outgoing IP packets of a client application running inside DeltaShaper's network namespace are intercepted by a kernel module which inspects the traffic flowing through the outer VETH interface (VETH0), using netfilter. Therefore, the packets originating in the inner VETH interface (VETH1) are handled to a user-space program which encodes and transmits them over Skype.

At the server side, IP packets are decoded and routed to the "localhost" interface to be delivered transparently to the server application. This last-mile routing is performed through raw sockets, which

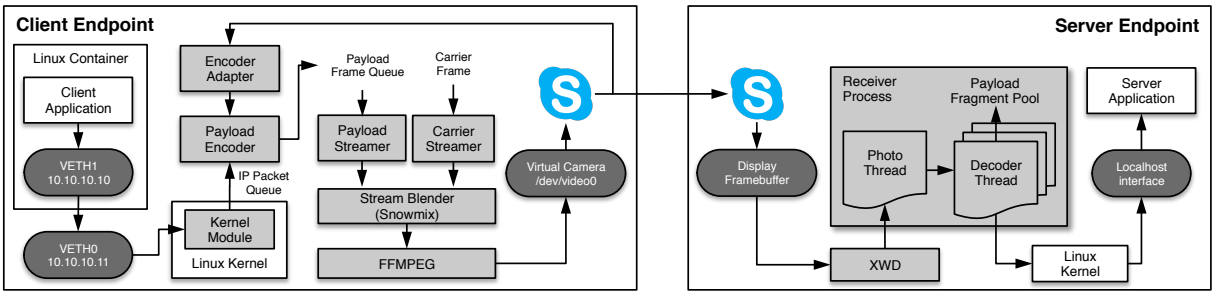

Figure 4.1: Components of the DeltaShaper prototype (shaded boxes).

enables the delivery of the IP packets exactly as they were received, without modifying the information contained in their headers.

Similarly to what is performed at the client side, the server side has a Linux network namespace with a pair of VETH interfaces, which IPs match those of the interfaces instantiated by the client. The answer from the server application is then directed at the local VETH interface with the same IP address as the request-issuing VETH interface on the client side. A kernel module inspects the traffic flowing to this VETH interface (VETH1') by intercepting the traffic in the VETH interface that bridges the communication to the network namespace (VETH0'). These packets are handled to a user-space program which will encode and transmit them via Skype back to the client.

4.1.2 Video Processing

A base requirement to build DeltaShaper consists in the ability to combine the synthesized payload frames with typical webcam videos. Snowmix (Maersk-Moller 2012), a live stream video mixer, is leveraged for producing the covert video frames that will be ultimately delivered to Skype. Snowmix allows for the composition of different video feeds, enabling the overlay of the payload video on top of a dummy carrier video. Internally, Snowmix enables the attachment of GStreamer (GStreamer 2001) video pipelines to input data into the different feeds. GStreamer is a pipeline-based multimedia framework which allows for media processing through the combination of several separate processing elements.

The dummy carrier video, selected as background video as per Snowmix's initial configuration, is fed to Snowmix through a GStreamer pipeline. For efficient video synthesis at the client side, DeltaShaper's payload encoder takes each IP packet, generates the corresponding payload frame, and forwards this frame to the payload streamer. The payload streamer is a user-level process that feeds payload frames into a GStreamer pipeline which output is then overlayed over the background feed. When there are no new payload frames waiting to be delivered, the payload streamer loads frames filled with pseudo-randomly generated dummy data.

The output of both Gstreamer pipelines, as well as the global output offered by Snowmix, is set to 30 frames-per-second. Although the payload streamer process is programmatically set to output 30 frames-per-second, the GStreamer pipeline responsible for outputting the carrier video may drop or duplicate frames in order to match the 30 frames-per-second restriction, depending on the original video framerate. Snowmix is configured to output a MJPEG video stream to a named pipe, where each frame consists in an individual high-quality JPEG image.

4.1.3 Skype Interfacing

Lastly, it is necessary to interface with Skype without modifying the Skype client software. For this purpose, the covert video is routed to a virtual v4l2loopback (v4l2loopback 2005) camera device. Skype's input source is then configured to read from the virtual camera. This video routing operation is directly supported by Snowmix when coupled with the FFMPEG (FFmpeg 2000) video encoding tool. Here, FFMPEG will simply read the output of Snowmix from a named pipe and convert the stream's frames to the YUV colorspace. As described in Section 3.3.2.1, Skype's H.264 video codec requires that input frames are delivered in this format.

Skype4Py (skype4py 2007) is used in order to start DeltaShaper's client and server components according to the ongoing call status. Skype4Py is a software library which allows for the programmable control of Skype client applications. More concretely, DeltaShaper's client component is activated before a call takes place, setting up all the necessary video pipelines. In its turn, the server component is activated as soon as a call is established so that each endpoint can inspect the video content being rendered on its virtual display.

At the receiver's endpoint, DeltaShaper relies on a thread of the receiver process to periodically launch the XWD tool to obtain a screenshot of the Skype call projected in the virtual display. This thread is calibrated to ensure that the polling frequency is higher than the payload transmission rate. This condition ensures that no covert frames are lost due to late screen captures. The same receiver process manages a pool of worker threads which extract the payload block out of the gathered covert frames, and routes the resulting IP packets to the Linux kernel.

4.2 System Setup and Operation

To initialize the system and establish a covert communication channel, DeltaShaper must be launched at both channel endpoints so that it can instantiate the required components for both downstream and upstream pipelines. Once the channel has been established, a server application listening at one channel endpoint can be contacted by a client application running on the counterpart endpoint. As described before, communication takes place over standard TCP/IP sockets without the need to modify the client-server application.

For an efficient and correct transmission of IP packets, DeltaShaper implements error recovery and packet fragmentation mechanisms. Firstly, IP packets are encapsulated into payload blocks so that these can be encoded into payload frames and exchanged with the receiver. Since Skype provides ordered frame delivery, DeltaShaper system does not exhibit the increased overhead of keeping track of the delivery of out of order covert frames and re-arrange them. However, covert frames and their associated payload blocks may be lost due to network breakups. To keep the message protocol simple, the IP packets transmitted in such frames are assumed to be lost, leaving it up to TCP to request the corresponding retransmission.

Internally, DeltaShaper manages the internal state of the covert channel, assuring that the communicating parties are synchronized so that IP packets can be exchanged on overlapping data transmission phases only. The management of the covert channel's state, including calibration and data transmission phases is further detailed in Section 4.6.

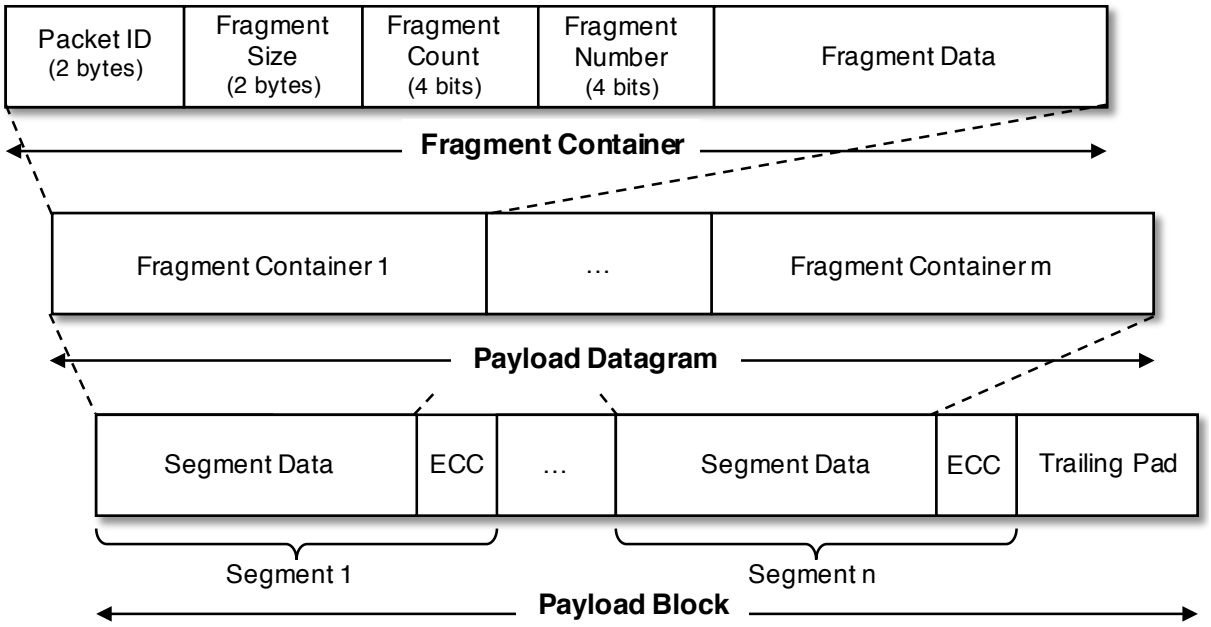

Figure 4.2: Format of DeltaShaper messages.

4.3 Message Format

To support error correction and packet fragmentation, a simple message format protocol is specified. Essentially, this protocol maps messages between the high-level IP layer (IP packets) and the low-level frame layer supported by DeltaShaper (payload blocks). In designing this protocol, an additional concern was to improve the efficiency of the channel by reducing the amount of bits allocated to metadata.

Figure 4.2 represents the format of DeltaShaper messages, highlighting three internal abstraction layers. At the bottommost layer, it is possible to find the *payload block*, whose bits are directly encoded into cells of a payload frame. The layout of the payload block consists of a body divided into segments which are designed to support bit error recovery, resorting to Error-Correcting Codes (ECC). Each segment contains a chunk of application data followed by redundancy bits (see Section 4.4). To support multiple ECC schemes, the metadata block contains a header which indicates the number of a segment's bits reserved to redundancy data.

The intermediate abstraction layer results from extracting the payload block from error redundancy metadata and concatenating the resulting data segments into a single byte sequence named *payload datagram*. This data structure contains fragments of IP packets. There can be multiple fragments contained in a payload datagram.

The uppermost abstraction layer specifies the data structure *fragment container* which represents an individual fragment of an IP packet. Fragment containers are included in the body of payload datagrams. The body of each fragment container contains IP packet data, whereas the header includes four fields: a packet ID, fragment size, fragment count, and fragment number. These fields enable the recipient party to reconstruct a sequence of received fragments into complete IP packets, as explained below in Section 4.5.

4.4 Error Recovery

In DeltaShaper, payload blocks are encoded into covert frames by mapping the bits of each payload block to the pixel colors of a payload frame. The receiver can reconstruct the original payload block by decoding the color code of each cell and chaining all decoded color codes into a bit sequence that will constitute the reconstructed payload block. However, since the color of the received frame may have been altered by Skype's video compression algorithm, the recovered payload block may include bit errors. Since discarding entire frames in the presence of errors would considerably penalize throughput, error-correcting codes are employed.

A general payload block layout that supports configurable error-correcting codes has been defined. The current prototype adopts the Reed-Solomon (Wicker 1994) ECC as a result of an empirical evaluation. The prototype uses a commonly used code denoted as $(n, k) = (255, 223)$, where n corresponds to 255 bytes of data symbols, out of which $k = 223$ bytes consist of application data and the remaining 32 bytes encode parity bit symbols. This code can correct up to 16 symbol errors (16 different bytes) per symbol block. By adopting the Reed-Solomon code $(255, 223)$, the total segment size of a payload block is limited to 255 bytes, out of which 32 bytes form the ECC (see Figure 4.2).

DeltaShaper allows for the dynamic adaptation of k, which can range from 1 to 254. This is particularly useful for encoding schemes which error threshold surpasses the error-correction capability provided by the default $(255, 223)$ configuration. In order to announce to the receiving endpoint which ECC parameter should be used for the decoding of a given payload block, the payload frames' metadata block has a dedicated field, which represents the value of k to be used. This parameter, introduced in Section 3.3.4, is named ECC Payload Parameter and can be observed in Figure 3.4.

4.5 Packet Fragmentation and Reassembly

Packet fragmentation and reassembly mechanisms are required in order to efficiently make use of the covert channel implemented by DeltaShaper. This is a consequence of its encoding technique, in which application data must be transmitted in individual frames by chunks that are bound by the payload block size. IP packets that exceed the maximum size of a given segment of a payload block must be broken up and transmitted along multiple segments. Conversely, multiple small IP packets can be transmitted on a single payload frame. If the size of the current IP packet exceeds the capacity of the available frame space, it needs to be fragmented and transmitted in multiple payload frames. To address the aforementioned requirements, DeltaShaper implements the following algorithms:

1. Packet fragmentation: At the sender side, DeltaShaper's payload encoder is essentially waiting for incoming packets sent by the local client application. Instead of following their regular path through the network stack, these packets are placed into a user-space accessible queue. The payload encoder reads the packets from this queue, wraps them around payload containers, and forms a payload block until its maximum capacity has been reached. The data type indicator of the header is set to indicate that this frame carries payload application data and the fragment count is initialized to hold the total number of fragment containers included in the body of the payload block. If a pending IP packet can fit entirely in the available free space of the payload block's body, the payload encoder can wrap the entire packet around a fragment container and include it in the payload block. Otherwise, the IP packet must be fragmented into as many pieces as necessary until all fragments are sent to the receiver endpoint.

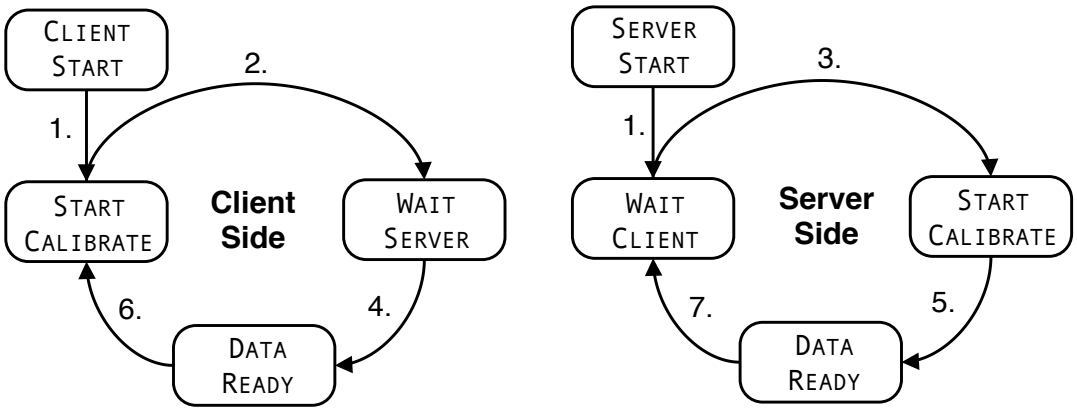

Figure 4.3: State machines of DeltaShaper endpoints.

2. Packet reassembly: Upon receiving a payload frame, the receiver process starts by interpreting the data type indicator in the metadata header to determine whether the frame carries useful data. If there is actual data to be decoded, the process will recover the existing data fragments from the payload frame's body. The header of each fragment container enables for the fully recover of embedded IP packets. The packet ID allows to determine the packet that each fragment belongs to. This identifier is taken from a counter that is maintained by the client and incremented for every IP packet sent by the client for that particular DeltaShaper channel. If a packet is fully enclosed in a given fragment container, the fragment count is one and the packet size can be read from the fragment size field. If the packet is fragmented, the fragment count is greater than one, matching the number of fragments needed to encode the packet. In order to reassemble the entire packet, the receiver process maintains a local fragment pool and waits until all fragments have been received in order to reorder the fragments and reassemble them. For improved throughput, the payload encoder can add multiple IP packets into each single frame until there is no more space left on the frame. However, in case no more IP packets exist in the queue, the frame is sent, rather than waiting for more packets, in order to reduce packet delivery latency.

In order to simplify the message protocol, DeltaShaper does not keep track or asks for the retransmission of lost frames. In the event that a given payload frame is lost, the responsibility of retransmission of pending IP packets is delegated to TCP. The receiver has a limited amount of buffering space which gets released on a FIFO policy: if some packet fragments have been lost, pending fragments in the buffer will be evicted by most recent ones. This is possible since DeltaShaper provides an ordered packet delivery. Due to the existence of packet IDs, the system is able to assess whether there are pending packets, further discarding them.

4.6 Channel State Management

Packet transmission is only allowed to occur during the data transmission phase of a given epoch. Communicating parties manage the internal state of the covert channel in order to implement state transition between calibration and data transmission phases and restrict the delivery of packets to data transmission phases. Moreover, since there can be several of such alternating phases associated with multiple epochs, some additional effort is required to keep both parties mutually synchronized. For this reason, client and server endpoints manage the covert channel according to the state transition diagram shown in Figure 4.3. For signaling state transitions, the client relies on the data type indicator present on the metadata block of outgoing covert frames.

Essentially, DeltaShaper's client and server coordinate each other to maintain the channel state according to a master-slave pattern. They alternate between calibration and data transmission phases, but the client takes an active role in triggering these transitions (master), being followed by the server (slave). The covert channel will be closed upon the termination of the Skype call or by an explicit termination action issued by an endpoint.

1. Calibration phases: When the system is initialized, the client starts a first epoch by entering calibration phase. The client changes to `STARTCALIBRATE` state and sends covert frames with the data type indicator set to "calibrating". In this state, the content found in the body of payload frames is simply random data. By carrying random data, calibration frames are expected to better simulate the transmission of arbitrary binary data and stress the compression capability of the video encoding algorithm. In contrast, the server initializes by waiting for client calibration frames (in state `WAITCLIENT`) before it begins its own calibration procedure (entering state `STARTCALIBRATE`). The "calibrating" data type indicator allows the server to transit to state `STARTCALIBRATE` and to immediately discard incoming payload frames, as these are issued for calibration purposes only. Eventually, the client starts receiving calibration frames from the server. Similarly to the server, the client discards such frames, since they are not being used to transmit any useful data. At this point, both the client and the server continue to send calibration frames up until a predefined calibration time which allows both endpoints to collect sufficient samples for determining the ideal payload frame parameters. Section 5.7 introduces further details on the choice of this calibration time, which is set to 30 seconds in the current implementation.

2. Data transmission phases: Finally, when the calibration timeout expires, both the client and server change their state to `DATAREADY` and start sending covert frames with the data type indicator set to "dummy" (along with random data on the payload block). Similarly to the action performed upon the reception of a calibration frame, both endpoints can discard the current payload frame without decoding the dummy data. The client sets another timeout – data timeout – corresponding to the maximum duration of the data transmission phase. Once the timeout elapses, the client side may trigger another calibration event to start a new epoch. When the calibration phase is over, both parties can exchange actual payload data. Whenever there is new data to send, the corresponding payload frames' data type indicator is set to "payload". Unlike "calibration" and "dummy" frames, "payload" frames' content is extracted upon reception by the counterpart endpoint.

4.7 Encoding Selector Algorithm

Finally, the network adaptation algorithm, implemented by each party during the calibration phase, is discussed. This algorithm is fundamental to determine the ideal parameters for payload frame encoding which preserve unobservability under the current network conditions.

Two alternative adaptive encoding selector algorithms, each with their pros and cons, can be envisioned. Both alternatives involve the generation of an offline reference table, which keeps data about regular reference streams' sets. Since only the censor will eventually possess packet traces that capture a global signature of regular streams within its borders, DeltaShaper users may build a personalized regular reference streams sets from their own past legitimate video-conferencing calls.

By building this reference table with packet traces obtained in different network conditions, the table can also be used to assess the network conditions in which a DeltaShaper connection takes place. To this end, the similarity between a carrier video and each of the different reference sets is computed. Furthermore, as the characteristics of reference streams sets change due to network perturbations, so

Algorithm 1 Adaptive encoding selector algorithm

procedure ENCODINGSELECTOR(s, s_p, T_r)
  $s_r, s_p, \Delta_{min} \leftarrow null, null, \infty$
  for all r in reference streams sets T_r **do**
    $N \leftarrow r.nStreams$
    $\Delta_r \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} \text{EMD}(f_l(s), f_l(r.dist(i)))$
    if $\Delta_r < \Delta_{min}$ **then**
      $s_r, \Delta_{min} \leftarrow r, \Delta_r$

  $\Delta \leftarrow s_r.threshold$
  if $\Delta_{min} > \Delta$ **then**
    return $null$

  for all p in encoding parameters samples S_p **do**
    $N \leftarrow s_r.nStreams$
    $\Delta_p \leftarrow \frac{1}{N} \sum_{i=0}^{N-1} \text{EMD}(f_l(p), f_l(s_r.dist(i)))$
    if $\Delta_p < \Delta$ **then**
      $s_p \leftarrow s_p + p$

  if s_p is empty **then**
    return $null$
  return sp

does the associated Δ similarity threshold. For this reason, the reference table includes the Δ similarity threshold for each reference set.

The first alternative is based on an offline approximation approach where, apart from the Δ threshold, the reference table also stores possible encoding parameters for each reference stream set. The algorithm starts by playing the carrier video, intercepting packets of the corresponding Skype stream. For each packet, it stores relevant information about the packet in a local database, namely the packet sizes. Then, it computes the average similarity between the stored stream and every reference stream set present in the reference table. The reference set is chosen by checking which similarity value obtained in the previous step is closest to a candidate reference stream set, providing that the similarity value does not surpasses the Δ threshold for that given reference set. The next step is to retrieve hard-coded values for encoding parameters which are known to provide unobservability when coupled with pre-established carrier videos.

In short, this adaptive encoding selector algorithm allows for a quick calibration phase, since it just needs to transmit the carrier video once. From the moment the reference stream set is chosen, encoding parameters are directly retrieved from the reference table and the system may issue payload frames using such parameters. However, it also has some drawbacks. The reference table on which this algorithm depends involves the long process of gathering traffic samples of different carrier videos, for different network conditions, with the overhead of gathering traffic samples of covert videos with an exploding number of encoding schemes. Another problem with this approach is that it blindly trusts the encoding selector retrieved from the reference table without further verification.

A second alternative that departs from the blind use of possible non-ideal hard-coded parameters actively checks the similarity of a covert video with its reference stream set. Algorithm 1 provides a sketch of this algorithm. It starts with parameters set to null. When the channel state enters the calibration phase, the algorithm starts playing the carrier video (s) and intercepting packets of the correspond-

ing Skype stream. For each packet, it stores relevant information about the packet in a local database, namely the packet size. When the calibration phase ends, the similarity metric (EMD) between the obtained sample and a set of reference streams (r) for different network conditions is calculated. Each of these computations yields a Δ_r value. The algorithm then chooses the reference stream set that yielded the smallest Δ_r to be the reference set (s_r) for the remaining calibration operations. Shall the algorithm be unable to identify the carrier video as a regular stream, the user is recommended to abort the transmission (Line 11).

After transmitting the carrier video for a given period, DeltaShaper starts overlaying payload frames with different encoding parameters (p), also intercepting packets of the corresponding Skype stream. The next step is to compute the similarity of each sample to the previously fixed reference set, obtaining Δ_p. The algorithm gathers the set of encoding parameters which yielded a Δ_p smaller than the threshold Δ, for the given reference set. These (s_p) correspond to several alternatives which are mostly similar to the reference set, and that provide unobservability. If this set is empty, it means that none of the sampled encoding schemes is considered safe and the system recommends the transmission's abortion (Line 20). Otherwise, the communication can be safely undertaken while preserving unobservability. As a result, the algorithm sets the frame encoding parameters from the recommended parameters for the selected reference set s_r (Line 21). If the identified set offers more than one alternative, DeltaShaper shall choose the one which enables for a greater throughput. When transmitting data, these parameters will be adopted by the payload encoding scheme (see Section 3.3.2).

As a side-effect, this adaptation scheme allows for the client to choose a new input carrier video (eventually his own recorded or live webcam videos), and quickly compute parameters that may allow that video to be used as covert data carrier. Moreover, it may be the case that users come up with particular videos which, due to their intrinsic characteristics, allow for the encoding of larger amounts of covert data while maintaining unobservability.

The current implementation of DeltaShaper employs this adaptation scheme in its calibration phase. Additionally, although the current implementation uses EMD and packet sizes to establish Δ thresholds, the encoder selection algorithm can be easily extended to employ different similarity functions and traffic features. This may prove to be an advantage as the system will be able to further adapt by taking into account novel traffic analysis techniques devised in the nearby future.

Summary

This chapter has described the implementation of the DeltaShaper prototype. The specification of the system's message format and network layer packets' fragmentation and reassembly mechanisms has been presented. Together, these allow the current implementation to tunnel TCP/IP packets between the video-conferencing call endpoints. The prototype's implementation reveals a mechanism for managing the covert channel's state, allowing the system to repeatedly calibrate and adapt to network contingencies. As part of its calibration procedure, DeltaShaper's encoder selector algorithm offers an extensible way to readjust the system with different similarity functions and traffic features. The next chapter presents a comprehensive experimental evaluation of the prototype.

5 Evaluation

This chapter describes the experiments that were carried out in order to evaluate the proposed solution. It starts by detailing the experimental settings used in the evaluation of the system (Section 5.1). Then, it describes and discusses the outcomes of a series of experiments that aim to test the effectiveness of EMD and Δ threshold metrics in characterizing Skype streams (Section 5.2). Section 5.3 assesses DeltaShaper's ability to generate unobservable covert Skype channels based on such metrics. Section 5.4 details the measurements of the performance of DeltaShaper channels. Later, this chapter addresses DeltaShaper's unobservability resilience against controlled network perturbations (Section 5.5) and the study of alternative traffic features on the classification of Skype streams (Section 5.6). Section 5.7 introduces a study over the calibration period with which DeltaShaper's adaptation mechanism may be configured. The evaluation of the end-user experience of DeltaShaper for several use cases, comparing its use against overt communication channels, is detailed in Section 5.8. Lastly, this chapter discusses several security considerations of the system, in light of the major categories of attacks that a censor may conduct over DeltaShaper (Section 5.9).

5.1 Experimental Settings

The system's experimental evaluation was performed on two 32bit Ubuntu 14.04.4 LTS virtual machines (VMs) with 8GB RAM and 4 virtual Intel Core 2 Duo T7700 2.40GHz CPUs. Each VM runs an instance of Skype v4.3.0.37 and DeltaShaper acting, respectively, as caller and callee of video-conferencing calls. The native *netem* Linux network emulation functionality was used to enforce limitations over the network conditions between the VMs. More specifically, *netem* was configured in the caller machine for the experiments described in Section 5.5.

In order to characterize regular video-conferencing streams, 30 videos representative of actual videocalls have been selected. These videos are used as training set for regular streams. Such videos generally exhibit low movement as users typically sit in front of a computer, moving sparingly as they speak. These videos have not been edited and are free of watermarks or other visual artifacts. The training set for irregular streams consists of 30 YouTube videos, involving more dynamic movement, where both rapidly changing scenes and artifacts introduced by video editing software are common. The duration of each video sample is 30 seconds. Samples are captured after 10 seconds of the initial call establishment in the machine acting as caller. For these experiments, the calls' audio packets carry data representing silence.

The following experiments are based on a single traffic feature, namely the packet sizes. Packet lengths are widely used for feature extraction in the related literature. Notwithstanding, the obtained results are further discussed considering different traffic features, such as inter-packet timing, in Section 5.6. The computation of the EMD cost between the packet size frequency distribution of different streams assumes that packet lengths are discretized into equal partitions (bins) of 50 units ($K = 50$).

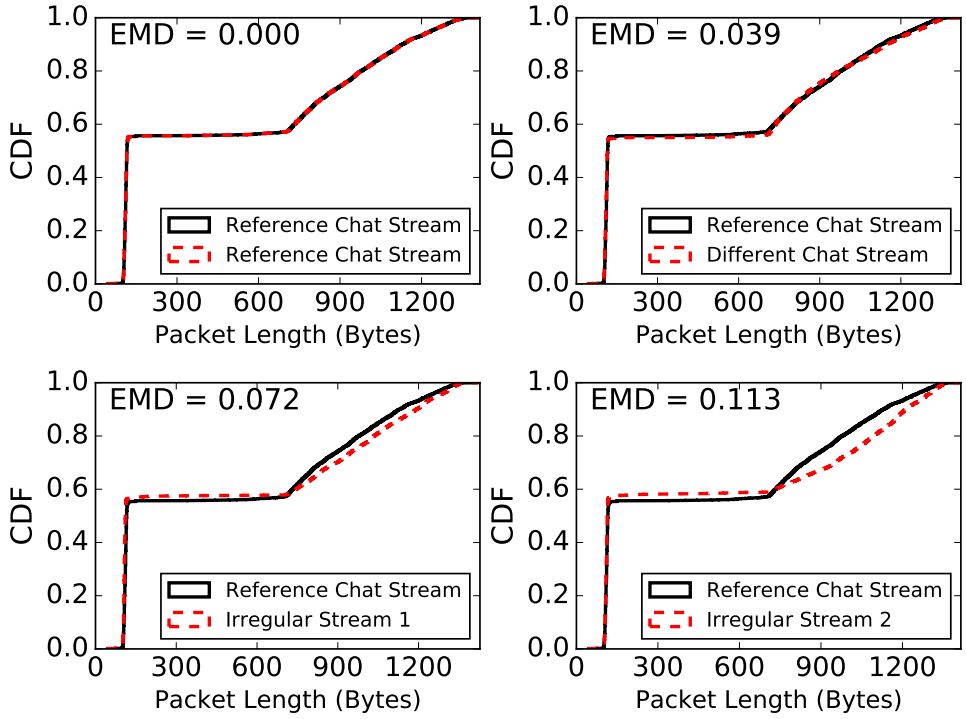

Figure 5.1: Packet length CDF of sample streams.

Experimentally, this bin size was found to create more accurate signatures for regular / irregular streams than bigger-sized bins, whereas smaller bins failed to accurately represent the differences between streams. This phenomenon occurs due to the “curse of dimensionality”, where in a dimensional feature space with many possible states, increasing amounts of data are needed to ensure that several samples exist with each possible combination of values to accurately build a classification signature.

5.2 Characterization of Skype Streams

Firstly, it is studied whether Skype calls exhibit measurable patterns that allow the differentiation between regular and irregular calls. This question is of utmost importance since such patterns may be used by a censor to detect suspicious videocalls (i.e., irregular ones) and further block them. The data in Figure 5.1 indicates that such patterns do exist. It shows the cumulative distribution function (CDF) of packet lengths for four Skype test streams, respectively represented in a different plot: (a) the stream of an actual videocall which is taken as reference stream; (b) a stream of a regular call from a different user; and two irregular streams corresponding to (c) a football match and, (d) a music concert. Each plot represents the distribution of the test stream along with the packet length distribution of the reference stream (the black curve), exhibiting their similarity by calculating the respective EMD value. It can be seen that the EMD increases progressively, reaching 0.113 for the most dynamic video, i.e., the music concert stream. The main differences can be observed for 40% of packets, which correspond to the largest packets (above 745 bytes) transmitted. This is congruent with VBR encoding procedures, where more dynamic scenes typically lead to the generation of larger network packets due to the higher amount of inter- and intra-frame differences. In particular, this comparison shows that these differences are more acute in dynamic videos than in rather static videocalls. For conducting the experiment with reduced interference, the four test streams have been transmitted in a one-way Skype video-conferencing call.

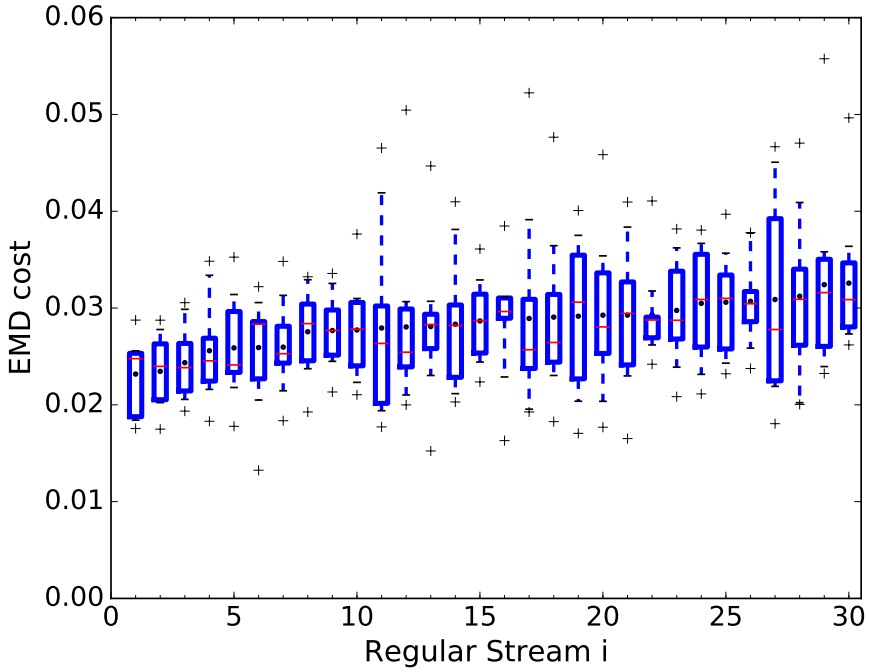

Figure 5.2: EMD cost of multiple videocall streams.

To better understand whether these traffic patterns are stable, and therefore can be reliably used for characterizing regular Skype streams, a study must be conducted to assess whether there are significant differences in the packet length distribution when streaming the same videocall multiple times over Skype. Each regular video call sample in the dataset is replayed 10 times. The EMD of each resulting stream is calculated, taking as baseline the average distribution of all 10 runs. These experiments were performed through one-way video-conferencing calls taking place in unconstrained network conditions. Figure 5.2 plots the most relevant statistical indicators for the resulting EMD values of each video: min, max, mean, and percentiles 5, 25, 50, 75, and 95. On the one hand, packet length distributions of the same video tend to be quite similar. This is attested by the fact that the largest difference observed between 25th and 75th percentile of a single video is only 0.02. Moreover, the average EMD value tends to be very similar among different videos, varying between 0.025 and 0.031. It is possible to conclude that, under the same network conditions, regular Skype streams display a high degree of similarity.

The next step is to study whether a censor can differentiate regular from irregular streams by computing the similarity of packet length distributions. To that end, a regular stream is used as reference stream to calculate the EMD cost of other video streams. These video streams were generated by running each video of the data set 10 times and calculating statistical indicators of the resulting EMD cost. Similarly to the previous experiments, the traffic samples were obtained from one-way video-conferencing calls, under unconstrained network conditions. Figure 5.3 (a) shows the results obtained, plotting on the left hand side the EMD cost for regular streams, and on the right hand side the EMD cost for irregular streams. It is possible to immediately observe a pattern in which regular streams tend to result in a constantly low EMD cost (below 0.1), whereas irregular streams produce a significantly more scattered pattern varying EMD cost approximately from as low as 0.025 to as high as 0.25, i.e., by an order of magnitude.

The question is then whether it is possible to define an EMD cost Δ threshold that can be used as stream classifier such that a stream s is considered regular if EMD$(s_R, s) < \Delta$ or irregular otherwise (see

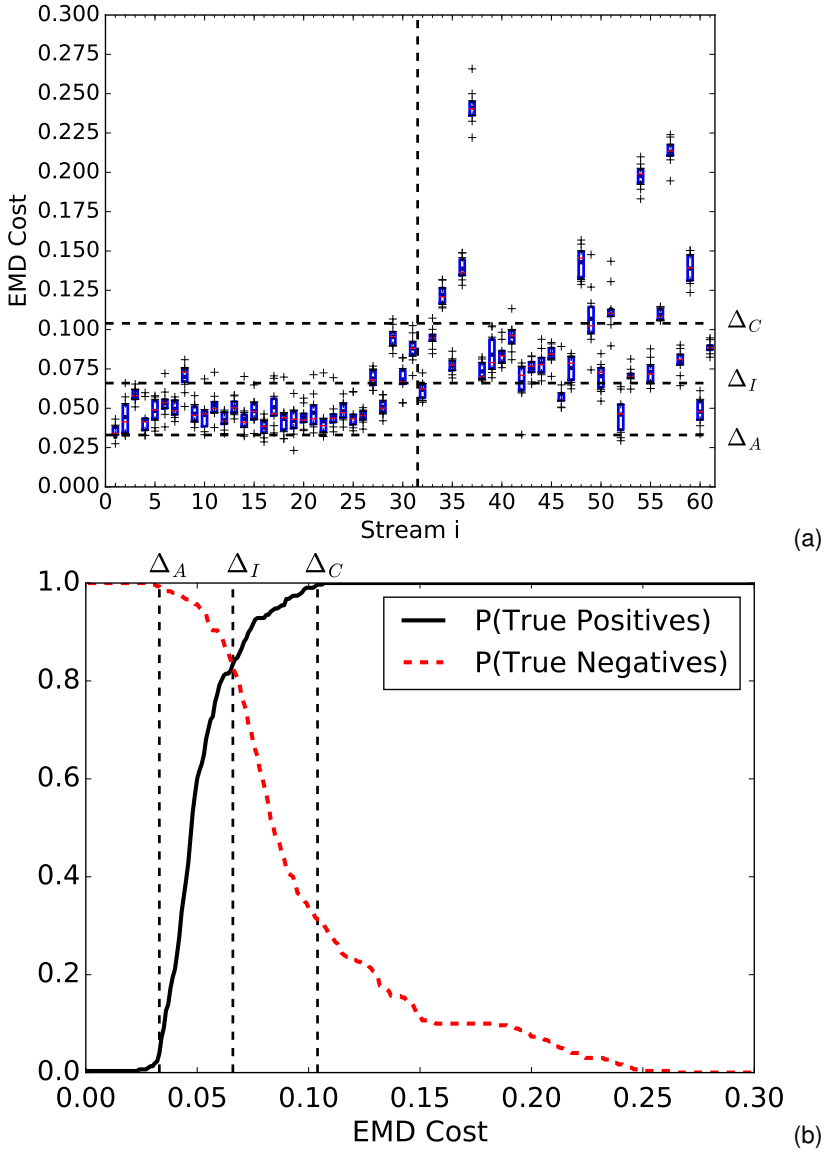

Figure 5.3: EMD based on reference stream.

Section 3.3.3). For this particular experiment, Δ can be set to any value between 0 and the maximum observed EMD cost (0.275). To evaluate the effectiveness of this classifier, Figure 5.3 (b) shows the probability of true positives (sensitivity) and true negatives (specificity) as Δ varies (in the x-axis). It is possible to see that, as Δ increases the number of true negatives starts at 1, meaning that all irregular streams are correctly identified by the classifier, but eventually starts decreasing at 0.025 (Δ_A) because some irregular streams start being classified as regular. In contrast, the true positive rate curve begins in 0 and starts increasing when the EMD cost of some regular streams becomes lower than Δ. Eventually, when Δ surpasses 0.1 (Δ_C), the classifier is able to correctly identify all regular streams.

Based on how the Δ threshold is set, several classification policies are possible. Suppose that a censor wishes to apply an *aggressive classification policy* by blocking all streams that are truly irregular. In this case, Δ must be set to Δ_A, which is the point where the true negative rate starts falling below 100%. The downside of this policy, however, is that a large number of regular streams would also be blocked, more specifically 95% of regular streams (false negatives) causing a massive denial of service of legitimate Skype users. On the other hand, if the censor aims to prevent blocking of any regular Skype

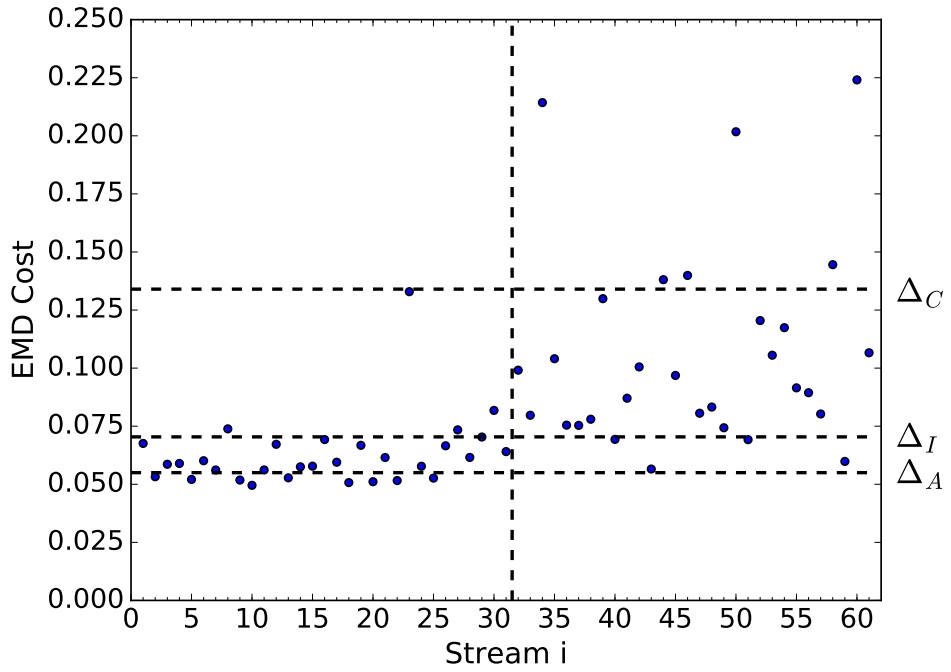

Figure 5.4: Average EMD with respect to the regular reference streams dataset.

transmissions (i.e., a *conservative classification policy*), Δ must be set to Δ_C. The negative side-effect of this policy is, however, a loss in specificity since approximately 70% of irregular streams would also be classified as regular (false positives). An intermediate possibility that maximizes the classifier's accuracy is to take the cutoff point where the probability of true negatives equals the probability of true positives. For the classifier, this point corresponds to EMD cost 0.066 (Δ_I) which means that setting Δ to this value results in 83% accuracy in classifying a stream. Thus, it is possible to define a Δ threshold which allows for identification of regular streams with high probability. This is crucial as DeltaShaper explores this property to hide within regular streams.

It is reasonable to wonder whether a single reference stream is able to provide an accurate baseline for establishing Δ thresholds. Even though Figure 5.2 shows that the majority of regular streams are close to the chosen reference stream, regular streams may exhibit underlying differences among themselves that are not captured by this simple comparison method. The assumption that the packet length distribution can be used to effectively identify regular streams is then strengthened by a second test. In this test, every single stream in both datasets is compared to each and every stream pertaining to the regular streams dataset. Such comparison allows for the computation of a value which indicates, in average, how much does a stream differ from the whole regular dataset itself. This test is currently implemented as part of DeltaShaper's adaptation mechanism which was already described in Section 3.3.3. To perform this experiment, one sample is randomly selected from each stream. Then, its EMD value against every regular stream is computed. Finally, its average is plotted. The results of this experiment, depicted in Figure 5.4, show that it is still possible to define accurate Δ thresholds to distinguish regular from irregular streams. Notably, the classifier is able to achieve 88% accuracy when setting Δ_I as threshold. This outcome suggests that a censor is able to better distinguish Skype streams by employing a more robust test which takes into account the intrinsic differences between regular streams.

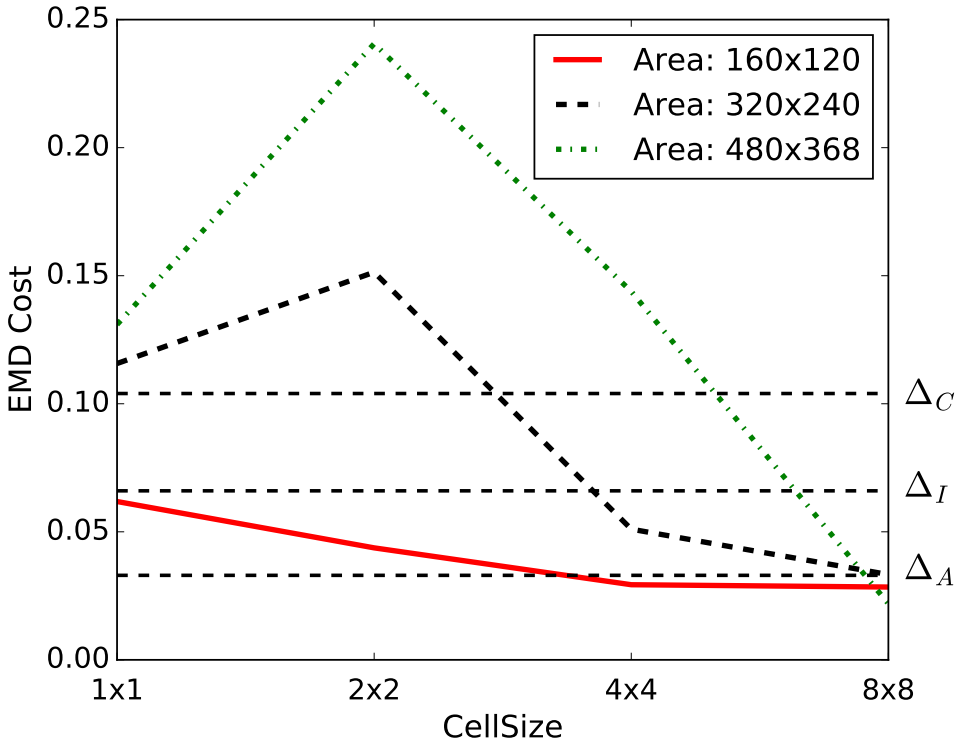

Figure 5.5: EMD cost changing area and cell size.

5.3 Unobservability of DeltaShaper Channels

As stated previously, Skype streams exhibit specific packet length patterns that allow a censor to distinguish regular from irregular streams based on an EMD classifier. Consequently, in order to produce covert Skype streams that can be deemed indistinguishable from a regular stream, DeltaShaper must be set up such that the EMD cost of resulting stream remains below the Δ threshold. Since the properties of a resulting stream depend on the encoding parameters provided to DeltaShaper, it is fundamental to study which range of encoding parameters can be reliably used to produce unobservable covert streams.

As listed in Table 3.1, DeltaShaper can be configured with four parameters: payload area size, cell size, bit number, and frame rate. Since covering the entire configuration space requires covering a large number of configurations, this work focuses on a subset of parameters that result in valid configurations, but not necessarily optimal in terms of the maximum throughput that can be achieved. In this study, the reference stream that was selected in the previous section, as well as the Δ threshold values that were found for the same reference stream, are used. For this test, several payload frames are synthesized and combined with the carrier video that originally generated the reference stream. These “offline” samples are then transmitted over Skype, allowing the gathering of samples from the resulting network streams.

The analysis of the combined effects of the payload area size and the cell size starts by fixing the bit number in 1 bit/cell and the frame rate in 1 frame-per-second. Figure 5.5 shows the EMD cost for several configurations varying the cell size between 1x1 and 8x8 pixels and the area size ranging from 160x120, 320x240, and 480x368. The area sizes were chosen to cover roughly 1/16, 1/4, and 1/2 of the frame size, respectively, while aligning the payload size to the size of a macroblock. As described in Section 3.3.2.1, frames are split into small matrices of pixels, named macroblocks, which are used as part of the video compression algorithm. The plot is annotated (in dashed lines) with Δ threshold values for the three

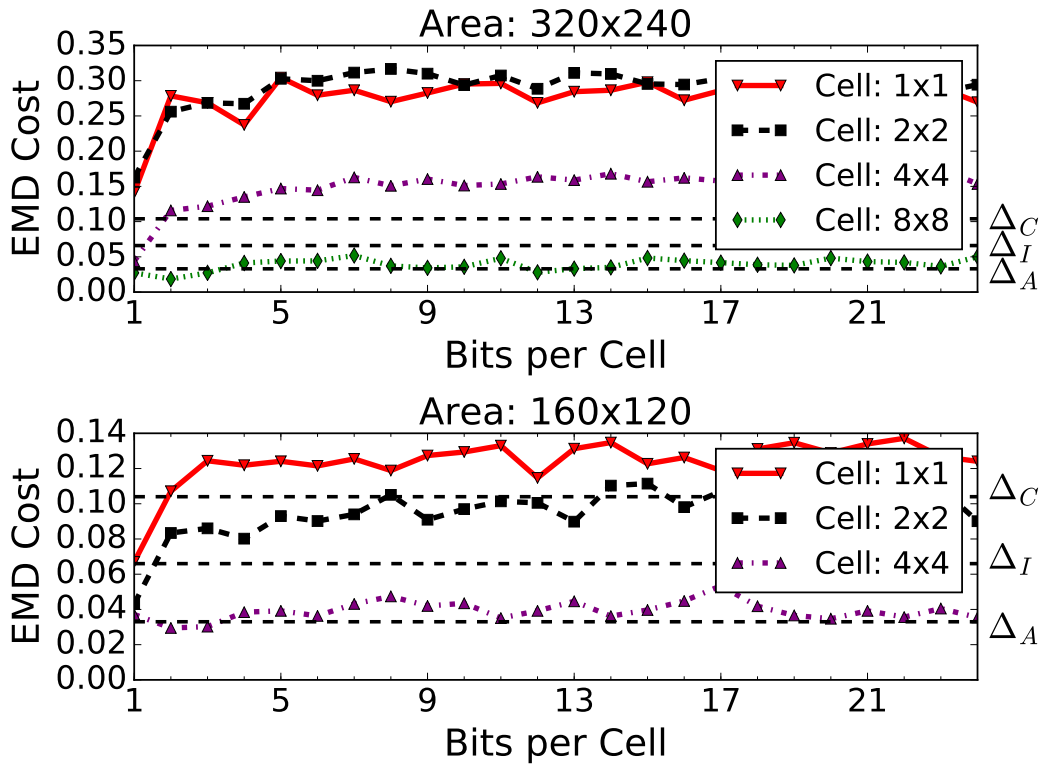

Figure 5.6: EMD cost varying the bits per cell.

policies discussed in the previous section: aggressive (Δ_A), conservative (Δ_C), and intermediate (Δ_I). For example, it can be seen that, for an intermediate policy, there are five configurations that produce unobservable streams, i.e., for area sizes 160x120 or 320x240 and cell sizes 4x4 or 8x8; and for area size 480x368 and cell size 8x8. The payload area size 480x368 was consistently found to generate streams identified as irregular by the DeltaShaper classifier when encoding more than 1 bit per cell.

Results show that as the cell size increases, the EMD cost tends to decrease. This is because macroblock partitions highly affect the achievable compression: larger partitions imply that larger areas of the frames will be colored with the same color thereby improving the efficiency of the video compression algorithm. Considering the worst case, an 8x8 cell results in splitting a macroblock into 4 different colored partitions; a 4x4 cell results in splitting a macroblock into 16 different colored partitions. Therefore, H.264's run-length scheme-based coding can more efficiently compress 8x8 cells than 4x4 cells. As for 2x2 and 1x1 cells, there is no prediction mode that can natively accommodate for such small areas. Therefore, the corresponding pixels are expected to be transmitted in a raw form, decreasing the compression's effectiveness.

A study of how unobservability changes, as a function of the number of bits per cell, was conducted for the area/cell size configurations found to be valid. Figure 5.6 shows the results, which cover the domain of data bit numbers, i.e., between 1 and 24 bits. In general, unobservability tends to be degraded as the number of bits increases. Some configurations, however, have a more flattened evolution of the EMD cost. In particular, two configurations fall consistently below the Δ threshold value for intermediate blocking policy (Δ_I), namely (160x120, 4x4) and (320x240,8x8). This means that both these encoding configurations are good candidates to generate covert streams that are both unobservable and can deliver a large data throughput range (due to the large data bit number). Note that since (320x240,8x8) is proportionally larger than (160x120, 4x4), both these configurations can encode the same amount of cells per frame.

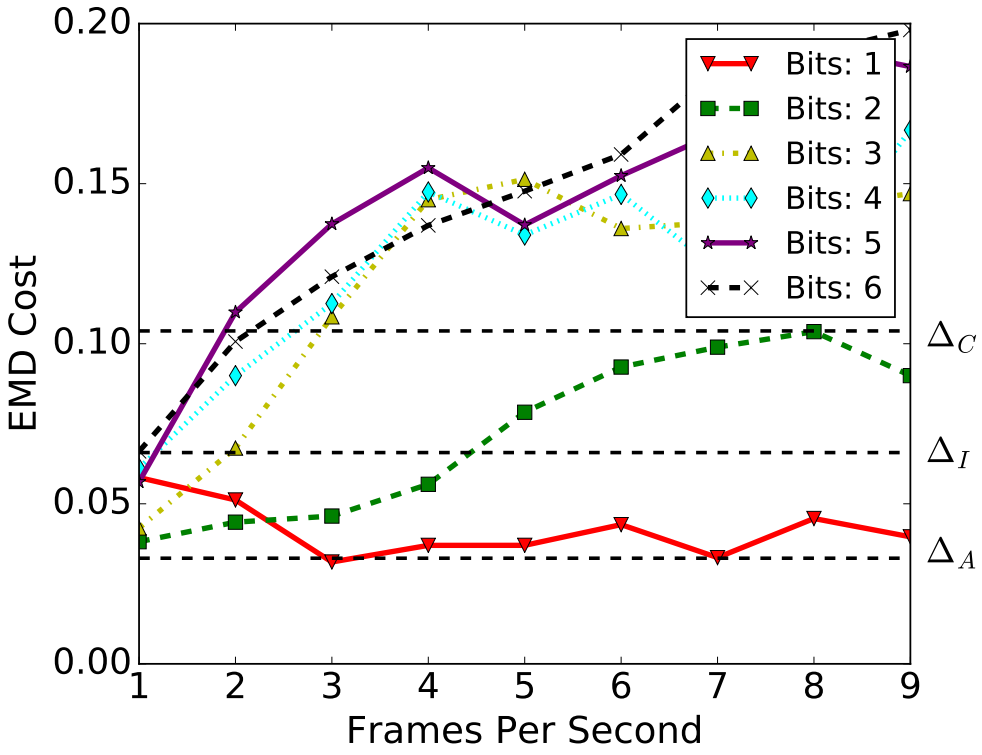

Figure 5.7: EMD cost varying the frame rate.

Lastly, it is conducted a study over how the frame rate affects unobservability. Figure 5.7 shows how the EMD cost varies as the frame rate is increased. To that end, a fixed payload area size 320x240 and cell size 8x8 are defined. Then, the EMD cost for a cell bit encoding range, varying between 1 and 6 bits per cell, is measured. Although the (160x120, 4x4) configuration encodes the same amount of bits per cell, these are mode difficult to decode due to higher error rate. The results show that increasing the frame rate will quickly result in EMD cost above Δ. A notable exception is the the data encoding scheme of 1 bit per cell, which remains below Δ for all tested frame rates. Encoding schemes with higher bit numbers can only tolerate the minimal frame rate value (1 frame-per-second).

5.4 Performance of the Covert Channel

In spite of the possibility to generate covert streams from numerous encoding configurations, DeltaShaper can only safely adopt those that result in unobservable streams. The fact that DeltaShaper has to satisfy this property constrains the maximum amount of data that can be encoded into the payload frames, enforcing a limit to the maximum performance that can be delivered by a DeltaShaper channel. Furthermore, performance can also be affected by decoding errors at the receiver when interpreting the cell color of payload frames. In fact, as the number of bits encoded per cell increases, the video compression algorithm tends to introduce more changes in the less significant bits of the color of each pixel, therefore introducing more decoding errors. This trend can be observed in Figure 5.8, which shows the growth of the error rate as the number of encoding bits increases for configurations (160x120, 4x4) and (320x240, 8x8), especially for encoding schemes above 9 bits per cell.

Taking into account both restrictions in terms of stream unobservability and decoding errors, it is possible to identify a candidate encoding configuration for DeltaShaper, which consists of: 320x240

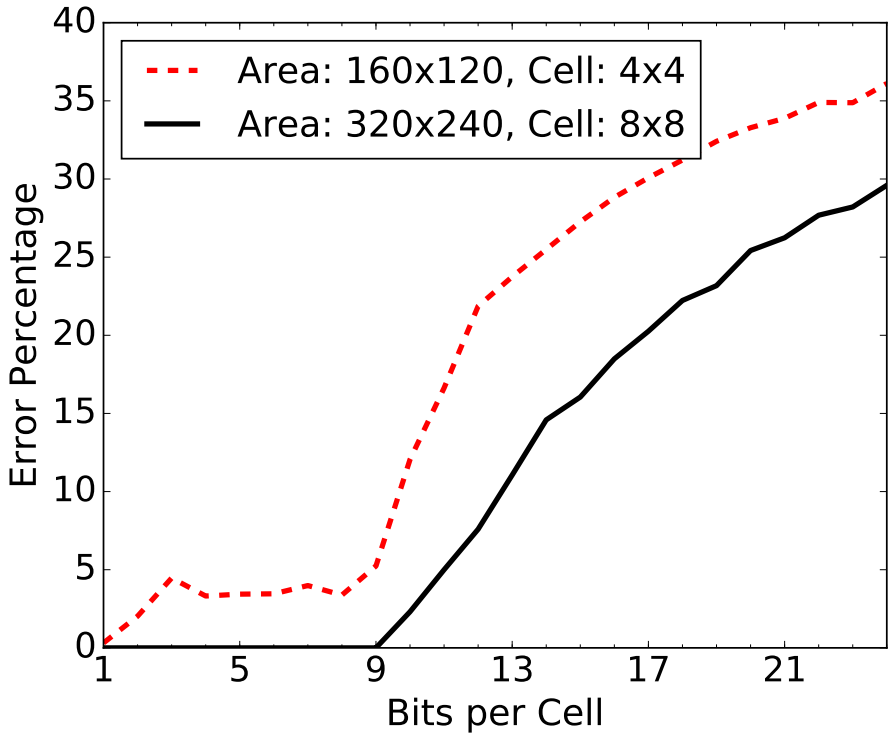

Figure 5.8: Error rate increasing bits per cell.

Params	With ECC	Without ECC
Throughput	0.32 KBps	0.39 KBps
RTT	2s 984ms	2s 973ms

Table 5.1: Performance measurements.

area size, 8x8 cell size, 6 bits per cell, at 1 frame per second. Due to design decisions, the bit encoding scheme is conservative (less than 9 bits per cell) because sporadic unrecoverable bit errors could result in the loss of an entire payload frame, which would significantly affect the covert data transfer rate. Table 5.1 lists the maximum throughput that DeltaShaper is able to achieve under this scheme: 0.32 and 0.39 KBps, respectively with and without error correction codes.

5.5 Resilience Against Active Attacks

The characterization of Skype streams performed up until this point admitted unconstrained network conditions for Skype transmissions. However, a censor may introduce controlled perturbations in the network in an attempt to increase the classification accuracy of Skype streams. By increasing its classification accuracy, a censor may be able to establish improved Δ thresholds that lead to the unveiling of DeltaShaper's covert channel. Therefore, attacks that may give the censor an advantage in breaking DeltaShaper's unobservability are now studied. In particular, the following network limitations are addressed: bandwidth throttling; loss of random packets; and the introduction of jitter between packet delivery. For assessing the impact of such network constraints over a censor's ability to distinguish between regular and irregular streams, the last test described in Section 5.2 is repeated for packet traces obtained in impaired network conditions. The visual impact of network perturbations on video streams is also measured as censors generally look for minimizing collateral damage on legitimate streams.

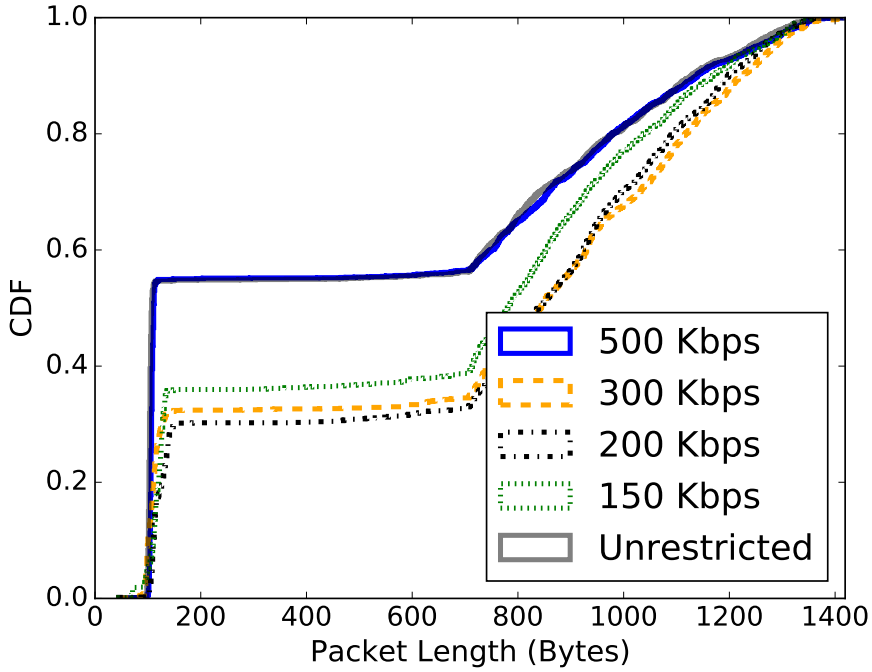

Figure 5.9: Packet length CDF of a reference stream varying the connection bandwidth.

Bandwidth	Classifier Accuracy
Unrestricted	88%
500Kbps	85%
300Kbps	75%

Table 5.2: Classifier accuracy when distinguishing Skype streams with varying bandwidth.

Bandwidth Throttling: The packet length distribution of Skype streams was found to be heavily dependent on the available network bandwidth. Figure 5.9 shows the packet length CDF of the reference stream (initially sampled with no network constraints) as the channel's bandwidth is further throttled between 150 and 500Kbps. Two main trends are evident. On the one hand, it is possible to identify two groups of streams which display different CDF shapes: In line with Skype's bandwidth requirements, one group corresponds to "Regular" video quality and comprises a bandwidth range between 128Kbps and 300Kbps; a second group includes streams between 400Kbps and 500Kbps corresponding to "High Quality Video" transmission. In addition, "Regular" video quality streams exhibit notable differences among them, especially for larger packets (700+ bytes length). These observations suggest that DeltaShaper's reference streams' sets must be chosen for specific network conditions.

In order to assess whether there is a gain in classification accuracy by reducing the bandwidth available to Skype, packet traces were obtained for three different bandwidth configurations: Unrestricted; 500Kbps; 300Kbps. For simulating reduced bandwidth, packets are queued for a maximum time t at the caller node, and dropped shall they be unable to be forwarded within t. In practice, t sets the maximum latency expected for the video-conferencing channel. For these experiments, and to set a bound on the maximum expected latency of a video-conferencing call for legitimate users, t is set to 1 second. Due to this fact, throttling the bandwidth to 300Kbps resulted in a high packet loss rate, which ultimately culminated in a high frames-per-second drop (from 30 to 5) and impairing the visual experience.

The results, depicted in Table 5.2, show that the classifier's accuracy worsens alongside bandwidth

Packet Loss	Classifier Accuracy
5%	76%
10%	75%
20%	78%

Table 5.3: Classifier accuracy when distinguishing Skype streams with varying packet loss rate.

reductions. Recall that the classifier's accuracy is measured in the cutoff point where true positive / negative rates are equal (and the respective false positive / negative rates are equal as well). For unrestricted bandwidth measurements, the classifier reaches an 88% accuracy, while reducing bandwidth to 500Kbps and 300Kbps yields a classification accuracy of 85% and 75%, respectively. As the results in Figure 5.9 suggest, the classification results between streams captured in unrestricted network conditions and in a network with its bandwidth limited to 500Kbps are very similar. This suggests that Skype regulates its output bit rate up to 500Kbps. In the contrary, bandwidth reduction is expected to increase Skype's video codec compression aggressiveness, which may hide more intricate details about a given video stream. By throttling bandwidth to 300Kbps, false positive / negative rates reach 25%, approximately doubling the false positive / negative rates observed under unconstrained bandwidth conditions (12%). As such, a censor employing bandwidth throttling faces an increase of false-negative rates and may incur in collateral damage by erroneously blocking legitimate Skype calls.

Packet Loss: To study the effect of packet loss in unobservability, packet traces were obtained in three different packet loss rates: 5%; 10%; and 20%. By visually examining ongoing streams, it is possible to set bounds over the visual experience that legitimate Skype users would expect to get while engaged in a video-conferencing call. Concretely, a 5% packet dropping rate is sufficient for sustaining an acceptable viewing experience, with the absence of significant frame dropping or the manifestation of compression artifacts. In its turn, a 10% packet dropping rate has shown to decrease the video-conferencing call's quality by introducing several compression artifacts on the video stream. Lastly, a 20% packet dropping rate proved to greatly affect the stream's quality, increasing the number of occurrences of visible compression artifacts. Furthermore, such degree of perturbation also introduced noticeable freezing of the stream's video layer.

The collected packet traces revealed a significant increase in the number of packets sent, which indicates that Skype adjusts its transmission rate to deal with adverse network conditions. The experiment results, depicted in Table 5.3, show that the classifier's accuracy is severely affected by packet dropping. A reduction in the classifier's accuracy means that a censor faces higher false negative / positive rates, and that it may be unable to correctly classify video streams without incurring in significant collateral damage. For this test, when analyzing the minimal degree of packet loss tested (5%), the false positive / negative rates are doubled when compared with unimpaired network conditions (an increase from 12% to 24%). While a censor may arguably induce network perturbations that cause the least connection impairment to legitimate users, these are shown to impact classification accuracy in a similar way as heavily disruptive packet dropping rates.

Packet Delay and Jitter: Video-conferencing applications are typically sensitive to network jitter, since multimedia data must be delivered to users in a timely and sequential fashion. However, the manipulation of jitter may induce changes on the network streams that carry such data. To study whether this effect translates into an increase of classification accuracy, packet traces were obtained for three different configurations of packet delay and jitter. More specifically, the connection's delay was set to 20ms (an acceptable end-to-end latency in legitimate video-conferencing calls) while jitter was adjusted in each different experiment to 10ms, 20ms, and 50ms, respectively. The described settings resulted in a gradu-

Network Jitter	Classifier Accuracy
10ms	82%
20ms	78%
50ms	82%

Table 5.4: Classifier accuracy when distinguishing Skype streams with varying network jitter.

ally worsening visual experience. Despite the first configuration - 10ms jitter - had no perceivable impact on the stream's quality, the remaining configurations led to the occurrence of compression artifacts and transiently frozen video, peaking at prohibitive frames-per-second drops on the last configuration tested.

The experiment's results, which are depicted in Table 5.4, show that the classifier's accuracy is negatively affected, although moderately, by the introduction of jitter. Although the packets carrying multimedia data may be received out-of-order, the introduction of jitter has not triggered a noticeable compensation mechanism from Skype, as it had happened with the increase of packet dropping rates. It can also be argued that a censor would refrain from employing other perturbations than the one which constitutes the first experiment, since the introduction of 20ms or 50ms of jitter greatly impacts the end-user experience for legitimate connections.

5.6 Alternative Traffic Features and Similarity Functions

While the classification of different kinds of Skype streams can be successfully achieved by using the frequency distribution of packet lengths, it is possible that some alternative characteristics of the network traffic can offer better classification accuracy. In the same way, it is questionable whether similarity functions other than EMD may be used for computing the similarity between streams. In order to answer these questions, Section 5.6.1 introduces a further study over other traffic characteristics that may be used for classification, namely: bi-gram distribution of packet sizes, inter-packet time and bi-gram distribution of inter-packet times. Since the content of packet payloads is ciphered, a censor may leverage such traffic features in order to distinguish between regular / irregular streams. While the analysis of packet inter-arrival times, packet lengths and bi-grams of packet lengths have been able to distinguish different kinds of network traffic in related literature, this section also introduces a study over bi-grams of inter-arrival times. Section 5.6.2 provides a comparison of EMD with the 2-sample Kolmogorov-Smirnov test, a popular similarity function used in related literature.

5.6.1 Exploring Alternative Traffic Features

In the interest of studying the impact of different traffic features on classification accuracy, the classifier has been tested over the unperturbed traffic samples introduced in Section 5.2 and the perturbed samples already studied in the previous section. The setup for each experiment is described below:

Bi-gram Distribution of Packet Sizes: In this test, the classifier assumes a bi-gram feature extraction from packet sizes. This process works by taking the size of each two packets, in every contiguous sequence of the original traffic. For instance, supposing that the traffic takes the form (150,100,700,800), the extracted bi-grams are (150,100), (100,700), (700,800). Similarly to previous experiments, each packet size is placed into the closest K-units bin. When compared to the analysis of the frequency distribution of packet lengths, the construction of bi-gram models is more computationally intensive.

Traffic Feature Similarity	Packet Length (PL) EMD / KS	Bi-Grams PL EMD / KS	Inter-Packet Time (IPT) EMD / KS	Bi-Grams IPT EMD / KS
	Unperturbed Streams			
	88% / 78%	85% / 72%	85% / 82%	88% / 77%
Bin size EMD / KS	K = 50 / K = 50	K = 50 / K = 50	K = 5 / K = 2,5	K = 5 / K = 5
	Perturbed Streams			
Bandwidth				
500Kbps	85% / 70%	82% / 78%	85% / 68%	88% / 68%
300Kbps	75% / 68%	75% / 58%	70% / 68%	72% / 68%
Bin size EMD / KS	K = 50 / K = 50	K = 50 / K = 50	K = 5 / K = 2,5	K = 5 / K = 5
Packet Loss				
5%	76% / 72%	82% / 72%	68% / 75%	78% / 70%
10%	75% / 82%	78% / 78%	72% / 73%	75% / 60%
20%	78% / 75%	68% / 72%	75% / 58%	88% / 72%
Bin size EMD / KS	K = 50 / K = 50	K = 50 / K = 200	K = 2,5 / K = 1	K = 5 / K = 10
Network Jitter				
10ms	82% / 78%	85% / 72%	75% / 72%	77% / 68%
20ms	78% / 85%	85% / 78%	67% / 60%	67% / 67%
50ms	82% / 78%	82% / 78%	68% / 55%	62% / 53%
Bin size EMD / KS	K = 50 / K = 50	K = 50 / K = 100	K = 5 / K = 2,5	K = 10 / K = 5

Table 5.5: Classifier accuracy using different feature and similarity functions.

Inter-Packet Times: A different traffic characteristic which departs from the use of the size of packets as feature extraction, is the arrival time between the packets of a given connection. To build this model, the time difference between any two consecutive packets must be computed and further discretized into K-units bins. For this case, a single bin unit represents 1 millisecond. When compared to the analysis of the frequency distribution of packet lengths, inter-packet time analysis implies the added cost of computing the time difference between consecutive packets.

Bi-gram Distribution of Inter-Packet Times: Similarly to the computation of bi-grams of packet lengths, it is possible to devise yet another statistical test, which takes into account the bi-grams for a contiguous series of inter-packet times. This test combines the overhead of constructing the bi-gram model with the computation of inter-packet times difference. Therefore, it represents the most computation intensive test presented in this section.

The classification results for the above feature sets are depicted in Table 5.5. The bin size which allowed for an overall better classification accuracy for a particular network setting is represented under the results obtained for each classifier. The experiments' outcome shows that the classifier attains at least 85% accuracy in distinguishing between regular and irregular Skype streams, even when using traffic features other than the frequency distribution of packet sizes of a given Skype connection. For unperturbed network conditions, the accuracy of the classifier based on bi-grams of inter-packet times matches that of the simpler packet length-based classifier, attaining a classification accuracy of 88%.

When alternative traffic features were used to classify streams in constrained networks, the classifier generally followed a decline in accuracy versus unrestricted conditions. However, classifiers based on different traffic features are able to obtain better classification accuracy in specific network constraints settings. For a bandwidth restriction of 500Kbps, the use of bi-grams of inter-packet times yields the best classification accuracy among the different traffic features, peaking at 88% accuracy. For a network with bandwidth throttled to 300Kbps, the classifiers based on the analysis of packet sizes were able to achieve a better accuracy (75%). For packet loss restrictions and the introduction of network jitter, the analysis of bi-grams of packet lengths results in the best overall accuracy among the remaining traffic

features. For such restrictions, the accuracy of the classifier is comprehended between 68% - 82% and 82% - 85%, respectively. It is also possible to observe that classifiers based on the analysis of inter-packet times and respective bi-grams suffer a high decrease in accuracy when used in networks with artificial jitter introduction, resulting in a classification accuracy comprehended between 67% - 75% and 62% - 77%, respectively. Thus, these results suggest that the introduction of artificial jitter prevents the construction of an accurate model for distinguishing between regular / irregular streams when using inter-packet times for feature extraction.

It is possible to conclude that, although the use of the frequency distribution of packet sizes may not be the absolute best traffic feature to inspect in every test-case scenario, it still offers a consistent and reasonable classification accuracy while providing a sound and lightweight approach for building classification models. The analysis of the frequency distribution of packet lengths has also shown to be the best approach to classify streams in unperturbed network environments. Hence, the analysis of such traffic feature may be advantageous to a censor which attempts to classify streams while refraining itself from introducing network perturbations that can affect the visual quality of legitimate Skype streams.

5.6.2 Exploring Alternative Similarity Functions

While the EMD translates into the hard cost of transforming one probability distribution into another, the 2-sample Kolmogorov-Smirnov (KS) measures the maximum vertical distance between the empirical cumulative distribution functions of two given samples. Although the KS test for statistical significance has been successfully applied for traffic classification in related literature, it was found that EMD provides a better traffic classification accuracy in the context of this thesis. A comparison between EMD and KS distance-based classifiers, for all previously studied traffic features, is presented in Table 5.5. In a general way, the use of KS translates in a worse classification accuracy. Particularly, for unrestricted network conditions, the combination of the KS similarity metric and the analysis of inter-packet times yields 82% accuracy, 6% short of the maximum classification accuracy that can be achieved by EMD under the same network conditions (88%).

5.7 Finding a Proper Calibration Period for DeltaShaper

The evaluation process conducted thus far has admitted the use of packet traces captured in a 30 seconds time-span. Naturally, this implies that a censor is considered to attempt the classification of Skype streams within this time interval. In practice, a real-world censor is expected to attempt the classification of regular / irregular streams as quickly as possible so that it may rapidly detect and block the transmission of prohibited content. Although 30 seconds already comprises a relatively short time-span, Section 5.2 has shown that the analysis of packet traces captured over this period is sufficient for a censor to distinguish between regular / irregular streams with a high degree of confidence. However, it is arguable whether a classifier applied by a censor is able to correctly distinguish between regular / irregular streams in shorter time-spans.

In the interest of choosing adequate parameters for the transmission of covert data, DeltaShaper's calibration period shall be set for the time-span a censor is expected to obtain packet traces of a given stream, deploying its own traffic analysis techniques thereafter. Naturally, this calibration period should match the censor's packet traces' collection time-span that is more advantageous for distinguishing between regular / irregular streams. To check how the classification of Skype streams is affected by

Traffic Collection Duration	5s	10s	15s	20s	25s	30s
Classifier Accuracy	72%	82%	82%	82%	85%	88%

Table 5.6: Classifier accuracy when distinguishing Skype streams with varying traffic collection times.

Use Case	With DeltaShaper	Without DeltaShaper	Overhead
A. Wget	1m 9s 830ms	7ms	9,975.7×
B. FTP	2m 45s	8s 528ms	19×
C. SMTP	2m 42s	37s 913ms	4.3×
D. SSH	1m 51s 493ms	6s 485ms	17.2×
E. Telnet	1m 17s 471ms	7s 670ms	10.1×
F. Netcat Chat	1s 147ms	11ms	133×
G. SSH Tunnel	3m 46s 55ms	21s 940ms	10.3×

Table 5.7: Execution time for DeltaShaper use cases.

the duration of the collected packet traces, the original 30 seconds packet traces were increasingly split in 5 seconds intervals. Then, the classifier based in EMD and packet lengths was used to classify the resulting partial streams.

Table 5.6 depicts the results of this study for the 6 different time-spans obtained. It is possible to observe that the accuracy of the classifier follows the increase of the time period dedicated to the collection of Skype streams' packet traces. Clearly, traffic signatures obtained from captures lasting for 5 seconds translate into a weak classification accuracy (72%). However, a significant improvement of the accuracy of the classifier is noticeable when applied over traffic captures within the range of 10 - 25 seconds (82% - 85%). Lastly, a traffic capture with the duration of 30 seconds has allowed the classifier to hit 88% accuracy, as already detailed in previous sections. These results suggest that the increase of the traffic collection period translates into the gathering of higher amounts of data which can then be used to build more accurate signatures for Skype streams. For instance, the collection of packet traces over small periods of time may fail to collect data that better characterize irregular streams, such as scene changes or quick motion scenes in the transmitted videos.

5.8 Use Cases

Given that the data throughput that can be achieved while preserving unobservability is relatively small, DeltaShaper is not adequate for the transmission of bulk data. Nevertheless, it can sustain the execution of applications that are not bandwidth hungry and are latency tolerant. To confirm this hypothesis, DeltaShaper has been tested with seven use cases: fetching a 4KB web page from the receiver (Case A), downloading a 4KB file from an FTP server running on the receiver (Case B), tunneling a small email (two small sentences) through an SMTP server running on the receiver (Case C), issuing an SSH session to the receiver and performing the "ls" command (Case D), issuing a telnet session to the receiver and performing the "ls" command (Case E), sending a message directed at a netcat server running on the receiver, mimicking a text chat (Case F), tunneling an SSH session to a remote SSH server through the receiver, and performing the "ls" command (Case G). In use cases A-F, the client communicates only with the receiver over a DeltaShaper channel. In case G, the receiver acts as a relay by tunneling traffic between the client and a remote party. Excepting case A, all other use cases are performed interactively, where a proficient user types the commands required to establish the different types of connections in a terminal.

Table 5.7 provides a summary of the execution time for each use case when performed with and

without DeltaShaper, i.e., using overt communication channels between client and receiver. As depicted, the execution time is several orders of magnitude higher in DeltaShaper than in overt channels. For instance, the establishment of Telnet and SSH sessions implies a added time overhead between 10× - 20×. In a similar way, the FTP session establishment and subsequent file download implied an added time overhead of about 20×. As for sending an email through an SMTP server available on the callee machine, about half of the experiment time was spent in the Telnet session establishment. The remaining time was used for issuing commands through the Telnet session in order to connect to the SMTP server and send the email message. Nevertheless, in spite of the high delay experienced by users, all tested use cases are fully functional.

Such large overheads are expected given the low throughput and high latency that DeltaShaper is currently able to deliver. However, although DeltaShaper is able to correctly forward IP packets between the system's endpoints, in some circumstances the client / server applications took some added time to produce the expected answer. This happened even after the confirmation that a packet had been successfully delivered to the receiver network stack. Some exploratory work regarding this issue lead to the acknowledgment of the concept of bandwidth-delay product (BDP). The BDP consists in the product of a data link's capacity (in bps) and its round-trip time (in seconds), resulting in the maximum amount of data that can travel through a network link at any given time. To operate at peak efficiency, a sender shall send a sufficiently large quantity of data before being required to stop and wait until an acknowledgment is received from the counterpart endpoint. If the sent data is insufficient for filling the link, when compared with the BDP, then the link is not being used at its fullest capacity. Several TCP optimization techniques for high bandwidth-delay product / high-latency links focus on the redefinition of the size of internal buffers and TCP receive window sizes. To address this issue in further iterations of DeltaShaper, specific TCP tuning may need to be performed at the network stack level.

5.9 Security Considerations

The following paragraphs discuss relevant security properties of DeltaShaper against attacks launched by a censor, with particular emphasis on the mismatches that can be observed between covert and cover channels introduced in Section 2.2.6:

Defense Against Active Probing: In order to thwart active probing attempts, DeltaShaper servers can be configured to only accept calls from added contacts, ignoring calls from unknown Skype IDs. To advertise DeltaShaper servers, volunteers may share circumvention servers' Skype IDs and respective access credentials (like a password) through some out-of-band channel among users within the censored region. A client can then place a contact request, including the corresponding credential, to a DeltaShaper server. When a DeltaShaper server receives such request, it is able to validate the credential, allowing video-conferencing calls to take place. In this way, the circumvention server does not blindly accept calls from Skype endpoints which may be controlled by the censor.

Architectural Mismatches: To use DeltaShaper, citizens living in repressive regimes need to place a call to an entity out of the censor's control. Such entity may be a friend or relative living outside of the censor's sphere of influence, although volunteers or service providers may also set up DeltaShaper servers to aid in circumvention. In DeltaShaper, the only contact information that must be shared in the clear is in the form of Skype IDs, preventing the censor from directly acting upon particular IP ranges. However, to avoid having their IPs identified as Skype calls' hotspots, volunteers can further rotate their servers' IPs frequently.

Content Mismatches: Given that DeltaShaper generates covert streams indistinguishable from regular streams, censors may not block covert streams without causing considerable disruption of the Skype service. Nevertheless, it is possible that by replaying the same carrier video for covert channel modulation purposes, repeatable packet distribution patterns can emerge and be spotted by a censor. To mitigate this attack, the carrier video may be rotated periodically by DeltaShaper so as to limit pattern exposure to the censor. Further iterations of DeltaShaper may also allow for a live webcam feed to be used as carrier video.

Channel Mismatches: The censor may manipulate the traffic so as to launch a denial of service attack or delay the delivery of IP frames. Such could be achieved by dropping or delaying packets of active Skype streams. However, shall this attack prevent DeltaShaper from delivering a given IP frame, the TCP layer is trusted to actively recover from losses by retransmitting missing packets automatically. If the censor extends its attack period, the Skype connection will experience a reduction in quality which will also affect the Skype calls of legitimate users.

Utilization Mismatches: The censor may attempt to identify a DeltaShaper connection by detecting the abnormal duration of video-conferencing calls. Despite the high latency offered by DeltaShaper's covert channel, a client is still able to execute several applications within reasonable times a regular video-conferencing call is expected to hold. Ultimately, DeltaShaper can be configured with time thresholds aimed at keeping the connection alive or tearing it down automatically (and warn users), preventing a covert transmission to last for an unusual duration.

Summary

This chapter introduced the experimental evaluation of DeltaShaper, detailing the experiments conducted in order to assess the unobservability of its covert channels. While DeltaShaper offers a high-latency and low-throughput covert channel, it is sufficient for the execution of several common TCP/IP applications. DeltaShaper has shown to be resilient against traffic analysis while exhibiting an interesting trade-off between unobservability and achievable performance. The perturbation of network conditions has shown to negatively impact the classification accuracy of Skype streams. Thus, apart from possibly impacting the quality of legitimate streams, a censor has no apparent gain in perturbing the network to establish thresholds aimed at detecting DeltaShaper connections. Regular and irregular Skype streams have also been classified with several traffic features and similarity functions. As a result, the analysis of the frequency distribution of packet lengths has shown to accurately classify streams, when used in tandem with EMD. This chapter concludes by discussing relevant security considerations of DeltaShaper, describing how the system is able to prevent mismatches between its cover / covert protocol and how it is able to defend itself against active probing attempts perpetrated by a censor. The next chapter concludes this document by summarizing the main findings of this thesis and introducing some directions for future work.

6 Conclusions and Future Work

Repressive regimes around the world have employed Internet censorship techniques so as to prevent their citizens from accessing or publishing rightful information. To tackle this issue, increasingly sophisticated censorship circumvention systems have been proposed over the last few years. Recent efforts have focused on enabling covert channels over popular protocols that a censor may be unwilling to block due to collateral damage. In particular, relevant interest has grown on offering covert channels over multimedia streaming applications. For this approach to be successful the covert channel must be *unobservable*, i.e, a censor must not be able to distinguish regular call streams from streams carrying covert data. Although recent systems are able to deploy unobservable covert channels over multimedia applications, these systems fail to provide a covert channel which allows for the interactive transmission of arbitrary data and that yields a sufficient throughput for the use of typical TCP/IP applications.

In this thesis, we have described the design and implementation of DeltaShaper, a novel Internet censorship circumvention system which leverages the video layer of video-conferencing systems to build a covert channel. By synthesizing and overlaying carefully crafted data into video frames, DeltaShaper offers a covert channel which allows for the tunneling of network layer packets, thus supporting the forwarding of high-level application protocols. The main technical challenge addressed was to design an encoding scheme that was able to preserve the unobservability of the covert channel while providing acceptable throughput.

The experimental evaluation conducted over DeltaShaper shows that a careful choice of encoding parameters is able to produce synthesized video streams which transmission is indistinguishable from typical video-chat sessions. Moreover, the covert channel offered by DeltaShaper is sufficient to withstand the execution of applications that can tolerate a high-latency and low-throughput, enabling interactive sessions of several high-level application protocols with an added time cost between ten to twenty times when compared to the use of overt channels.

Our current evaluation assumes an almost exclusive use of the video-conferencing software alongside with either DeltaShaper or the video processing pipelines that enables us to perform a simple video transmission. However, the processor workload may affect the video encoding process, leading to further discrepancies on the generated network traffic.

An important direction for future work comprises a deeper study of traffic patterns generated by video-conferencing applications upon which DeltaShaper may run. Firstly, it would be interesting to generate traffic analysis models which can better capture the impact of the endpoints' CPU activity on network traffic. Secondly, traffic analysis can be extended in order to study a wider range of network perturbations that may be introduced by a censor. Lastly, further analysis is necessary in order to assess whether traffic patterns change significantly with the introduction of real chat audio transmission alongside video.

References

Al-Saqaf, W. (2016). Internet censorship circumvention tools: Escaping the control of the Syrian regime. *Media and Communication 4*(1).

Aryan, S., H. Aryan, and J. A. Halderman (2013). Internet censorship in Iran : A first look. In *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet*, Washington, DC, USA.

Barradas, D., N. Santos, and L. Rodrigues (2016). Síntese de Vídeo para Evasão de Censura na Internet. In *Actas do Oitavo Simpósio de Informática, INForum 16*, Lisbon, Portugal.

Bowen, K. and J. Marchant (2014/2015). Internet Censorship in Iran: Preventative, Interceptive and Reactive. `https://smallmedia.org.uk/revolutiondecoded/a/RevolutionDecoded_Ch2_InternetCensorship.pdf`. Last Accessed: 2016-07-13.

Brubaker, C., A. Houmansadr, and V. Shmatikov (2014). CloudTransport: Using cloud storage for censorship-resistant networking. In E. De Cristofaro and S. Murdoch (Eds.), *Privacy Enhancing Technologies*, Volume 8555 of *Lecture Notes in Computer Science*, pp. 1–20. Springer International Publishing.

Burnett, S., N. Feamster, and S. Vempala (2010). Chipping away at censorship firewalls with user-generated content. In *Proceedings of the 19th USENIX Conference on Security*, Washington, DC, USA.

Cai, X., X. C. Zhang, B. Joshi, and R. Johnson (2012). Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, USA, pp. 605–616.

Chaabane, A., T. Chen, M. Cunche, E. De Cristofaro, A. Friedman, and M. A. Kaafar (2014). Censorship in the wild: Analyzing Internet filtering in Syria. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, Vancouver, BC, Canada, pp. 285–298.

Cheddad, A., J. Condell, K. Curran, and P. M. Kevitt (2010). Digital image steganography: Survey and analysis of current methods. In *Signal Processing 90(3)*, pp. 727–752.

Chen, J.-W., C.-Y. Kao, and Y.-L. Lin (2006). Introduction to H. 264 advanced video coding. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*.

Dainotti, A., C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé (2011). Analysis of country-wide Internet outages caused by censorship. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, Berlin, Germany.

David Fifield (2014). A Child's Garden Of Pluggable Transports. `https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports`. Last Accessed: 2016-07-13.

Dingledine, R., N. Mathewson, and P. Syverson (2004). Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium*.

Djebbar, F., B. Ayad, K. A. Meraim, and H. Hamam (2012). Comparative study of digital audio steganography techniques. In *EURASIP Journal on Audio, Speech, and Music Processing*, pp. 1–16.

Douceur, J. R. (2002). The Sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, London, UK, pp. 251–260.

Dyer, K. P., S. E. Coull, T. Ristenpart, and T. Shrimpton (2013). Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, Berlin, Germany, pp. 61–72.

Dyer, K. P., S. E. Coull, and T. Shrimpton (2015). Marionette: A programmable network-traffic obfuscation system. In *Proceedings of the 24th USENIX Conference on Security Symposium*, Washington, D.C., USA, pp. 367–382.

Elahi, T., C. M. Swanson, and I. Goldberg (2015). Slipping past the cordon: A systematization of Internet censorship resistance. In *CACR Tech Report 2015-10*.

Ensafi, R., D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson (2015). Examining how the Great Firewall discovers hidden circumvention servers. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, Tokyo, Japan, pp. 445–458.

Esfandiari, G. (2013). Iran admits throttling Internet to 'preserve calm' during election. `http://www.rferl.org/content/iran-internet-disruptions-election/25028696.html`. Last Accessed: 2016-07-13.

Feamster, N., M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger (2002). Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, USA, pp. 247–262.

Feller, C., J. Wuenschmann, T. Roll, and A. Rothermel (2011). The VP8 video codec-overview and comparison to H. 264/AVC. In *Proceedings of the IEEE International Conference on Consumer Electronics*, Berlin, Germany.

FFmpeg (2000). `https://ffmpeg.org`. Last Accessed: 2016-07-13.

Fifield, D., C. Lan, R. Hynes, P. Wegmann, and V. Paxson (2015). Blocking-resistant communication through domain fronting. In *Proceedings on Privacy Enhancing Technologies 2015.2*, Philadelphia, PA, USA, pp. 46–64.

Geddes, J., M. Schuchard, and N. Hopper (2013). Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, Berlin, Germany, pp. 361–372.

GStreamer (2001). `https://gstreamer.freedesktop.org/`. Last Accessed: 2016-07-13.

Hahn, B., R. Nithyanand, P. Gill, and R. Johnson (2016). Games without frontiers: Investigating video games as a covert channel. In *Proceedings of the IEEE 1st European Symposium on Security & Privacy*.

Herrmann, D., R. Wendolsky, and H. Federrath (2009). Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, Chicago, Illinois, USA, pp. 31–42.

Houmansadr, A., C. Brubaker, and V. Shmatikov (2013). The parrot is dead: Observing unobservable network communications. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, pp. 65–79.

Houmansadr, A., G. T. Nguyen, M. Caesar, and N. Borisov (2011). Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, pp. 187–200.

Houmansadr, A., T. J. Riedl, N. Borisov, and A. C. Singer (2013). I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*.

Houmansadr, A., E. L. Wong, M. Inc, and V. Shmatikov (2014). No direction home: The true cost of routing around decoys. In *Proceedings of the 21th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA.

Kadianakis, G. and N. Mathewson. Obfsproxy architecture. `https://www.torproject.org/projects/obfsproxy.html.en`. Last Accessed: 2016-07-13.

Karlin, J., D. Ellard, A. Jackson, C. Jones, G. Lauer, D. Mankins, and T. Strayer (2011). Decoy routing: Toward unblockable Internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*, San Francisco, CA, USA.

Knockel, J., J. R. Crandall, and J. Saia (2011). Three researchers, five conjectures: An empirical analysis of tom-skype censorship and surveillance. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*.

Kohls, K., T. Holz, D. Kolossa, and C. Pöpper (2016). SkypeLine: Robust hidden data transmission for VoIP. In *Proceedings of the 2016 ASIA Computer and Communications Security*.

Li, B., J. He, J. Huang, and Y. Q. Shi (2011). A survey on image steganography and steganalysis. In *Journal of Information Hiding and Multimedia Signal Processing, 2(2)*, pp. 142–172.

Li, S., M. Schliep, and N. Hopper (2014). Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, Scottsdale, Arizona, USA, pp. 163–172.

Maersk-Moller, P. (2012). Snowmix. `https://sourceforge.net/projects/snowmix/`. Last Accessed: 2016-07-13.

Marczak, B., N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson (2015). China's great cannon. `https://citizenlab.org/wp-content/uploads/2009/10/ChinasGreatCannon.pdf`. Last Accessed: 2016-07-13.

McPherson, R., A. Houmansadr, and V. Shmatikov (2016). CovertCast: Using Live Streaming to Evade Internet Censorship. In *Proceedings of Privacy Enhancing Technologies*.

Moghaddam, H., B. Li, M. Derakhshani, and I. Goldberg (2012). SkypeMorph: Protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, USA, pp. 97–108.

Netfilter (1998). `http://www.netfilter.org/`. Last Accessed: 2016-07-13.

OpenNet Initiative (2009). China's Green Dam: The Implications of Government Control Encroaching on the Home PC. `https://opennet.net/sites/opennet.net/files/GreenDam_bulletin.pdf`. Last Accessed: 2016-07-13.

Panchenko, A., L. Niessen, A. Zinnen, and T. Engel (2011). Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, Chicago, Illinois, USA, pp. 103–114.

Reuters (2016). Iran orders social media sites to store data inside country. `http://www.reuters.com/article/internet-iran-idusl8n18q0in`. Last Accessed: 2016-07-13.

RFC 6246 - TLSHandshake. `https://tools.ietf.org/html/rfc5246`. Last Accessed: 2015-11-19.

Rubner, Y., C. Tomasi, and L. J. Guibas (2000, November). The Earth Mover's Distance As a Metric for Image Retrieval. *Int. J. Comput. Vision 40*(2), 99–121.

Schuchard, M., J. Geddes, C. Thompson, and N. Hopper (2012). Routing around decoys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, USA, pp. 85–96.

skype4py (2007). `https://github.com/Skype4Py/Skype4Py`. Last Accessed: 2016-07-13.

Statistic Brain Research Institute (2015). Skype Company Statistics. `http://www.statisticbrain.com/skype-statistics/`. Research Date: June 6th, 2015, Last Accessed: 2016-07-13.

Tanash, R. S., Z. Chen, T. Thakur, D. S. Wallach, and D. Subramanian (2015). Known unknowns: An analysis of Twitter censorship in Turkey. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, Denver, Colorado, USA, pp. 11–20.

Ultrareach Internet Corp. (2002). Ultrasurf. `http://ultrasurf.us`. Last Accessed: 2016-07-13.

v4l2loopback (2005). `https://github.com/umlaeute/v4l2loopback`. Last Accessed: 2016-07-13.

Vines, P. and T. Kohno (2015). Rook: Using Video Games As a Low-Bandwidth Censorship Resistant Communication Platform. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*.

Wagner, B. (2008). Deep packet inspection and Internet censorship: International convergence on an 'integrated technology of control'. In *Proceedings of the 3rd Annual Giganet Symposium*, Hyderabad, India.

Wang, L., K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton (2015). Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, Colorado, UK, pp. 57–69.

Wang, Q., X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov (2012). CensorSpoofer: Asymmetric communication using IP spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, USA, pp. 121–132.

Weinberg, Z., J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh (2012). StegoTorus: A camouflage proxy for the Tor anonymity system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, USA, pp. 109–120.

White, A. M., A. R. Matthews, K. Z. Snow, and F. Monrose (2011). Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, pp. 3–18.

Wicker, S. B. (1994). *Reed-Solomon Codes and Their Applications*. IEEE Press.

Wiegand, T., G. J. Sullivan, G. Bjontegaard, and A. Luthra (2003). Overview of the H.264/AVC video coding standard. In *IEEE Transactions on Circuits and Systems for Video Technology Volume: 13 Issue: 7*, pp. 560–576.

Winter, P., T. Pulls, and J. Fuss (2013). ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, Berlin, Germany, pp. 213–224.

Wright, C. V., L. Ballard, F. Monrose, and G. M. Masson (2007). Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of 16th USENIX Security Symposium*, Boston, MA, pp. 4:1–4:12.

Wright, C. V., S. E. Coull, and F. Monrose (2009). Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, San Diego, CA, USA.

Wustrow, E., S. Wolchok, I. Goldberg, and J. A. Halderman (2011). Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Conference on Security*, San Francisco, CA, USA.

Zhou, W., A. Houmansadr, M. Caesar, and N. Borisov (2013). SWEET: Serving the web by exploiting email tunnels. In *Proceedings of the 6th Workshop on Hot Topics in Privacy Enhancing Technologies*.

Zittrain, J. L. and J. G. Palfrey (2007). Access denied: the practice and policy of global internet filtering. In *Oxford Internet Institute, Research Report No. 14*.