UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Unobservable Multimedia-based Covert Channels for Internet Censorship Circumvention

Diogo Miguel Barrinha Barradas

Supervisor: Doctor Luís Eduardo Teixeira Rodrigues
Co-Supervisor: Doctor Nuno Miguel Carvalho dos Santos

Thesis approved in public session to obtain the PhD Degree in Computer Science and Engineering

Jury Final Classification: Pass with Distinction and Honour

2021

TÉCNICO LISBOA

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Unobservable Multimedia-based Covert Channels for Internet Censorship Circumvention

Diogo Miguel Barrinha Barradas

Supervisor: Doctor Luís Eduardo Teixeira Rodrigues
Co-Supervisor: Doctor Nuno Miguel Carvalho dos Santos

Thesis approved in public session to obtain the PhD Degree in Computer Science and Engineering

Jury Final Classification: Pass with Distinction and Honour

Jury

Chairperson: Doctor José Manuel da Costa Alves Marques, Instituto Superior Técnico, Universidade de Lisboa

Members of the Committee:

Doctor Luís Eduardo Teixeira Rodrigues, Instituto Superior Técnico, Universidade de Lisboa

Doctor Henrique João Lopes Domingos, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Doctor Mohammad Tariq Ehsan Elahi, School of Informatics, University of Edinburgh, UK

Doctor Bruno Emanuel da Graça Martins, Instituto Superior Técnico, Universidade de Lisboa

Funding Institutions

Universidade de Lisboa
Fundação para a Ciência e Tecnologia

2021

Acknowledgements

The work presented in this thesis was only made possible with the help of many people who crossed my path during the last few years. I am forever grateful to them.

I start by expressing my kind thanks to my supervisor, Prof. Luís Rodrigues, who first incentivized me to pursue a Ph.D.. Apart from instilling the value of hard work, from all the helpful discussions, and from the countless insights he shared with me regarding numerous academic affairs, I mainly thank him for continuously challenging me to deliver high-quality work and for pushing me beyond my limits. In like manner, I warmly thank my co-supervisor, Prof. Nuno Santos, for his constant and energetic encouragement, absolutely critical brainstorming sessions, and for the many hours invested in methodologically training me to conduct research, to write clearly, and to present ideas neatly. I am deeply honored for the opportunity to work, study, and learn under their guidance. Their fierce dedication has inspired me to pursue an academic career and I hope our paths may cross again in the future.

During my Ph.D., I had the opportunity to collaborate and learn from other talented researchers and faculty. I thank my co-authors Salvatore Signorello, Fernando Ramos, Zachary Weinberg, and Nicolas Christin. Working with them gave me the opportunity to broaden the spectrum of my research and helped me grow as a scientist. I would also like to thank the faculty of the Distributed Systems Group (GSD) at INESC-ID for their willingness to exchange ideas and for the many times their valuable advice helped me shape and improve my research.

I would like to thank all my friends and colleagues at GSD for their companionship and support. I am grateful for all the great times we spent together, including the many meals, drinks, and coffee breaks shared over the last few years. In no particular order, I thank: João Loff, Tiago Brito, Miguel Coimbra, João Santos, Igor Zavalyshyn, Daniel Castro, Daniel Andrade, Paulo Silva, Pedro Joaquim, Manuel Bravo, Subhajit Sidhanta, Daharewa Gureya, Rodrigo Bruno, Richard Martinez, Paolo Laffranchini, Shady Issa, Cláudio Correia, Taras Lykhenko, João Neto, Mennan Selimi, and Pedro Raminhas. A special thanks goes to Daniel Porto, the networking wizard who always found the time for discussing the details of my experimental setups with me.

Most importantly, I would like to thank the unconditional support of all my long-time friends and family who were always there for me throughout this journey. I wholeheartedly thank Maria for her never-ending encouragement, for her patience to stay by my side despite the constant deadlines, and for shifting the weights on my work-life balance scale. Our evening strolls kept me sane during the long weeks where all experiments seemed to go nowhere.

This work has been partially funded by the Universidade de Lisboa (via the BD2018 doctoral scholarship) and by Fundação para a Ciência e Tecnologia (FCT) (via the SFRH/BD/136967/2018 grant, the project COSMOS with reference PTDC/EEI-COM/29271/2017, and the INESC-ID funding with reference UIDB/50021/2020).

To a world where people can speak
up their minds without fear,

Abstract

Totalitarian states are known to deploy large-scale surveillance and censorship mechanisms in order to deter citizens from accessing or publishing information on the Internet. Still, even the most oppressive regimes cannot afford to always block all channels with the outside world, and usually allow the operation of widely used services such as video-conferencing applications. This has given rise to the development of censorship-resistant communication tools that rely on the establishment of covert channels in the Internet by encoding covert data within popular multimedia protocols that use encrypted communication, e.g., Skype.

A recent approach for the design of such tools, named multimedia protocol tunneling, modulates covert data into the audio and/or video feeds provided to multimedia applications. However, depending on the techniques used to embed covert data, and on the amount of information to embed, multimedia protocol tunneling tools may generate network flows that differ subtly from legitimate flows that do not carry covert channels. Notably, such differences can be uncovered using strictly passive methods (e.g., by observing the length or inter-arrival time of network packets). Incidentally, one of the major challenges faced by the above tools is that of achieving a proper balance between traffic analysis resistance and performance (e.g., achieve sufficient throughput for enabling web browsing activities).

This thesis focuses on the study of the efficacy of multimedia protocol tunneling tools to evade the censorship apparatus deployed by network adversaries, while providing sufficient performance for enabling common Internet activities (e.g., web browsing). First, we show that the covert channels generated by existing tools are prone to detection. Specifically, we developed a new machine learning (ML)-based traffic analysis framework which has broken the security assumptions of recent multimedia protocol tunneling tools. Second, we show that network adversaries currently possess the means to perform sophisticated ML-based network flow classification tasks at line-speed. To this end, we worked towards the efficient deployment of multiple ML-based traffic analysis frameworks (including our own) in programmable switches. Third, we devised a new technique for creating traffic analysis resistant covert channels over multimedia streams. Our approach, based on the careful modification of the video encoding pipeline of the WebRTC framework, allows for the creation of high-speed covert channels over multimedia flows whose traffic patterns closely resemble those of legitimate flows.

Organization of the document: The dissertation is divided in two parts. The first part provides the motivation and background for my work, identifies the main contributions of the thesis, offers an overview of the results achieved, and proposes directions for future work. The second part consists of a collection of the main papers that resulted from my work. This part reflects the content of the published papers, with minor formatting adjustments to fit the layout of the dissertation.

Resumo

Nos dias de hoje, vários regimes repressivos empregam mecanismos de censura na Internet com o intuito de impedir o acesso ou publicação de conteúdo considerado sensível, por parte dos cidadãos. Apesar disto, a maioria destes regimes permite a operação de serviços considerados essenciais, tais como aplicações de videoconferência. Este facto motivou o desenvolvimento de ferramentas de comunicação resistentes à censura, que dependem da ocultação de dados sensíveis em protocolos multimédia, como o Skype, que usam canais de comunicação cifrados.

Uma abordagem promissora, seguida na concepção destas ferramentas, é baseada na modulação de dados ocultos nas tramas de vídeo fornecidas às aplicações multimédia. No entanto, dependendo da função moduladora e da quantidade de dados a codificar, a utilização das ferramentas geradoras de túneis em protocolos multimédia (FGTPM) pode originar fluxos de rede que diferem de fluxos legítimos que não transportam dados ocultos. É de frisar que estas diferenças podem ser identificadas através de métodos estritamente passivos (por exemplo, através da observação do comprimento dos pacotes dos fluxos). Assim, um dos principais desafios enfrentados pelas FGTPM consiste em alcançar um equilíbrio entre o grau de resistência a análise de tráfego e o desempenho oferecido (e.g., atingir débito suficiente para permitir a navegação na Internet).

Esta tese visa o estudo das possibilidades oferecidas pelas FGTPM no que diz respeito à geração de canais encobertos que oferecem simultaneamente alto débito e resistência a análise de tráfego. Em primeiro lugar, mostramos que as FGTPM existentes são facilmente detectáveis. Especificamente, desenvolvemos uma nova metodologia de análise de tráfego baseada em aprendizagem automática (AA) capaz de detectar as FGTPM recentes com elevada precisão. Em segundo lugar, mostramos que um censor contemporâneo possui os meios para executar a classificação de fluxos de rede, baseada em AA, em tempo real. Nomeadamente, estudámos a utilização de técnicas de análise de tráfego baseadas em AA em comutadores programáveis. Por último, introduzimos uma nova técnica para gerar canais encobertos em fluxos multimédia, resistentes à análise de tráfego. A nossa abordagem, baseada na instrumentação da codificação de vídeo da plataforma WebRTC, permite a criação de canais encobertos com débito elevado através de fluxos multimédia cujos padrões de tráfego se assemelham aos de fluxos legítimos.

Organização do documento: Esta dissertação encontra-se dividida em duas partes. A primeira parte apresenta a motivação e o trabalho relacionado, identifica as contribuições principais da tese, oferece uma panorâmica sobre os resultados atingidos, e propõe direcções para trabalho futuro. A segunda parte consiste numa selecção das principais publicações que resultaram do meu trabalho. O conteúdo desta parte espelha o conteúdo dos artigos tal como foram publicados, salvo alterações de pormenor para acomodar a formatação usada na dissertação.

Keywords
Palavras Chave

Keywords

Internet censorship

Machine learning-based traffic analysis

Multimedia covert channels

Programmable switches

WebRTC streams

Palavras Chave

Censura na Internet

Análise de tráfego baseada em aprendizagem automática

Canais encobertos em fluxos multimédia

Comutadores programáveis

Fluxos WebRTC

Table of Contents

1 Introduction

1

Overview

Totalitarian states are known to deploy large-scale surveillance and censorship mechanisms in order to deter citizens from accessing or publishing information on the Internet [167]. However, not even the most oppressive regimes can afford to block all channels with the outside world, as these may be instrumental to preserve the sustainability of the regime itself and the reputation of the state before its international peers [58]. In particular, there is evidence that several countries do restrict access to information but maintain operational multiple communication services that are widely used by the population [82, 105]. This fact has given rise to the development of censorship-resistant communication tools that rely on the establishment of covert channels on the Internet by encoding covert data within popular encrypted protocols [29, 60, 71, 82, 129].

The primary goal of network covert channels is to conceal the existence of selected information flows between any two communicating parties from an adversary capable of monitoring traffic flows [226]. Due to the wide variety and increasing bandwidth of existing application layer protocols, these have become a prime target for the deployment of covert channels. Lately, tools able to generate network covert channels leverage the widespread use of encryption for developing data hiding mechanisms resorting to carrier applications like Skype [129]. A state-of-the-art approach for the design of such tools, named *multimedia protocol tunneling*, embeds concealed data into the application layer of multimedia protocols by modulating the audio or video feeds provided to the application [82, 97, 105, 121]. For instance, FreeWave [82] allows users located within censored regions to exchange network traffic with proxies outside the censored region by encoding traffic into acoustic signals sent over Skype calls. In short, multimedia protocol tunneling tools take advantage of two fundamental properties for hindering an adversary's efforts aimed at unveiling the presence of covert channels: i) they prevent an adversary from inspecting the contents of transmissions in plaintext; ii) they force an adversary to distinguish between a protocol's legitimate and covert executions through the analysis of traffic patterns alone.

However, depending on the techniques used to embed covert data, and on the amount of information to embed, multimedia protocol tunneling tools may generate network flows that subtly differ from legitimate flows that do not carry covert channels [66]. Such differences can be uncovered using strictly passive methods (e.g., by observing the length or inter-arrival delay of transmitted network packets) or by analyzing the protocol's behavior in response to active network manipulations (e.g., the loss or reorder of packets). Thus, an important property that all multimedia protocol tunneling tools strive to achieve is *unobservability*. A covert channel is deemed unobservable if an adversary is unable to distinguish network flows that carry a covert channel from those that do not [66]. In practice, tools that provide a high degree of

unobservability can prevent an adversary from flagging a large fraction of covert flows unless they risk to erroneously flag a large amount of regular traffic as covert flows.

The major challenge faced during the development of multimedia protocol tunneling tools is that of achieving a proper balance between unobservability and performance. While the throughput of the covert channel can be increased by growing the amount of covert data embedded within the carrier protocol, the embedding of additional data may cause the traffic patterns generated by the carrier protocol to deviate from the typical patterns generated when transmitting legitimate data. Ergo, we face conflicting requirements when designing protocol tunneling tools that simultaneously feature high-throughput and unobservability. On the one hand, the traffic generated by these tools should be able to resist against the detection efforts of sophisticated network adversaries. If not, it is likely that citizens trusting the unobservability properties of these channels will end up being prosecuted. On the other hand, it is desirable that the covert channels generated by such tools achieve a sufficient throughput for accommodating typical tasks, e.g. web browsing. This thesis aims to study the balance between the unobservability and performance achieved by multimedia protocol tunneling tools, a subject of important practical implications which has not been thoroughly addressed in the past.

The remainder of this chapter is organized as follows. Section 1.1 describes the necessary background on the evolution of network covert channels for the purpose of censorship-circumvention, and describes encrypted traffic analysis techniques that may be applied to the detection of such covert channels. Section 1.2 details the contributions of this thesis. Lastly, Section 1.3 presents our conclusions and discusses directions for future work.

1.1 Background

In this section, we provide the necessary background to understand and discuss the challenges of building unobservable covert channels for evading Internet censorship. First, Section 1.1.1 describes the way network covert channels can be used to bypass typical Internet censorship mechanisms enforced by state-level actors. Then, Section 1.1.2 covers the background on state-of-the-art approaches for the creation of network covert channels, and Section 1.1.3 provides a deeper look at the properties of the protocol tunneling approach. Section 1.1.4 details our previous efforts on the design and implementation of a multimedia protocol tunneling system capable of transmitting Internet traffic through videoconferencing streams. Lastly, Section 1.1.5 details current traffic analysis techniques and shows how these can be re-purposed for the identification of obfuscated covert traffic in computer networks.

1.1.1 Network Covert Channels for Internet Censorship Circumvention

Network covert channels allow for the stealthy transmission of sensitive data through an apparently innocuous carrier medium [226]. In the context of Internet censorship circumvention, covert channels have become a fundamental building block of the design of multiple censorship-resistant communication tools that enable users to freely access and share information on the Internet [91, 192]. The general communication model of a censorship-resistant communication tool based on network covert channels is illustrated in Figure 1.1. Below, we describe this model in light of a typical Internet censorship apparatus operated by state-level actors and present a set of techniques that may be used by adversaries attempting the detection of covert channels.

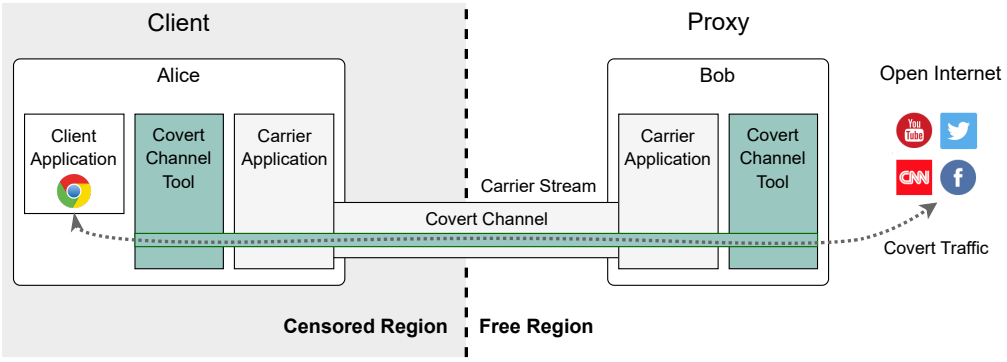


Figure 1.1: General model of a covert channel tool for censorship-resistant Internet browsing.

Censorship scenario: A state-level Internet censorship scenario typically considers two network regions: (i) the *censored* region, and (ii) the *free* region. The censored region is controlled by a state-level adversary which is able to observe, store, interfere with, and analyze all the network flows within its jurisdiction, and block the access to remote Internet services, such as CNN or Twitter, by the residents in the censored region. Censorship policies can be based on the IP address or the domain name of the target destination, the blacklisting of content (e.g., through keyword and image filtering), or the protocol used in the communication (e.g., BitTorrent or Tor). The free region consists of the Internet portion that is considered not to be under the control of the adversary or any other entity that aims to block Internet communications.

Covert channel-based evasion mechanism: Covert channels aim at enabling clients located within a censored region (e.g., Alice) to bypass the Internet communication constraints enforced by the adversary by leveraging the cooperation of a proxy located in the free region (Bob), and a carrier application featuring an encrypted communication protocol (e.g., Skype) whose traffic the adversary authorizes to cross the boundaries of the censored region. Note that, considering how instrumental many applications (e.g., media streaming services) are for the tissue of economic and social interactions within censored regions, the costs of blocking popular applications can produce overwhelming collateral damage to the country's own sustainability [58]. To create a covert tunnel through the data stream generated by the carrier application, Alice and Bob must run a special software on their local computers. This software will be responsible for embedding covert data within the seemingly legitimate data stream (e.g., by encoding covert data in images sent through a Skype video call [10]). This tunnel will allow Alice to contact remote hosts on the open Internet, for instance, by tunneling the traffic generated by her web browser.

Identifying and thwarting covert channels: To detect the operation of a covert channel tool, the adversary can make use of deep packet inspection for pinpointing traffic indicators that lead to the identification of a covert channel. To increase its chances of detection, the adversary may also apply statistical traffic analysis techniques over the collected network traces. The adversary may also launch indiscriminate active network attacks aimed at perturbing the correct behavior of covert channels lurking under seemingly legitimate carrier streams while ensuring that legitimate streams maintain a reasonable quality.

With regards to Internet censorship circumvention, the purpose of network covert channels is twofold: (i) to enable a client located within a censored region to communicate with services located in the free Internet while (ii) ensuring that the existence of such communication cannot be easily detected by an adversary. In other words, the main goal of network covert channels

is to ensure *unobservability*, that is, an adversary able to inspect any number of network flows should be unable to accurately identify those that carry a covert channel. In the next section, we describe several advancements in the creation of network covert channels whose design is tailored for the purpose of Internet censorship circumvention.

1.1.2 The Evolution of Censorship-Resistant Network Covert Channels

Over time, multiple research efforts have been conducted with the aim of designing covert channels on the Internet. Traditional techniques for the generation of network covert channels hold on the encoding of data in the inter-arrival timing patterns of packets, or on concealing data within reserved/unused packet header fields [205]. However, such methods have been progressively defeated by countermeasures based on traffic normalization mechanisms. Briefly, traffic normalizers are composed of network gateways that are able to modify forwarded traffic by, for instance, clearing bits of network protocols' headers or arbitrarily delaying the transmission of packets. Such mechanisms have thus been found to successfully prevent the correct behavior of a wide range of covert channels [226]. Additionally, and due to the reduced amount of covert data that can be embedded in these covert channels, these generally achieve a rather low throughput.

Recent literature describes several novel approaches for the creation of covert channels for circumventing censorship [91, 192]. The majority of such approaches stem from the need to create high-bandwidth communication channels that allow users to perform common Internet tasks such as web browsing or bulk data downloads. Next, we describe a range of proposals for the creation of unobservable network covert channels which exhibit different degrees of complexity.

Protocol randomization: This technique holds on to the manipulation of covert traffic so as to make it seem random and fool an adversary's protocol blacklist. For instance, Obfsproxy [46] and ScrambleSuit [208] respectively encrypt and randomize Tor's [47] network traffic patterns in order to evade an adversary's firewall that targets unmodified Tor traffic. Yet, entropy tests have been successfully used for distinguishing regular TLS traffic from Obfsproxy encrypted traffic [200]. Moreover, by transforming a given application-level protocol's traffic into some unknown protocol, this approach fails to evade an adversary that performs protocol whitelisting. This fact has sparked the development of improved traffic shaping approaches that aim to mimic the traffic patterns of known protocols allowed to cross censors' firewalls.

Protocol imitation: The key idea underlying protocol imitation tools is that of shaping covert traffic to mimic the behavior of a popular network protocol that is not blocked by censors. For instance, StegoTorus [204] steganographically conceals chops of Tor traffic on the messages of a cover protocol, while SkypeMorph [129] and CensorSpoofer [201] mimic the statistical properties of video and VoIP calls, respectively. In its turn, Format-Transforming Encryption (FTE) [51] produces ciphertexts that match the content definition of some target protocol. However, due to the difficulties of mimicking all aspects of a protocol, the former tools are vulnerable to several attacks [66, 80] while FTE can be detected through the use of entropy tests [200]. Marionette [52] employs automata composition to control fine-grained aspects of mimicry. Still, candidates for imitation may be proprietary software, demanding its reverse engineering in order to build a model for imitation. This is a tedious effort which must be repeated for each software release. To raise the difficulty of detection by a censor, an updated branch of tools has been proposed to bridge the drawbacks of protocol imitation. Namely, protocol tunneling tools avoid the need to faithfully mimic a popular cover protocol by effectively using it to piggyback covert data.

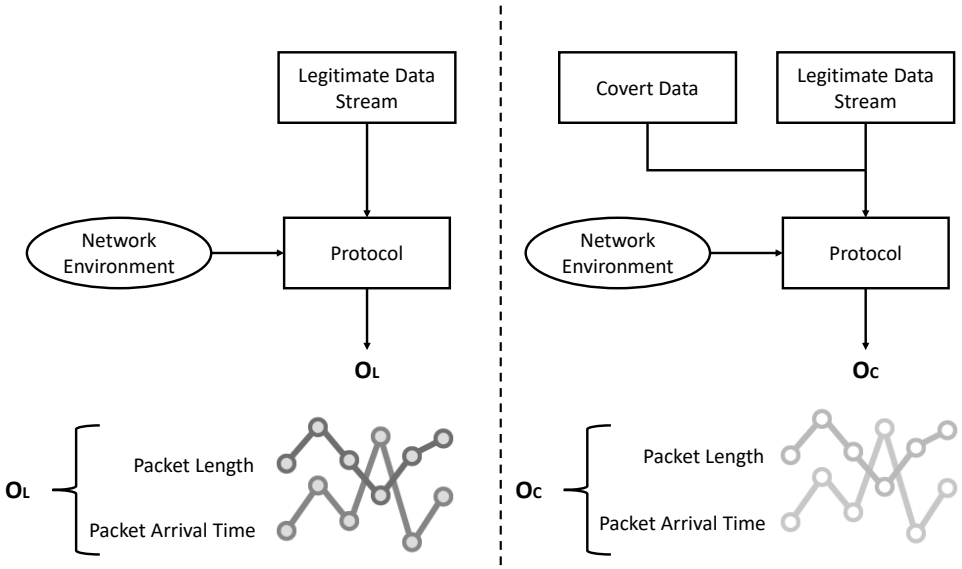


Figure 1.2: Overview of the protocol tunneling approach for the creation of covert channels.

Protocol tunneling: In this approach, the traffic produced by a covert channel is *tunneled* through a given protocol's application layer, i.e., covert traffic is encapsulated within a given cover protocol, e.g., Skype [226]. Such a technique enables protocol tunneling tools to directly deal with several cover protocol's intricacies which can be difficult to mimic. In *multimedia protocol tunneling* [82, 97, 105, 121], covert data is modulated into video and/or audio and is transmitted through the application layer of encrypted multimedia streaming applications such as Skype or YouTube live streams. Differently, SWEET [233] relays covert traffic through encrypted or steganography-protected email messages. CloudTransport [29] adopts a similar principle but makes use of public cloud storage services for covert message forwarding instead. *meek* [60] leverages domain fronting, the use of different domain names at different communication layers, to tunnel covert traffic to a given website over HTTPS connections while appearing to communicate with a different website. *Refraction networking* [81, 90] systems like Slitheen [23] are based on the deployment of special routers in the network path between a user and an allowed destination. These routers replace the leaf resources requested from overt websites with censored material while keeping covert traffic appear indistinguishable from allowed, uncensored traffic. Castle [71] and Rook [198] provide an alternative approach to exchange covert messages over Real-Time Strategy (RTS) games, by encoding covert data in valid game commands.

Next, we deliver a more detailed characterization of protocol tunneling and describe the rationale for the creation of unobservable network covert channels while leveraging this technique.

1.1.3 A Closer Look at Protocol Tunneling

As described in the previous section, protocol tunneling aims to create unobservable covert channels by embedding secret data through the application layer of encrypted protocols. Figure 1.2 depicts an overview of the protocol tunneling approach. The left-hand side of the figure shows the generation of a legitimate network traffic stream. We have that a legitimate data stream (e.g., a chat video stream) is transmitted over the network by a given data exchange protocol (e.g., a multimedia streaming protocol such as Skype) in a given network environment with a

particular set of characteristics (e.g., bandwidth, packet loss rate). The protocol generates a legitimate network traffic stream (O_L) according to the present network conditions and video input. Conversely, the right-hand side of the figure shows the generation of a network traffic stream that embeds a covert channel in a protocol with the same characteristics as the one previously described. Here, covert data (e.g., a text file) is modulated in order to match the expected content of the cover protocol's application layer (e.g., video content), and is combined with a legitimate chat video. Such an embedding may be accomplished, for instance, by overlaying a portion of each frame pertaining to a legitimate chat video with an image that encodes covert data. The protocol then generates a covert network traffic stream (O_C).

For ensuring the unobservability of covert channels, the designer of protocol tunneling tools must embed covert data into legitimate streams in such a way that the characteristics of a covert network stream (O_C) are similar to those of a model representing the expected distribution of legitimate traffic (O_L). Since these tools admit that the application protocols used as cover encrypt data in transit, the observation of network packets' payload does not convey any useful information for an adversary. Thus, both O_L and O_C are essentially comprised of a pair of timeseries: a timeseries of packet lengths and a timeseries of packet arrival times. For attempting the identification of a covert stream, an adversary may proceed as follows. First, the adversary must be able to understand which characteristics define legitimate traffic. A possible method for representing the characteristics of a wide range of legitimate traffic is to build a model based on the frequency distribution of packet lengths of multiple legitimate streams (O_L). Then, the adversary must decide whether a particular stream matches the model for legitimate traffic. Such a decision can be performed by employing, for instance, a similarity metric computed through the Kolmogorov-Smirnov test [117], to quantify the distance between the packet length frequency distribution of a given traffic sample and the reference legitimate traffic model.

While protocol tunneling is a conceptually simple approach, the successful implementation of this technique is far from trivial. Specifically, it has been shown that blindly tunneling covert data through the application layer of a cover protocol does not suffice to resist detection against an adversary able to inspect the characteristics of the cover application protocols' traffic. A particular example consists in the detection of FreeWave [82], where the transmission of audio-modulated data through Skype VoIP calls can be distinguished from the transmission of actual human speech [66]. Incidentally, multimedia protocol tunneling schemes have recently received renewed attention due to the introduction of multiple approaches for data modulation aimed at raising the difficulty for adversaries to identify covert transmissions [97, 105, 121].

Next, we describe our past efforts on developing DeltaShaper, a multimedia protocol tunneling tool that leverages Skype video calls to embed covert Internet traffic, and that can be parameterized to provide different levels of resistance against traffic analysis.

1.1.4 Tunneling TCP/IP Traffic over Videoconferencing Streams

In the past, our analysis over the existing multimedia protocol tunneling landscape led up to the development of DeltaShaper [10], a censorship-resistant tool that supports bi-directional TCP/IP tunneling over videoconferencing Skype streams.

Figure 1.3 illustrates the operation of DeltaShaper. On the sending side, the transmitter receives the payload and encodes it into colored matrices (payload frames) that are overlayed on top of a video stream that is fed to Skype using a virtual camera interface. Skype transmits

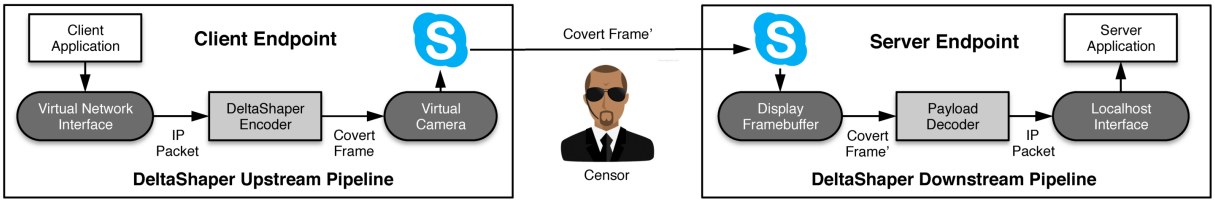


Figure 1.3: Architecture of DeltaShaper.

this video to the remote Skype instance and the received stream is captured from the Skype video buffer. A decoder then extracts the payload from the video stream and delivers it to the application. The same procedure is applied at both endpoints of a Skype call. To make the system as general as possible, the architecture exposes a data-link level protocol to the upper layers, such that an IP packet can be accepted, encoded, decoded, and delivered remotely. As a result, DeltaShaper allows for the execution of traditional TCP/IP applications that can tolerate low throughput and high latency links, and which can be used to cover different needs exhibited by users actively evading censorship. Some practical examples include fetching simple webpages, establishing remote SSH tunnels, or exchanging e-mail messages.

DeltaShaper features two major distinguishing aspects. First, DeltaShaper is able to trade-off unobservability and performance in a controlled fashion. Experimentally, we verified that by carefully tuning the composition of the payload frames (e.g., by changing their size or the rate at which they are rotated to convey new data), it was possible to produce network traffic akin to that generated when placing legitimate Skype calls. Moreover, we geared the composition of payload frames to maximize the achievable throughput while ensuring unobservability. This analysis was performed by comparing DeltaShaper streams with legitimate Skype streams with the help of a similarity metric computed over the packet sizes and inter-packet delays distributions of both types of streams. Second, DeltaShaper is able to automatically adjust the composition of the payload frame in face of multiple network conditions. As per our evaluation, placing DeltaShaper calls in adverse network conditions could demand an adjustment of the payload configuration to maintain unobservability, e.g., reducing the payload frame rate in reduced bandwidth scenarios. To do this, DeltaShaper periodically selects payload frame parameters according to pre-recorded traffic signatures collected in different network conditions.

Our experience during the development of DeltaShaper showed that the comprehensive evaluation of unobservability has important practical consequences for the users of censorship-circumvention tools. In particular, since multimedia protocol tunneling implementations (like our own) carefully calibrate a number of parameters to meet a given traffic analysis resistance threshold, the evaluation of unobservability is by itself a fundamental component for the successful deployment of circumvention tools. Until then, the unobservability evaluation of protocol tunneling tools had only been performed in an *ad hoc* fashion, resorting to classifiers based on different similarity metrics. This fact is of particular relevance since the unobservability guarantees of a tool can turn out to be rather optimistic if the evaluation is conducted using a classifier that achieves only a mediocre performance in detecting covert channels. In practice, this means that a tool is prone to be detected by an adversary that can employ a better classifier.

In the next section, we provide an overview of emerging traffic analysis techniques that have been instrumental for enabling different network security-focused applications, and which may also be used for disclosing the activity of increasingly sophisticated protocol tunneling tools.

1.1.5 Traffic Analysis for Network Security Applications

The process of encrypting data in the network has become widespread so as to provide an answer to multiple privacy concerns. However, the use of encryption has also reduced the ability of network operators to monitor their network infrastructure due to the inability to read packet contents. Nevertheless, encrypted network traffic can still leak interesting information to an observer in the network [211]. In fact, the ability to infer information from encrypted flows is of interest in multiple network security-focused tasks [54, 196], ranging from the detection of malware [3, 100, 122] to intrusion detection [30, 128] and the identification of potentially vulnerable applications [189]. Unfortunately, traffic analysis also allows ill-intended network eavesdroppers to breach the security of privacy-sensitive applications. Some examples include the ability to derive approximate transcripts of encrypted VoIP conversations [206] or the identification of websites visited through anonymity networks [74, 150, 165].

Traffic analysis workflow: The literature is rich in a variety of techniques to infer useful information from encrypted traffic flows. A prominent approach for classifying flows according to their traffic patterns relies on the use of statistical traffic analysis methodologies [145, 196]. A typical statistical traffic analysis workflow consists of four main steps:

1. Data acquisition: Traces of flows of interest must be acquired in network vantage points. These traces may correspond both to raw packet data, or to summarized data resulting from the pre-processing of network flows. Traffic samples may be collected in a controlled fashion so as to ensure their correct categorical labelling, e.g., as benign or malicious traffic samples, or kept unlabeled for later analysis.

2. Feature extraction: To infer details about the content of a particular network flow, we must identify a number of characteristics (or features) of traffic that can better describe the flow. Due to the use of encryption, these features typically correspond to the summarization of the sequences of packet lengths or packet inter-arrival times of a flow into a reduced set of features (e.g., descriptive statistics). However, it is also possible to extract meaningful information from the observation of the encrypted payload itself, or from the observation of ancillary information such as protocol handshakes, ACK traffic, and unencrypted packet header values.

3. Model training: Upon finding a set of features that can properly describe a particular kind of network flows, it is then possible to build traffic *classifiers*, i.e., predictive models for some classes of traffic. While these models can be based on simple statistical tests like Kolmogorov-Smirnov [10, 129], recent traffic analysis approaches have made extensive use of Machine Learning (ML) algorithms for building increasingly accurate traffic classifiers [131, 145, 196]. In general, ML algorithms can be split into two different realms: *supervised* and *unsupervised* [73]. In supervised learning, models are trained using labeled data and aim to find a function that can predict the associated class label of a given sample. Examples of such algorithms comprise decision trees, support vector machines (SVM), and neural networks. In contrast, unsupervised learning algorithms analyze unlabeled input data to infer patterns in such data. In particular, these algorithms group a set of samples into multiple *clusters* so that samples within the same cluster have high similarity, but are very dissimilar to samples in other clusters. Examples of unsupervised ML algorithms include k-Means and DBSCAN. In addition, *outlier detection* algorithms attempt to build a model that is trained only on data labeled as "normal", and can then be used to find anomalies, i.e., samples which do not correspond to the definition of "normal". Examples of such algorithms are one-class SVM and autoencoders.

4. Flow classification: Once the statistical model has been trained, newly observed flows can be classified or clustered according to the chosen model.

In the context of this thesis, we can observe that the traffic analysis tooling presented above is also available to network adversaries interested in detecting obfuscated Internet traffic. We now describe past attempts for the identification of Internet covert channels for censorship circumvention through the use of increasingly sophisticated statistical traffic analysis techniques.

Identification of obfuscated covert traffic: As described in Section 1.1.3, protocol tunneling aims to generate unobservable covert channels on the Internet by embedding secret data through the application layer of encrypted protocols. While this approach raises the difficulty of an adversary to identify network flows carrying covert data, it was still not well understood how the unobservability of state-of-the-art protocol tunneling tools holds against recent statistical traffic analysis techniques. In fact, we may find in the literature two previous efforts which have succeeded at detecting the presence of obfuscated traffic generated by such tools:

Detection via statistical tests: Geddes et al. [66] were the first to analyze the unobservability properties of FreeWave [82], a multimedia protocol tunneling tool that modulates covert data into the audio of Skype VoIP calls. In their study, Geddes et al. analyzed the distribution of packet lengths generated by Skype when used to carry covert data. First, the study corroborated the claims of FreeWave's authors by verifying that the tool could not be accurately detected by the inspection of basic traffic features such as the minimum/average/maximum values of packet lengths. Second, and posing a more pressing concern, the study revealed that a closer look into alternative statistics computed from packet lengths was sufficient for the detection of covert traffic generated by FreeWave. Indeed, a simple threshold-based classifier based on the standard deviation of packet lengths allowed a passive adversary to efficiently detect FreeWave flows.

Detection via machine learning: Wang et al. [200] introduced the application of ML techniques to the problem of detecting covert Tor traffic tunneled over HTTPS requests, a technique implemented by *meek* [60]. Wang et al. observed that *meek*'s TCP ACK traffic is distinct from that of regular TLS connections. Then, they manually crafted a feature set from three tailored batches of traffic characteristics expected to maximize the success in identifying *meek*'s flows. The first batch comprises entropy-based characteristics, where the Shannon entropy of packets in each direction of a flow are used as features. The second batch of features is comprised of timing characteristics based on the interval between TCP ACK packets (in the same direction) of a given flow. The third batch is based on packet-header features. Finally, Wang et al. have conducted experiments resorting to the k-Nearest Neighbors, Naïve Bayes and CART decision tree classifiers and found out the latter to provide the best accuracy in identifying covert flows.

Apart from the two aforementioned studies, a number of authors have proposed multiple protocol tunneling designs and evaluated their resistance against traffic analysis. However, the existing evaluation approaches reported in the literature had so far been confined to the inspection of limited feature sets using a narrow scope of supervised classification techniques. For instance, the unobservability assessment of systems such as Facet [105], CovertCast [121], or DeltaShaper [10] solely relies on statistical tests based on the frequency distribution of packets lengths and/or inter-packet delays. These assessments ignore the latest advances in ML-based traffic analysis, performing an evaluation relying on rather simplistic and outdated classifiers. This fact casts doubt on whether the current procedures took forth to gauge the unobservability properties of protocol tunneling systems are robust against current traffic analysis techniques.

1.2 Contributions

This thesis focuses on the development of tools that allow Internet users to retain the security and privacy of their communications while evading the tight Internet control apparatus maintained by powerful network adversaries. In particular, it concentrates on the study and design of tools that surreptitiously encode information within popular multimedia exchange protocols so as to provide users with a secure medium for unrestricted access to information online. To evaluate the security of these tools, we switch hats to play the part of an adversary aimed at detecting or subverting the operation of such tools. Motivated by the above considerations, we address the following questions in this thesis:

- **Q1:** *Can network adversaries detect existing multimedia protocol tunneling tools by making use of ML-based traffic analysis techniques?*
- **Q2:** *Can network adversaries employ ML-based traffic analysis techniques to identify the presence of covert channels in large-scale high speed networks in an efficient fashion?*
- **Q3:** *Is it possible to build network covert channels over multimedia streams that can both resist against ML-based traffic analysis techniques and achieve a high throughput?*

Next, we detail the main contributions of this thesis. Our exposition is based on the description of the main results of a set of three representative publications, where each publication tackled a specific research question among the ones presented above.

1.2.1 (Q1) Effective ML-based Detection of Multimedia Protocol Tunneling

In the past, the unobservability evaluation of multimedia protocol tunneling tools resorted to *ad-hoc* similarity-based classifiers. To better understand whether these tools are robust against traffic analysis, we conducted the first extensive experimental study over the unobservability properties of three state-of-the-art multimedia protocol tunneling tools: Facet [105], CovertCast [121], and DeltaShaper [10]. In this study, we started by comparing the performance of three different similarity-based classifiers which had been proposed earlier for the unobservability evaluation of the aforementioned tools. Further, we tested the unobservability of the same covert channels against a number of sophisticated traffic analysis techniques based on machine learning. In particular, we made use of three different kinds of machine learning algorithms: a) supervised (CART, Random Forest, and XGBoost); b) unsupervised (Isolation Forest), and; c) outlier detection (One-class SVM and Autoencoder Neural Networks).

We highlight three main findings of our study. First, we found that *ad-hoc* similarity-based classifiers were not generally able to detect covert channels with high accuracy. For instance, the better performing of such classifiers, based on the χ^2 test, would mistakenly flag 45% of legitimate network flows when attempting to flag 90% of all Facet covert channels. An exception was CovertCast, whose flows were accurately detected when resorting to existing similarity-based classifiers. Second, we found that decision tree-based classifiers are extremely effective at detecting traffic generated by existing multimedia protocol tunneling tools. For instance, we found that XGBoost is able to flag 90% of all Facet covert channels while mistakenly flagging only 2% legitimate network flows. Third, our findings suggested that the use of unsupervised learning and outlier detection techniques lead to a significant deterioration of classification accuracy when

compared to supervised approaches, suggesting that the existence of manually labeled samples is a requirement for the successful detection of multimedia protocol tunneling tools.

Our study showed that multimedia protocol tunneling tools can be detected with high accuracy by passively monitoring and analyzing traffic patterns resorting to ML algorithms. If the deployment of such traffic analysis mechanisms becomes affordable and widespread across the large-scale networks managed by state-level adversaries, it is possible to conjecture that censors will get the lead on the censorship arms race.

The results of our study were published at the USENIX Security 2018 conference [12]. This publication is included as Chapter 2 of this thesis.

1.2.2 (Q2) Efficient Flow Classification for ML-based Network Security Tasks

Determined state-level censors are expected to deploy increasingly sophisticated traffic analysis mechanisms to uncover the presence of covert traffic in the network. However, due to the sheer volume of traffic, collecting and analyzing live network flows can be quite challenging in high speed multi-gigabit networks. An emerging trend in network security consists in the adoption of programmable switches for performing various security tasks in large-scale, high speed networks, with the goal of decreasing reaction times to threats and reducing costs associated with equivalent centralized server-based infrastructures. However, existing solutions cannot accommodate ML tasks that perform targeted flow classification based on packet size or inter-packet frequency distributions without imposing considerable overheads in storage, communication, and computation. In particular, existing approaches [102, 215] cannot accommodate ML tasks like the one described in the previous section for detecting multimedia protocol tunneling tools.

To understand whether adversaries can deploy ML-based traffic analysis machinery at scale, we investigated the capabilities of modern programmable switches to efficiently process and analyze packet distributions. Our research efforts culminated with the development of FlowLens, a system that leverages programmable switches to collect packet distributions at line speed and that supports the efficient machine learning-based classification of flows on the switches. To cope with the resource constraints of programmable switches, FlowLens computes a memory-efficient representation of each flow's relevant features, named *flow marker*, which contains enough information to perform accurate flow classification. Through an automatic profiler, FlowLens is able to generate flow markers that are specifically tailored for a given ML task, ensuring a reasonable balance between the size of the flow marker and the accuracy of flow classification. FlowLens also implements a data structure, named flow marker accumulator, which is responsible for collecting flow markers as flows cross a programmable switch. Periodically, flow markers are used to classify flows directly on the switching device.

Our experiments on a programmable Barefoot Tofino switch have shown that FlowLens achieves considerable savings in storage and communication in contrast with typical traffic analysis infrastructures. In particular, FlowLens was able to increase the amount of simultaneously measured flows by an order of magnitude when detecting botnet traffic or performing website fingerprinting. Additionally, FlowLens achieves an increase of two orders of magnitude in monitoring capacity when accurately detecting the traffic generated by multimedia protocol tunneling tools. These results suggest that scaling current ML-based traffic analysis machinery is in the reach of contemporary network adversaries and that there is a pressing need for the design of protocol tunneling tools that can better mix with legitimate traffic, ideally with a very limited effect over the packet distributions of network flows.

The results of our work were accepted for publication at the NDSS 2021 conference [15]. This publication is included as Chapter 3 of this thesis.

1.2.3 (Q3) Evading Censorship by Parasitizing on WebRTC

The increasing sophistication of ML-based traffic analysis systems such as FlowLens has set a bleak scenario for the ability of multimedia protocol tunneling tools to successfully breach through state-level firewalls. While the design of these tools encompasses different mechanisms for embedding covert data within media streams, they fail to generate covert channels that simultaneously provide a high-throughput and strong unobservability guarantees. One possible explanation for this fact is that they were exclusively developed over proprietary software such as Skype. By having no access to the source code of media streaming applications, these carrier applications are generally treated as a black-box. On the one hand, multimedia protocol tools modulate covert data as an audio or video signal, in a trial-and-error fashion, towards making the traffic generated by the carrier application resemble legitimate traffic when transmitting covert content. Besides, the lossy compression performed by multimedia encoders forces modulation approaches to introduce redundancy to enable the correct decoding of covert data at the receiver's endpoint.

The above insight prompted us to search for other popular carrier multimedia streaming applications whose source code was available for modification, and which could allow us to develop a new efficient and traffic analysis-resistant covert data encoding technique. During our search, we came across WebRTC, a W3C standardization initiative for protocols and APIs for enabling secure real-time communication between web browsers. As of today, all major browsers include built-in WebRTC implementations. This has sparked the adoption of WebRTC across numerous services that integrate real-time communication capabilities, providing a range of potential carrier applications for performing covert data transmissions. It is worth noting that placing a WebRTC call is similar to placing a video call on a typical web streaming application. Additionally, WebRTC traffic is typically encrypted in transit between any two communicating endpoints, in a peer-to-peer fashion. These properties make WebRTC an attractive target for building improved network covert channels that can be made widely available.

Our analysis of WebRTC led to the design of Protozoa, a new censorship-resistant tool that instruments the video pipeline of WebRTC-enabled browsers to replace the contents of encoded video frames with covert data. Differently from existing protocol tunneling tools, we propose a covert data embedding procedure within the video encoding pipeline which sits immediately after the compression of an original video signal. Specifically, Protozoa replaces the bits that result from compressing legitimate video frames with bits of the covert payload. The resulting frames are then encrypted and transmitted over the network. Afterward, the receiver can decrypt the frames and extract the covert payload. Additionally, Protozoa injects valid pre-recorded encoded frames at the receiver's WebRTC media decoder to avoid decoding errors and the triggering of congestion control mechanisms that could reveal the presence of a covert channel.

Protozoa's encoding scheme brings several advantages. First, the throughput of the covert channel grows considerably, to the order of 1Mbps. This allows, as an example, for the covert transmission of YouTube video streams. This performance gain occurs because all bits of the compressed video frames can be used to encode covert data, thereby increasing the capacity of the channel. Second, we experimented Protozoa in the wild and observed that it allows for the evasion of censorship filters in multiple regions of the world which are known to experience

Internet censorship. Interestingly, we verified that the profusion of different WebRTC services makes Protozoa able to evade censorship, even when particular WebRTC services are blocked (for instance, `appr.tc` was blocked within China, but `whereby.com` was not). Third, the video streams generated by Protozoa are able to preserve the traffic characteristics of a legitimate transmission because the covert payload bits are replaced in the same amount as the number of bits in the original compressed video frame. This undermines the effectiveness of modern traffic analysis techniques [12]. By taking the above advantages combined, we showed that Protozoa is able to generate fast and unobservable covert channels over WebRTC multimedia streams.

Our results were published at the CCS 2020 conference [14]. This publication is included as Chapter 4 of this thesis.

1.2.4 Ramifications and Other Collaborations

In addition to the main contributions presented in this document, our research on the application of network covert channels to evade Internet censorship led us to collaborate in other related dimensions of the same problem. Specifically, we studied the effectiveness of the keyword filtering mechanisms employed by the Great Firewall of China.

The Great Firewall of China (GFW) is a set of technologies that prevents Chinese citizens from accessing online content deemed objectionable by the Chinese government. While the technical capabilities of the GFW have matured over the last decades, one of its main mechanisms for enforcing censorship rests on the search for forbidden keywords in unencrypted packet streams. When the GFW detects such keywords, it terminates the offending stream and blocks traffic between the same two hosts involved in communication for a few minutes.

We studied the effectiveness of the GFW to filter keywords included in HTTP requests, resorting to different lists of potentially blocked terms, curated between 2014 and 2020. We found that over 86% of the keywords targeted by the GFW were replaced since 2014, and that the GFW continuously updates its blocking list to include terms related to recent sensitive topics, such as the coronavirus outbreak. Additionally, we asserted that the GFW targets different sets of keywords when inspecting hostnames, page names, and search queries. Finer-grained experiments further revealed a number of instances where forbidden keywords do not trigger the GFW blocking mechanisms, e.g., the content of some HTTP headers is ignored.

A manuscript describing the work developed in the context of this collaboration was accepted for publication at the WebConf 2021 conference [159].

1.2.5 Summary of Contributions

In summary, the primary contributions of this thesis are the following:

- The finding that existing multimedia protocol tunneling tools can be detected by network adversaries that make use of ML-based traffic analysis techniques.
- The design of a packet frequency distribution compression and processing scheme within programmable switches. This set off the design of FlowLens, a system capable of conducting multiple ML-based traffic analysis tasks in high speed large-scale networks. FlowLens is able to detect the activity of multimedia protocol tunneling tools in such networks.

- The design of a network covert channel based on the instrumentation of WebRTC's video encoding pipeline. This culminated in the design of Protozoa, a tool that creates high speed covert channels which can evade current ML-based traffic analysis techniques.

1.2.6 Summary of Results

Considering the above contributions, the main results of this thesis are the following:

- An implementation [13] of a framework for the assessment of unobservability which can be used to guide the development and security analysis of novel network covert channels.
- An implementation [16] of a system that uses programmable switches to efficiently support multi-purpose ML-based security applications in high speed networks, and its evaluation on different scenarios, including the detection of multimedia protocol tunneling tools.
- An implementation [8] of a tool that allows for censorship-resistant Internet communications by embedding covert data on WebRTC media streams, and its performance evaluation through an experimental testbed that emulates realistic usage scenarios and workloads.

1.3 Conclusions and Future Work

In this thesis, we focus on the balance between the unobservability and performance achieved by multimedia protocol tunneling, a technique that has given birth to a set of tools that allow Internet users to retain the security and privacy of their communications while evading the tight Internet surveillance and control apparatus maintained by powerful network adversaries. Despite the reported advances in improving and scaling the deployment of detection machinery, we conclude that it is possible to build network covert channels over multimedia streams that simultaneously offer high-throughput and strong resistance to traffic analysis. Before reaching this conclusion, we worked in two complementary directions: (i) towards devising novel frameworks and deploying new systems aimed at detecting the activity of multimedia protocol tunneling tools (Chapters 2 and 3), and (ii) towards developing and scaling new tools that surreptitiously encode information within popular multimedia exchange protocols (Chapter 4). Next, we draw conclusions on three major outcomes of this thesis and point to future directions.

First, we show that a category of classifiers typically used for performing the unobservability assessment of multimedia protocol tunneling tools provides optimistic security guarantees. As detailed in our experimental study [12], the accuracy obtained by machine learning-based techniques largely surpasses that obtained by similarity-based classifiers in the task of detecting multimedia protocol tunneling tools. However, while the classifiers proposed in our unobservability assessment framework perform the most comprehensive analysis of flow features for the purpose of multimedia protocol tunneling detection to date, it is unknown whether the use of increasingly sophisticated classifiers can significantly challenge the unobservability claims of current tools. An interesting direction for future work is that of measuring information leakage in the traffic produced by protocol tunneling tools, grounded in theoretical schemes like WeFDE [103] or F-BLEAU [41], towards developing a classifier-agnostic unobservability evaluation framework. However, one particular challenge involved in using such approaches for estimating information leakage is that the estimation process is tightly coupled with the set of features being used to

describe the different classes of network traffic [164]. This suggests that feature engineering efforts and the exploration of alternative feature spaces may be required to conduct a better estimation of the security of multimedia protocol tunneling tools.

Second, we ascertain that knowledgeable network adversaries possess the technical means to efficiently carry out a range of security-focused tasks based on machine learning at the core of their network infrastructure. In particular, our work on FlowLens [15] showed that programmable switches can be used to facilitate the deployment of traffic classifiers that are able to perform the accurate detection of covert channels generated by multimedia protocol tunneling tools. One possible direction for future work is that of exploring ways to enrich FlowLens' analysis of packet frequency distributions with other properties derived from packet contents, such as the estimation of Shannon entropy or the ability to keep track of specific packet headers. This would not only enable the application of FlowLens to other ML-based tasks comprising the detection of network covert channels [200], but also to other ML-based tasks aimed at enhancing the QoS of business-level applications [196].

Thirdly, our research efforts on Protozoa [14] show that having control over the innards of the multimedia coding pipelines used by carrier applications opens the possibility for the development of more elaborate covert data embedding mechanisms, e.g., allowing for increased covert channel bandwidth while minimizing side-effects on the traffic patterns generated by the application when tunneling covert data. A possible direction for further increasing the throughput offered by Protozoa is to establish covert channels over high-definition video streams which can offer a larger amount of covert data bandwidth. Additionally, our ongoing work [9] aims to augment the scalability of Protozoa by introducing a new system design that will allow isolated users to reach out to remote proxies through a multi-hop chain of trusted nodes through the establishment of end-to-end covert channels. Future work should also consider the deployment of *WebRTC gateways*, application-controlled middleboxes which may intercept and enforce the validation of video data, possibly thwarting Protozoa's ability to transfer covert data. It will be interesting to explore whether Protozoa can be used to establish covert channels when faced with the operation of such interception devices, e.g., by making use of steganography techniques [214].

The work developed during this thesis also enables us to envision future work in interrelated directions. So far, the evaluation of proxy-based censorship-resistant tools, including protocol tunneling approaches, is performed on a per-flow basis. While the network characteristics of a single flow may not be sufficient to reveal the presence of a covert data transmission, it is possible that an adversary can profile the traffic patterns of users over long periods of time and find deviations that enable it to pinpoint potential users of censorship-circumvention tools. Specifically, deviations from a user's profile can take multiple forms, including the existence of uncommon connection times, frequency, and duration, or the establishment of connections towards atypical communication endpoints. Although this threat to proxy-based circumvention tools is well-known [9, 14], its viability is yet to be fully understood. In addition, we have seen that protocol tunneling tools inject covert data within the communication channels of popular applications like video-conferencing services. Oftentimes, the applications and protocols used for surreptitiously transmitting data are proprietary, leverage the network infrastructure of third-party companies to some extent, or are bound by Terms & Conditions which are largely ignored by the developers of circumvention tools. One other possible direction for future work is that of revisiting earlier protocol mimicking approaches and explore the use of generative adversarial networks to build improved circumvention approaches that generate network traffic akin to that of popular Internet services, without interacting with such services' resources themselves.

II Publications

List of Publications

The work presented in this thesis has contributed to three main publications in top-tier venues (CORE A*), listed below.

- Diogo Barradas, Nuno Santos, Luís Rodrigues. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. Proceedings of the 27th USENIX Security Symposium, Baltimore, MD, USA, August 2018
- Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando M. V. Ramos, André Madeira. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. Proceedings of the 27th Network and Distributed System Security Symposium, San Diego, CA, USA, February 2021
- Diogo Barradas, Nuno Santos, Luís Rodrigues, Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 2020

2

Effective Detection of Multimedia Protocol Tunneling using Machine Learning

Publication Data

Diogo Barradas, Nuno Santos, Luís Rodrigues. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. Proceedings of the 27th USENIX Security Symposium, Baltimore, MD, USA, August 2018

Abstract

Multimedia protocol tunneling enables the creation of covert channels by modulating data into the input of popular multimedia applications such as Skype. To be effective, protocol tunneling must be unobservable, i.e., an adversary should not be able to distinguish the streams that carry a covert channel from those that do not. However, existing multimedia protocol tunneling systems have been evaluated using ad hoc methods, which casts doubts on whether such systems are indeed secure, for instance, for censorship-resistant communication.

In this paper, we conduct an experimental study of the unobservability properties of three state of the art systems: Facet, CovertCast, and DeltaShaper. Our work unveils that previous claims regarding the unobservability of the covert channels produced by those tools were flawed and that existing machine learning techniques, namely those based on decision trees, can uncover the vast majority of those channels while incurring in comparatively lower false positive rates. We also explore the application of semi-supervised and unsupervised machine learning techniques. Our findings suggest that the existence of manually labeled samples is a requirement for the successful detection of covert channels.

The reproduction of this publication was slightly adapted to adhere to formatting requirements. The original version of this publication can be found at: `https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-barradas.pdf`.

2.1 Introduction

Multimedia protocol tunneling has emerged as a potentially effective technique to create covert channels which are difficult to identify. In a nutshell, this technique consists of encoding covert data into the video (and/or audio) channel of popular encrypted streaming applications such as Skype without requiring any changes to the carrier application. Systems such as Facet [105], CovertCast [121], and DeltaShaper [10] implement this technique, and introduce different approaches for data modulation that aim at raising the difficulty of an adversary to identify covert data transmissions.

An important property that all these systems strive to achieve is *unobservability*. A covert channel is deemed unobservable if an adversary that is able to scan any number of streams is not able to distinguish those that carry a covert channel from those that do not [66, 80]. Thus, an adversary aims at correctly detecting all streams that carry covert channels, among a set of genuine streams, as effectively as possible. In practice, a multimedia protocol tunneling system that provides a high degree of unobservability prevents an adversary from flagging a large fraction of covert flows (i.e., from attaining a high true positive rate) while flagging a low amount of regular traffic (i.e., while attaining a low false positive rate).

In spite of the efforts to build unobservable systems, the methodology currently employed for their evaluation raises concerns. To assess the unobservability of a system such as Facet, experiments are mounted in order to play regular traffic along with covert traffic, collect the resulting traces, and employ similarity-based classifiers (e.g., relying in the χ^2 similarity function) to determine whether covert traffic can be detected with a low number of false positives [105]. However, each system has been evaluated with a different classifier, making results hard to compare. Furthermore, those studies use just one among the many machine learning (ML) techniques available today. Yet, providing a common ground for assessing the unobservability of multimedia protocol tunneling systems is a relevant problem which, nevertheless, has been overlooked in the literature. Considering that such systems emerged from the need to circumvent Internet censorship, flawed systems may pose life-threatening risks to end-users, e.g., journalists that report news in extreme conditions may be prosecuted, imprisoned, or even murdered if covert channels are detected.

To fill this gap, our goal is to systematically assess the unobservability of existing systems against powerful adversaries making use of traffic analysis techniques based on ML. We aim at understanding which ML techniques are better suited for the purpose of detecting covert channels in multimedia streams and what are the limitations of such techniques. In particular, we seek to explore ML techniques which have yielded successful results when applied in other domains (e.g., Tor hidden services fingerprinting [74]), but have not yet been studied in the context of covert traffic detection.

In this paper, we present the first experimental study of the unobservability of covert channels produced by state-of-the-art multimedia protocol tunneling systems. We test three systems – Facet, CovertCast, and Deltashaper – using the original code provided by their maintainers. For our study, we take a systematic approach by investigating a spectrum of anomaly detection techniques, ranging from supervised, to semi-supervised and unsupervised, where for each category we explore different classifiers, and investigate the trade-offs involved in the ability to flag a large amount of covert channels while minimizing false positives. From our study, we highlight the following three main contributions.

First, our analysis reveals that some state-of-the-art systems are flawed. In particular, CovertCast flows can be detected with few false positives by an adversary, even when resorting to existing similarity-based classifiers. While the remaining systems exhibit different degrees of unobservability according to their parameterization, we show that none of the currently employed similarity-based classifiers can detect such channels without incurring in large numbers of false positives. We also conclude that one of the existing similarity-based classifiers – using χ^2 distance – consistently outperforms all others in the task of detecting covert channels.

Second, we show that ML techniques based on decision trees and some of their variants are extremely effective at detecting covert traffic with reduced false positive rates. For example, an adversary employing XGBoost would be able to flag 90% of all Facet traffic while erroneously flagging only 2% of legitimate connections. Moreover, the performance of such techniques is very high, meaning that the adversary is able to classify traffic in a few seconds, with a relatively low number of samples per training set, and taking a low memory footprint. Additionally, the use of decision tree-based techniques allows us to understand which traffic features are most important for detecting particular multimedia protocol tunneling systems. These findings suggest that, apart from their performance, decision tree-based techniques can provide meaningful insight into the inner workings of these systems and we propose that they should be used for assessing the unobservability of multimedia protocol tunneling systems in the future.

Third, we explore alternative ML approaches for the detection of covert channels when the adversary is assumed to be partially or totally deprived of labeled data. Our findings suggest that unsupervised learning techniques provide no advantage for the classification of multimedia protocol tunneling covert channels, while the application of semi-supervised learning techniques yields a significant fraction of false positives. However, we note that the performance of semi-supervised techniques can be significantly improved through the optimization of parameters or by providing algorithms with extra training data. The study of semi-supervised anomaly detection techniques with an ability to self-tune parameters can be a promising future direction of research which would enable adversaries to detect covert traffic while avoiding the burden of generating and manually label data.

We note that we synthesize legitimate and covert traffic samples in laboratory settings for creating our datasets. While this is a common approach for generating datasets for the type of unobservability assessment we conduct in this paper, it is possible that adversaries possessing a privileged position in the network can build a more accurate representation of traffic.

The remainder of our paper is organized as follows. Section 2.2 presents the methodology of our study. Section 2.3 presents the main findings of our study regarding the comparison of similarity-based classifiers. Section 2.4 presents the results obtained when assessing unobservability resorting to decision tree-based classifiers. Section 2.5 presents our first insights on using semi-supervised and unsupervised anomaly detection techniques for the identification of covert traffic. In Section 2.6, we discuss obtained results and we present the related work in Section 2.7. Lastly, we conclude our work in Section 2.8.

2.2 Methodology

This section introduces the systems we analyzed, our adversary model, and the experimental setup of our study.

2.2.1 Systems Under Analysis

Below, we describe three state-of-the-art approaches at multimedia protocol tunneling which serve as a basis for our study. We selected these systems because all of them encode data into video streams, and their code is publicly available for open testing. We note that although these systems have been conceived for the purpose of censorship circumvention, in practice, they may be used for other purposes, such as concealing criminal activity.

Facet [105] allows clients to watch arbitrary videos by replacing the audio and video feeds of Skype videocalls. To watch a video, clients contact a Facet server by sending it a message containing the desired video URL. Afterwards, the Facet server downloads the requested video and feeds its content to microphone and camera emulators. Then, the server places a videocall to the client transmitting the selected video and audio instead. Thus, clients are not required to install any software in order to use the system. For approximating the traffic patterns of regular videocalls, Facet re-samples the audio frequency and overlays the desired video in a fraction of each frame while the remaining frame area is filled up by a video resembling a typical videocall. Decreasing the area occupied by the concealed video translates into increased resistance against traffic analysis.

CovertCast [121] scrapes and modulates the content of web pages into images which are distributed via live-streaming platforms such as YouTube. Multiple clients can consume the data being transmitted in a particular live stream simultaneously. CovertCast modulates web content by encoding it into colored matrix images. A colored matrix is parameterized by a cell size (adjacent pixels with a given color), the number of bits encoded in each cell (represented with a color), and the rate at which a matrix containing new data is loaded. Clients scrape and demodulate the images served through the live stream extracting the desired web content.

DeltaShaper [10] differentiates itself from the previous systems in that it allows for tunneling arbitrary TCP/IP traffic. This is achieved by modulating covert data into images which are transmitted through a bi-directional Skype videocall. DeltaShaper follows a similar data encoding mechanism to that of CovertCast. However, and similarly to Facet, a colored matrix is overlayed in a fraction of the call screen, on top of a typical chat video running in the background. This overlay, named payload frame, can be carefully parameterized to provide different levels of resistance against traffic analysis. On call start, DeltaShaper undergoes a calibration phase for adjusting its encoding parameters according to the current network conditions in order to preserve unobservability.

2.2.2 Adversary Model

To study the unobservability properties of the aforementioned systems, we emulate a state-level adversary which will attempt to detect the traffic of multimedia protocol tunneling tools while resorting to different anomaly detection techniques. The providers of encrypted multimedia applications which are used as carriers for covert channels are not assumed to collude with the adversary. Thus, the adversary cannot simply demand application providers to decipher and disclose raw multimedia content which could be easily screened for the presence of covert data. The adversary is also assumed to be unable to control the software installed in the computers of end-users. However, domestic ISPs are assumed to cooperate with the adversary, enabling it to monitor, store and inspect all traffic flows crossing its borders.

An adversary faces an inherent trade-off between the ability to correctly detect a large amount of covert channels and to erroneously flag legitimate flows. Flagging legitimate flows as covert channels is something that the adversary wants to avoid in most practical settings. For example, a censor that aims at blocking flows containing covert channels may not be willing to block large fractions of legitimate calls, that are used daily by companies and business, as these calls may be key for the economy of the censor's regime [58]. Also, law-enforcement agencies may not be willing to risk to falsely flag legitimate actions of citizens as criminal activity.

2.2.3 Performance Metrics

In face of the previous observations, when comparing the different techniques we mainly use the following metrics: *true positive rate*, *false positive rate*, *accuracy*, and the *area under the ROC curve*. The True Positive Rate (TPR) measures the fraction of positive samples that are correctly identified as such, while the False Positive Rate (FPR) measures the proportion of negative samples erroneously classified as positive. Thus, adversaries will attempt to obtain a high TPR and a low FPR when performing covert traffic classification. Accuracy captures the fraction of correct labels output by the classifier among all predictions, and can be used as a summary of the classification performance since high accuracy implies a high true positive rate and a low false positive rate. The ROC curve plots the TPR against the FPR for the different possible cutout points for classifiers possessing adjustable internal thresholds. The area under the ROC curve (ROC AUC) [56] summarizes this trade-off. While a classifier outputting a random guess has an AUC=0.5, a perfect classifier would achieve an AUC=1, where the optimal point on the ROC curve is FPR=0 and TPR=1.

2.2.4 Experimental Setup

For conducting our study, we were required to analyze a number of network traces produced by the systems described in Section 2.2.1. For our testbed, we used two 64-bit Ubuntu 14.04.5 LTS virtual machines (VMs) provisioned with a 2.40GHz Intel Core2 Duo CPU and 8GB of RAM configured in a LAN setting. We used the *v4l2loopback* camera emulator and the *pulseaudio* sound server to feed video and audio to the carrier multimedia applications. The prototypes of the considered systems were obtained from their respective websites [11, 104, 120]. Due to the deprecation of Skype v4.3 and the incompatibility of *v4l2loopback* with the latest Skype v8.x desktop version, we have resorted to Skype for Web. For gathering the traffic samples generated by each system, we captured the network packets produced by the carrier multimedia streams for a duration of 60 seconds after a given covert channel has been established. The methodology we followed for gathering traffic samples has been commonly used in the literature since it allows for the unobservability analysis of covert channels while executing in steady-state. Next, we describe the methodology we followed for generating our covert and legitimate traffic datasets.

Facet: For building our covert video dataset, we collected 1000 YouTube videos from the YouTube-curated Top Shared and Liked playlist. The legitimate Skype video dataset consists of 1000 recorded live chat videos available on YouTube. We adapted the Facet prototype to sample three types of Facet transmissions, corresponding to scaling the covert videos on top of legitimate videos by a factor of 50%, 25% and 12.5% – the available prototype represents a proof-of-concept only capable of a (unmorphed) 100% scaling. Then, we gathered 1000 traffic samples for each scaling factor by combining a pair of legitimate and covert videos while following the audio and

video morphing techniques detailed in Facet's original description. To emulate legitimate Skype calls, we streamed the media comprising our legitimate Skype video dataset. The resolution of the camera emulator was set to 320x240. For gathering traffic samples, we used each of the available VMs as a Skype peer.

CovertCast: For building our legitimate live-streaming dataset, we crawled 200 live-streams included in the Live YouTube-curated list. Then, we generated 200 CovertCast live-streams by broadcasting several news websites already included in the available CovertCast prototype. The server component, responsible for scraping websites, was executed in one of our VMs and streamed modulated video frames to YouTube. We used a Windows laptop running Google Chrome as a CovertCast client. Each video was streamed with a 1280x720 resolution.

DeltaShaper: We emulated 300 legitimate bi-directional Skype calls by streaming a subset of our legitimate Skype video dataset. We gathered DeltaShaper traffic samples by establishing a DeltaShaper connection between the Skype endpoints installed in both VMs. We gathered data for two DeltaShaper configurations, found to provide traffic analysis resistance guarantees, and which respected the tuple (payload frame area, cell size, number of bits, framerate). These were comprised by the $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ and $\langle 160 \times 120, 4 \times 4, 6, 1 \rangle$ tuples. Each video was streamed in a 640x480 resolution.

2.3 Similarity-based Classification

For the purpose of unobservability assessment, multiple similarity functions have been used to feed similarity-based classifiers. This section details the rationale behind each of these functions and how they have been used for the construction of similarity-based classifiers and applied to different multimedia protocol tunneling systems. Then, we conduct a comparative analysis of the performance of each of these classifiers.

2.3.1 Currently Used Similarity Functions

Next, we introduce the three similarity-based classifiers which have been previously used for evaluating the unobservability of Facet, CovertCast, and DeltaShaper.

In similarity-based classification [40], labeling is performed by taking into account the pairwise-similarities between the test sample and a set of labeled training samples (or a representative model based on these). In the context of traffic analysis, similarity scores are often obtained from the comparison of the frequency distribution of packet lengths or inter-arrival times of traffic samples.

Pearson's Chi-squared Test (χ^2) [154] tells us whether the distributions of two categorical variables differ significantly from each other, by comparing the observed and expected frequencies of each category. The χ^2 test is used in a classifier adapted for distinguishing Facet traffic [105, 209]. The classifier starts by building two models for legitimate and Facet traffic, respectively, using labeled samples. These models are based upon a selection of the bi-gram distribution of packet lengths, where bi-grams expected to hurt classification performance are identified and discarded. Test samples are compared to each of the models using the χ^2 test. A simpler version of this classifier labels a sample according to the minimum distance obtained when compared against each model. A more sophisticated version of the classifier labels samples according to whether

the ratio between the distance to each model surpasses a threshold. An adversary can adjust this threshold for balancing the classifier's true positive and false positive rates.

Kullback-Leibler Divergence (KL) [98] is a measure of relative entropy between two target distributions which is obtained by computing the information lost when trying to approximate one distribution with the other. The KL divergence is used for building a classifier for CovertCast traffic. The classifier aims at distinguishing a set of YouTube videos carrying modulated data from a set of regular YouTube videos through the comparison of the quantized frequency distribution of packet lengths. For each sample in a given set, the classifier computes its KL divergence from every other member in the same set and every member in the other set. Then, the classifier computes a success metric, corresponding to the number of times the KL divergence between a member of one set is more similar to another member of the same set, divided by the total KL divergences that were computed.

Earth Movers's Distance (EMD) [171] measures the dissimilarity between two distributions, where the distance between single features can be defined in a distance matrix. Informally, this dissimilarity represents the necessary amount of work to turn one probability distribution into the other, where the cost of this transformation translates to the amount of observations moved times the distance defined in the distance matrix. The EMD (provided with a unitary distance matrix) is used for comparing the quantized frequency distribution of packet lengths of traffic samples, and is used as basis for building a classifier for DeltaShaper traffic. First, the classifier computes the pairwise EMD between each sample in the dataset and each legitimate sample, recording its average. The intuition is that legitimate samples will exhibit a smaller average EMD. A threshold adjusts the trade-off between the true positive and false positive rates of the classifier. For labeling a new sample, the classifier computes the pairwise distance of this sample to each legitimate sample and verifies whether its average EMD surpasses the threshold.

2.3.2 Main Findings

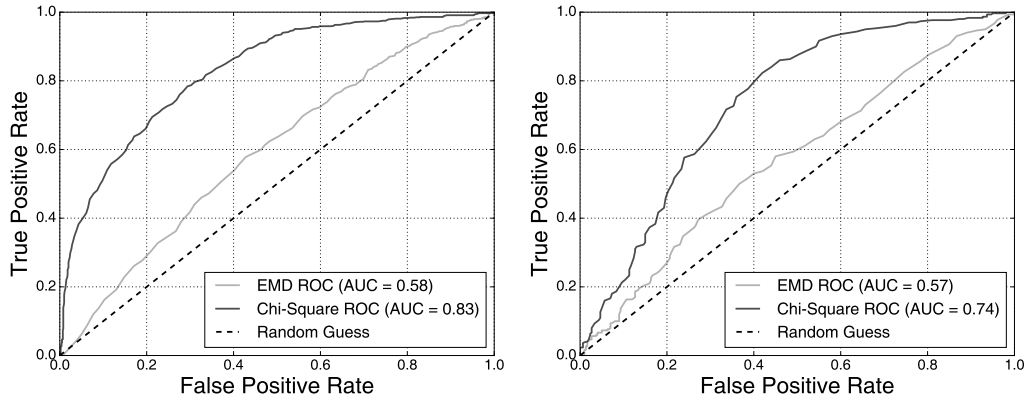
We now present the main findings of our analysis after assessing the unobservability of each system with all the similarity-based classifiers described above.

1. The claims on the unobservability guarantees of multimedia protocol tunneling systems are intimately tied to the classifier employed in their evaluation. This finding can be illustrated by the numbers in Table 2.1, which shows the accuracy, true positive and true negative rates obtained by the classifiers described in Section 2.3.1. For example, when detecting Facet s=50% traffic, we can see that the χ^2 performs relatively well, with an accuracy of 74.3%. Contrastingly, the KL and EMD classifiers attain an accuracy close to random guessing, providing an optimistic estimate on the unobservability of Facet s=50%. The values in Table 2.1 suggest a similar trend when detecting DeltaShaper and CovertCast traffic. The consistently higher accuracy values provided by χ^2 also suggest that this classifier outperforms all other similarity-based classifiers proposed for the assessment of unobservability.

2. χ^2 produces large false positive rates when classifying Facet and DeltaShaper traffic. Figure 2.1 depicts the ROC curve of the χ^2 and EMD classifiers when detecting Facet and DeltaShaper traffic. Figure 2.1a shows that for correctly identifying 90% of all Facet traffic (TPR=90%), with s=50%, an adversary would tag 45% of legitimate connections (45% FPR) as covert traffic, while employing the χ^2 classifier. For identifying 90% of all DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$ traffic, the adversary would face an FPR=51%. Thus, even the deployment

	χ^2 Classifier			KL Classifier			EMD Classifier		
Multimedia Protocol Tunneling System	ACC	TPR	TNR	ACC	TPR	TNR	ACC	TPR	TNR
Facet (s=50%)	0.743	0.797	0.689	0.575	0.675	0.476	0.575	0.578	0.572
Facet (s=25%)	0.713	0.795	0.630	0.558	0.615	0.500	0.535	0.827	0.242
Facet (s=12.5%)	0.772	0.793	0.750	0.551	0.596	0.506	0.530	0.793	0.267
DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$	0.690	0.716	0.663	0.546	0.628	0.464	0.567	0.500	0.633
DeltaShaper $\langle 160 \times 120, 4 \times 4, 6, 1\rangle$	0.540	0.437	0.650	0.515	0.531	0.500	0.528	0.223	0.833
CovertCast	0.990	1.000	0.980	0.923	0.999	0.846	0.830	0.965	0.695

Table 2.1: Accuracy, true positive, and true negative rates when detecting covert channels on different multimedia protocol tunneling systems. For the EMD classifier, the threshold value was chosen to be the one providing the highest accuracy, irrespective of the trade-off between the true positive and true negative rates of the classifier.



(a) Facet s=50% (b) DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$.

Figure 2.1: ROC curve for the χ^2 and EMD classifiers when identifying Facet and DeltaShaper.

of the best performing similarity-based classifier results in a large number of misclassifications for legitimate traffic. Misclassifications are further aggravated should an adversary resort to the EMD classifier. Figure 2.1 confirms that χ^2 performs only fairly in distinguishing covert channels (e.g., AUC=0.83 for Facet s=50%, AUC=0.74 for DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$). We do not show a ROC curve for KL as the classifier is not adjustable by an internal threshold.

3. CovertCast fails to provide unobservability. The results in Table 2.1 show that the χ^2 classifier can correctly identify all of CovertCast streams while incurring only in a 2% false positive rate. Additionally, the numbers show that the remaining classifiers can correctly identify >96.5% of CovertCast streams, albeit incurring in a larger false positive rate (e.g., EMD: TPR=0.965, FPR=0.305). We conjecture two explanations that may justify the differences beween our results and those published in the original CovertCast paper. Firstly, our results may stem from the use of a dataset which is one order of magnitude larger than the one used for CovertCast evaluation. Our dataset may more accurately represent the patterns generated by legitimate YouTube streams' traffic and reveal CovertCast activity. Secondly, implementation changes in YouTube may have impacted the unobservability properties provided by hardcoded data modulation parameters, which may in turn be no longer adequate to ensure unobservability.

2.4 Decision Tree-based Classification

In this section, we depart from the use of similarity-based classifiers for detecting the presence of covert traffic. As it is unpractical to explore all possible machine learning algorithms, we focus our experiments in a subset of algorithms based on decision trees. We have chosen these algorithms due to their ability of handling data in a non-linear fashion, their ability to perform feature selection, and the ease of interpretation of the resulting models. Our results show that this approach is highly effective at detecting covert traffic in the systems under study.

2.4.1 Selected Classifiers

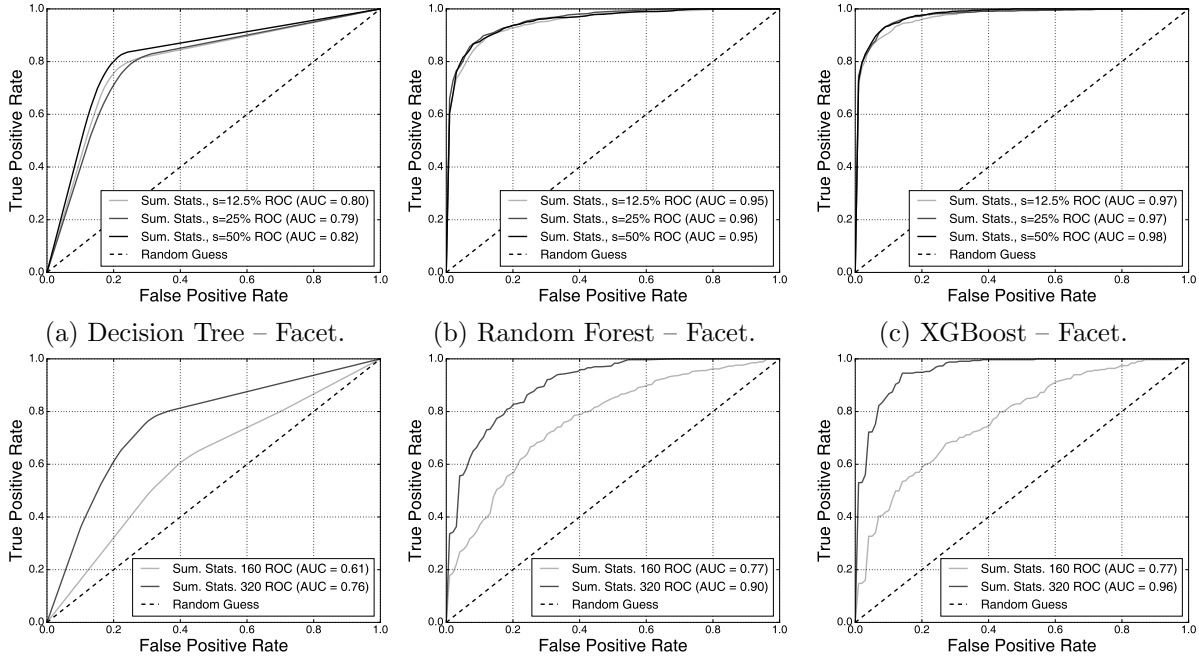
We present a description of the decision-tree based algorithms we have chosen for conducting our experiments:

Decision Trees [157] build a model in the form of a tree structure, where each tree node is either a decision or leaf node, representing a branch or a label, respectively. Decision nodes split the current branch by an attribute. A splitting attribute is commonly chosen according to its expected information gain, i.e. the expected reduction in entropy caused by choosing the attribute for a split. The importance of each particular attribute can be assessed by analyzing the tree structure, where nodes closer to the root have a higher importance than those down the tree. Despite its simple interpretation, decision trees can result in complex models unable to generalize well or can build unstable models due to the presence of large numbers of correlated features. A popular way to mitigate such disadvantages is to use decision tree ensembles.

Random Forests [27] are an ensemble learning method, where a label is predicted by performing a majority vote over the output of multiple decisions trees. To prevent overfitting, Random Forests introduce variance in the model through *bootstrap aggregation*, i.e. each tree is trained using a random sample (with replacement) of the training set. Additionally, Random Forests select random attributes of the feature set when building each tree, a technique named *feature bagging*. One method for assessing the importance of an attribute is to average its information gain across all trees in the ensemble.

eXtreme Gradient Boosting (XGBoost) [39] is another technique for building a model based on an ensemble of decision trees; it relies on a technique known as *gradient tree boosting*. XGBoost starts by building a shallow decision tree (i.e., a weak learner). In each step, XGBoost creates a new tree which optimizes the predictions performed by trees in earlier stages. XGBoost benefits from a regularized model formalization to control overfitting. The importance of individual attributes can be computed in a similar fashion to that of Random Forests. We find the use of XGBoost to be promising among a large pool of classification algorithms. In fact, XGBoost has played a central role on multiple winning solutions for recent data mining competitions, spawning multiple domains, such as the KDD Cup 2016 [48, 172]

The next sections detail our experiments for evaluating the unobservability of Facet and DeltaShaper with the decision tree-based classifiers enumerated above. In our experiments we have used two distinct sets of features: summary statistics and quantized packet lengths. We omit a discussion over CovertCast, as we have found that all of these techniques can identify its covert traffic with a negligible false positive rate.



(a) Decision Tree – Facet. (b) Random Forest – Facet. (c) XGBoost – Facet.

(d) Decision Tree – DeltaShaper. (e) Random Forest – DeltaShaper. (f) XGBoost – DeltaShaper.

Figure 2.2: ROC curves for decision tree-based classifiers when classifying Facet and DeltaShaper traffic using Feature Set 1 (summary statistics).

2.4.2 Feature Set 1: Summary Statistics

The collection of encrypted traffic provides an adversary with two main sources of data for extracting features necessary for the detection of covert channels: a timeseries of packet lengths, and a timeseries of packet inter-arrival times. Our first set of features comprises a collection of summary statistics computed over the network traces of legitimate and covert traffic. This is a prevalent approach at generating features for the problem of encrypted traffic fingerprinting [74, 143, 202]. Such set of features has not been previously applied in the detection of covert channels generated by multimedia protocol tunneling.

As for the choice of summary statistics, we compute multiple descriptive statistics for the ingress/ egress packet flows of a connection as a whole, as well as for ingress/ egress traffic individually. This feature set includes simple descriptive statistics over the packet length and inter-arrival time timeseries – such as maximum, minimum, mean, and percentiles – as well as higher-order statistics like the skew or kurtosis of these timeseries. We also consider burst behavior [2], where a burst is a sequence of consecutive packets transmitted along the same direction of a given connection. A total of 166 features are used for training our classifiers.

Next, we present our main findings after attempting to detect covert channels using the decision-tree based classifiers we have described, while feeding them with our collection of summary statistics. We report the performance of each classifier over 10-fold cross-validation.

1. The use of Random Forest/ XGBoost, used in tandem with summary statistics, largely undermines the unobservability claims of state-of-the-art multimedia protocol tunneling systems. Figure 2.2 shows the ROC curve for our decision tree-based classifiers when detecting Facet and DeltaShaper traffic resorting to summary statistic features (ST).

Random Forest – ST exhibits a minimum AUC=0.95 when classifying all configurations of Facet traffic, while XGBoost – ST exhibits a minimum AUC=0.97. When compared to XGBoost – ST, the χ^2 classifier attains a maximum AUC=0.85. For DeltaShaper traffic, XGBoost – ST attains an AUC which is 0.22 larger for both DeltaShaper configurations, when compared to that obtained by the χ^2 classifier.

2. It is possible to flag a vast majority of covert channels with a very small number of false positives. An adversary that aims at flagging at least 90% of all Facet s=50% connections incurs in a 14.1% FPR when resorting to Random Forest – ST, and a FPR as short as 7.1% when resorting to XGBoost – ST. To flag at least 70% of the same kind of traffic, XGBoost – ST incurs in a FPR of only 1%. In comparison, Figure 2.1a shows that for correctly identifying just 70% of Facet s=50% traffic when resorting to the χ^2 classifier, an adversary would face an alarming 21.5% FPR. The situation is similar for an adversary wishing to flag 90% of DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic. For flagging 90% of this kind of traffic, Random Forest – ST incurs in a 30.3% FPR and XGBoost – ST incurs in a 12.1% FPR. To flag 70% of the same kind of traffic, XGBoost – ST incurs in a FPR of 4%. Flagging just 70% of this kind of traffic with the χ^2 classifier would amount to a 32.2% FPR.

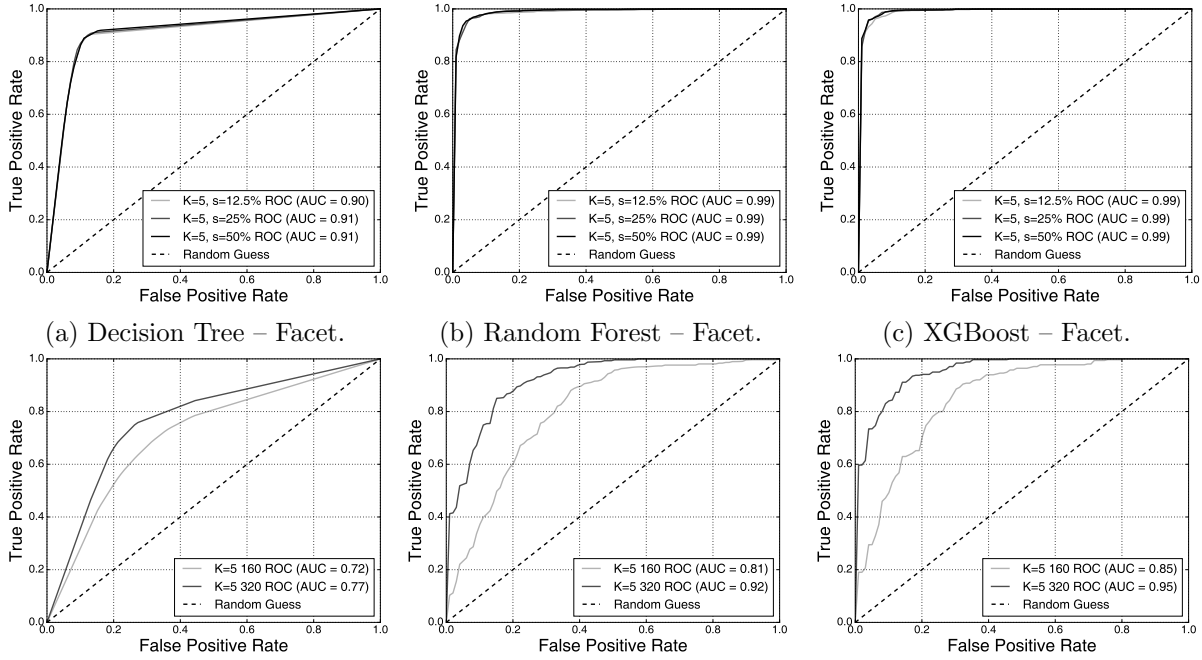
2.4.3 Feature Set 2: Quantized PLs

An alternative feature set is comprised of the quantized frequency distribution of packet lengths, where each K size bin acts as an individual feature. While this feature set is akin to that previously used in KL and EMD similarity-based classifiers, we process these features in a fundamentally different way. In particular, the similarity-based classifiers output a distance score based on the overall difference of the packet lengths frequency distribution, while failing to adjust this score according to the importance of relevant regions of the feature space. Informally, they risk to dilute the greater discriminating power of a given feature among that of possibly irrelevant features [126]. We aim at exploiting the different relevance of particular ranges of the feature space by feeding this feature set to decision tree-based classifiers.

For Facet, we take as features the quantized frequency distribution of packet lengths for the flow carrying covert data. We use K=5 as we have experimentally verified that the classification performance of our decision tree-based algorithms benefit from a fine-grained quantization. As for DeltaShaper, and due to the system's bidirectionality, we use the quantized frequency distribution of packet sizes flowing in both directions. Here, we also apply a quantization with K=5. Note that the evaluation performed with the similarity-based classifiers described in Section 2.3 also considers the same selection on the direction of traffic flows to analyze.

Next, we describe our findings after attempting to identify covert traffic with such feature sets. Figure 2.3 shows the ROC curve for our decision tree-based classifiers when detecting Facet and DeltaShaper traffic resorting to quantized packet lengths as features (PL).

1. Quantized packet lengths outperform the use of summary statistics. In general, the AUC obtained by our decision-tree based classifiers is comparable or superior to the AUC obtained by the same classifiers when making use of summary statistics. Both Random Forest - PL and XGBoost - PL obtain an AUC=0.99 when identifying Facet traffic. This represents a maximum improvement of 0.04 over Random Forest – ST and 0.02 over XGBoost – ST. While Decision Tree - PL attains a maximum AUC=0.91, it is still short of the maximum AUC attained by Random Forest – ST. This trend is similar in the classification of DeltaShaper traffic, where



(a) Decision Tree – Facet. (b) Random Forest – Facet. (c) XGBoost – Facet.

(d) Decision Tree – DeltaShaper. (e) Random Forest – DeltaShaper. (f) XGBoost – DeltaShaper.

Figure 2.3: ROC curves for decision tree-based classifiers when classifying Facet and DeltaShaper traffic using Feature Set 2 (quantized frequency distribution of packet lengths).

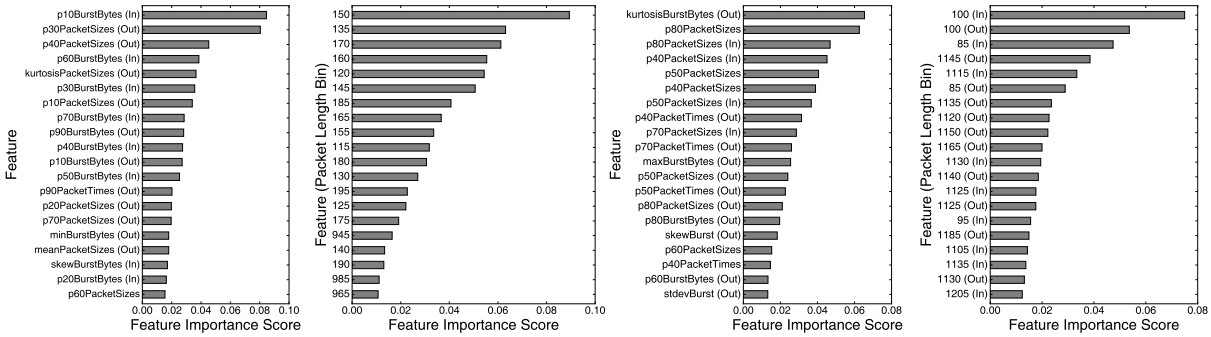
the AUC obtained by Decision Tree - PL is also inferior to that of tree ensembles. The detection of $\langle 160 \times 120, 4 \times 4, 6, 1 \rangle$ DeltaShaper traffic benefits the most from packet length features, where XGBoost - PL attains an AUC=0.85, 0.08 larger than that obtained by XGBoost – ST. Interestingly, the detection of $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ DeltaShaper traffic is better performed by XGBoost – ST, albeit by a slight improvement of 0.01 over the AUC of XGBoost - PL.

2.4.4 Feature Importance

The above set of experiments allowed us to implicitly identify which features are more important to distinguish between two classes of traffic. Figure 2.4a shows the top 20 most important summary statistics for detecting Facet traffic s=50%, as reported by the XGBoost algorithm. Figure 2.4b summarizes the 20 most important quantized ranges of packet lengths. The features annotated with "Out" correspond to those generated by the packet flow directed towards the client (carrying the covert payload), while the features annotated with "In" correspond to the packet flow directed towards the Facet server.

Figure 2.4c depicts the top 20 most important summary statistics for detecting DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic, as reported by XGBoost. Similarly, Figure 2.4d depicts the most important quantized ranges of packet lengths for detecting the same kind of traffic. Each feature is annotated with "Out" or "In", depending on the particular Skype peer originating covert traffic. We note that both peers generate covert traffic simultaneously due to DeltaShaper's bidirectionality. Below, we discuss the main findings of our analysis.

1. Facet is more vulnerable to analysis based on packet lengths and burst behavior. Figure 2.4a shows that Facet detection is driven by features related to the packet lengths and



(a) ST - Facet. (b) PL - Facet. (c) ST - DeltaShaper. (d) PL - DeltaShaper.

Figure 2.4: Top 20 most important features when classifying Facet s=50% and DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic, as calculated by XGBoost. We report the mean score of each feature across all 10 cross-validation folds.

the burst behavior of the connection, whereas packet timing does not contribute as much. An interesting observation is that the majority of packet bursts features considered important for classification are those included in the flow directed towards the Facet server, which carries no covert data. This fact suggests that Skype flows exhibit some degree of co-dependency and that both flows provide useful information for distinguishing between legitimate and covert transmissions. Features included in the top 10, and that directly concern the length of packets, index summary statistics from the flow carrying covert data. This suggests that the flow carrying covert data is the prime target for inspection when analyzing packet lengths. Additionally, packet lengths comprehended between the 10th and 40th percentiles, amounting to packets with a mean length comprehended between 138 and 213 bytes, have a superior discriminating power among other packet sizes. XGBoost ranks 123 of the 166 features with a non-zero importance score.

2. DeltaShaper is more vulnerable to analysis based packet lengths. Figure 2.4c shows that 10 of the most important features for detecting DeltaShaper regard descriptive statistics of packet lengths. In particular, 7 out of the top 10 most important features for identifying DeltaShaper traffic are related to the length of transmitted packets. Contrary to Facet, these features include a mixture of traffic originating in different peers, which is expected according to the bidirectionality of the covert channel. We find the most influential packet lengths to be within the range of the 40th and 80th percentiles, amounting to packets with a mean length comprehended between 1026-1180 bytes. XGBoost ranks 132 of the 166 features with an importance score larger than zero.

3. Facet covert channels can be spotted by looking for packets with a length comprehended between 115-195 bytes. Figure 2.4b not only shows that the most important bin corresponds to that by the packets which length is close to 150, but also that the top 10 features are dominated by packets which lengths are in the range of 115 to 195 bytes. This result concurs with our previous observation, where the most important percentiles of packet lengths focused packets with a mean length between 137 and 200 bytes. This observation is also true when detecting Facet s={12.5%,25%} traffic. This finding suggests that the major factor leading to the distinguishing of Facet traffic concerns the packets carrying audio, which are typically located in the range between 100 and 200 bytes [130]. Additionally, we can observe that some of the least important features included in the top 20 for identifying Facet s = 50% flows include packets with a length between 945-985 bytes. This result hints that larger areas dedicated to

video payload translate into packet-level modifications in a higher range of the feature space. Additionally, XGBoost ranks only 175 out of 300 features with a non-zero importance score, suggesting that only approximately half of the quantized packet length bins contribute for the discrimination of Facet traffic.

4. DeltaShaper covert channels can be spotted by looking for packets with a length between 85-100 and 1105-1205 bytes. Figure 2.4d shows that the two most important features for identifying DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic correspond to the packets which size is close to 100 bytes (flowing in both directions). The top 20 features are dominated by packet length bins in the range from 85-100 and 1105-1205 bytes, suggesting that DeltaShaper data modulation markedly affects two distinct regions of the feature space. The region including larger packets roughly overlaps the mean length of the packets included in the most important percentiles of our analysis of summary statistics. Considering that DeltaShaper's covert data embedding procedure targets the video layer of Skype calls, this finding suggests that such modulation affects larger packets of the connection. When classifying DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic, XGBoost ranks 253 out of 600 features with a non-zero importance score.

The most important features for detecting DeltaShaper $\langle 160 \times 120, 4 \times 4, 6, 1 \rangle$ traffic largely overlap the two feature set regions already reported. However, we verify that the region including larger packet lengths was significantly expanded, including bins representing packets with a size within the range of 885-1200 bytes.

2.4.5 Alternative Dataset Evaluation

We have constructed and handled our dataset by following the same methodology adopted by previous works under study. However, this methodology may raise a few concerns. In particular, the covert streams (positive class) have been produced using the available legitimate videos (negative class), which may introduce some form of correlation among classes. Furthermore, this methodology generates a 1:1 ratio of positive to negative classes, which may be unrealistic if covert streams are a minority among the traffic found in the wild. Thus, one may wonder how accurate is our classifier if: i) the positive class is no longer correlated with the negative class during testing; ii) the positive-to-negative sample ratio is low during testing. To validate the effectiveness of our approach, we performed two additional experiments.

First, we performed an experiment which removed the correlations between the positive and negative classes. We split our legitimate traffic dataset in half, using only one half as legitimate samples. Then, for creating our covert video dataset, we selected those covert videos which embed modulated data in the legitimate videos out of our reduced legitimate traffic dataset. We then used XGBoost to build a model through 10-fold cross-validation. To prevent the fitting of results to a particular choice of the initial legitimate samples, we repeated the process 10 times while randomly choosing such samples.

Second, we performed an experiment where we keep the positive-to-negative sample ratio low during testing. We split our data in training / testing sets in a 70 / 30 proportion, and where we kept the training set ratio as 1:1, and keep the positive to negative ratio of the testing set to 1:100. To prevent the fitting of results to a particular split of the data, we randomly choose each set 10 times.

The results of our additional experiments suggest that possible correlations among training and testing data, as well as sample ratios, do not limit the accuracy of our approach. For our

System	Feature Set	Memory (kB)	Storage (kB)
Facet	Summary Statistics (ST)	1.3	1.8
	Packet Lengths (PL)	2.4	1.0
DeltaShaper	Summary Statistics (ST)	1.3	1.9
	Packet Lengths (PL)	4.8	2.0

Table 2.2: Memory and storage requirements for a single Facet record using different feature sets. We report storage requirements for holding data in raw ASCII text.

first experiment, XGBoost obtained an AUC=0.94 for DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic (only 0.01 less than the results reported in Section 2.4.3), and an AUC=0.99 for traffic pertaining to Facet s=50% configuration. As for the second experiment, XGBoost was able to correctly identify 90% of Facet s=50% traffic with an FPR of only 2%, while it was able to identify 90% of DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1 \rangle$ traffic with an FPR of 18% (only 4% larger).

2.4.6 Practical Considerations

This section details several practical considerations which may be useful to an adversary considering the use of decision tree classifiers for the detection of covert channels. The following results reflect processing time in a VM configuration akin to that described in Section 2.2.4.

Feature extraction. The extraction of quantized packet length bins from a 60 second Facet network trace amounts to an average of 0.33s per sample. Generating summary statistics describing the same type of traffic flow amounts to an average of 0.44s per sample. This result indicates that an adversary can quickly generate feature vectors for conducting subsequent classification.

Memory and storage requirements. Table 2.2 depicts the memory and storage requirements for holding a single Facet or DeltaShaper sample. In our Python implementation, a NumPy [199] array storing the quantized packet lengths describing a Facet sample (300 attributes) occupies 2.4kB of memory per sample. In comparison, an array containing the bi-grams required by the χ^2 classifier occupy a total of 45kB per sample. The numbers in Table 2.2 suggest that an adversary can efficiently store and process large datasets. As an example, storing 1 million Facet quantized packet lengths feature vectors in a raw ASCII text file would only occupy approximately 1GB of disk space. Storing summary statistics in raw ASCII text would occupy nearly twofold the space due to the characters required to represent floating-point precision.

Model building and classification speed. Table 2.3 depicts the average training time of our classifiers, as well as the average time to output a prediction. Building a Decision Tree - PL for identifying Facet traffic takes an average of 0.27s. For an ensemble composed of 100 trees, Random Forest - PL and XGBoost – PL models are built in 1.45s and 0.41s, respectively. Moreover, the average classification time for an individual sample is 180μs for XGBoost – PL. XGBoost is not only more accurate but also trains faster and exhibits a faster classification speed than Random Forest. This relation is also present when classifying DeltaShaper traffic. These results stress the fact that an adversary would benefit from using XGBoost to detect multimedia protocol tunneling covert channels.

Generalization ability of the classifiers. A classifier with good generalization ability is able to perform correct predictions for previously unseen data. Albeit the AUC obtained by our

System	Classifier	Model Building (s)	Prediction (μs)
Facet	Decision Tree	0.27	40
	Random Forest	1.45	15000
	XGBoost	0.41	180
DeltaShaper	Decision Tree	0.13	90
	Random Forest	0.86	16000
	XGBoost	0.38	350

Table 2.3: Model building time and time for individual predictions for Facet s=50% and DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$ traffic, using quantized packet lengths (PL). Model building time is the average of 10 folds.

System	1s	5s	10s	30s	60s
Facet	0.81	0.92	0.96	0.99	0.99
DeltaShaper	0.75	0.88	0.93	0.95	0.95

Table 2.4: AUC of XGBoost – PL when classifying Facet s=50% and DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$ traffic for varying traffic collection time windows.

decision tree-based classifiers suggests that these can generalize well, we further assess their classification performance when training data is severely limited. We split our data in two 10 / 90 training and testing sets, and report the mean AUC obtained by the classifier after repeating this process 10 times while randomly choosing the samples making part of each set. In this setting, when classifying Facet s=50%, XGBoost - PL attains an AUC=0.98, only 0.01 short of that obtained after 10x cross-validation. For DeltaShaper $\langle 160 \times 120, 4 \times 4, 6, 1\rangle$ traffic, XGBoost - PL attains an AUC 0.1 smaller than their 10x cross-validation counterpart. These results suggest that an adversary can build accurate decision tree-based classifiers for detecting covert traffic while resorting to a small sample of data.

Impact of network traces collection time. Table 2.4 depicts the AUC obtained by XGBoost – PL when detecting different types of covert traffic for varying time-spans of traffic flows collection. Results show that capturing traffic by 30s is enough for attaining the same classification performance achieved in our initial experiments, which admitted 60s traffic captures. The numbers in Table 2.4 also show that classification performance decreases monotonically for traffic collections fewer than 30s, suggesting that the inspection of at least 30s of video traffic provides the adversary with sufficient data for identifying covert traffic flows with low false positives.

2.5 Beyond Supervised Anomaly Detection

While decision tree-based classifiers show promising results for the detection of multimedia protocol tunneling covert channels, they require the adversary to obtain a labeled dataset, including both legitimate and covert traffic. This usually requires the adversary to have a unlimited access to a particular multimedia protocol tunneling tool with which it may generate covert traffic samples. However, even if an adversary, for instance a censor, would have an expedite access to these tools [61], it is interesting to understand if detection is possible without this knowledge. Note that covert channels may also be used by organized criminals that can succeed in delaying the dissemination of such tools. Secondly, albeit the adversary is assumed to

possess a given tool, it is expected to spend a non-negligible time in synthesizing covert data samples for building a model. Overcoming such challenges opens a timeframe where the covert traffic generated by a given system would remain undetected.

This section explores alternative approaches at covert traffic detection in the absence of a fully labeled dataset.

2.5.1 Selected Anomaly Detection Methods

This section starts by describing several anomaly detection techniques which could be of interest for an adversary aiming at detecting covert traffic when it is deprived of labeled anomalies. First, we describe OCSVMs and autoencoders, two well-known approaches for anomaly detection, which are based on representational models of legitimate data and thus disregard the need of labeled anomaly data [207]. Then, we explore Isolation Forest, a competitive approach at unsupervised anomaly detection which does not require labeled data [21, 31, 89].

One-class SVMs [173] define a decision boundary between normal samples and anomalies by fitting a function around normal samples during training. OCSVMs attempt to find the maximal margin hyperplane which separates the normal data from the origin, which is treated as the single member of a second class. If data cannot be easily separated by a linear function, OCSVMs project the original feature space into a new feature space through the use of kernel functions, introducing non-linearity in the model. New data samples falling outside the decision boundary are considered anomalies.

Autoencoders [116] are a type of artificial neural networks which can approximate the identity function through a compressed representation of its inputs, forcing the algorithm to learn underlying structures in data. The ability to reconstruct inputs allows us to have a generative model of the training data. An autoencoder can be repurposed for anomaly detection by comparing the reconstruction error of training inputs with normal and anomalous data, where the latter is assumed to be larger.

Isolation Forest [109] performs outlier detection by isolating anomalous samples. To isolate a sample, the algorithm starts by selecting a random feature and selects a split between its minimum and maximum values. This process continues recursively until the considered sample is isolated. Recursive partitioning is represented by a tree, where the number of partitions required to isolate a sample corresponds to the length of the path traversed from the root node to a leaf. The Isolation Forest is built by combining isolation trees split on different attributes. Anomalies are expected to exhibit a smaller average path length than that of normal samples.

Hyperparameters. The classification performance of the above algorithms depends upon the choice of hyperparameters, i.e., parameters whose value must be set prior to the execution of the algorithm. The optimality of such parameters is intrinsically dependent on the dataset and tipically requires cross-validation with labeled anomalous data [231]. However, we are interested in assessing the average classification performance that an adversary would be able to achieve using such algorithms – albeit the adversary would be unable to find the optimal hyperparameter configuration for an algorithm, sub-optimal parameterizations may still provide the adversary with accurate traffic classifiers. To this end, we conduct a search over a space of parameters for the above algorithms and collect the maximum and average AUC obtained.

For OCSVM, we perform a grid search on the space of ν and γ. We also build a shallow autoencoder containing one hidden layer between the input and its compressed representation,

Multimedia Protocol Tunneling System	OCSVM		Autoencoder		Isolation Forest	
	Max AUC	Avg AUC	Max AUC	Avg AUC	Max AUC	Avg AUC
Facet (s=50%)	0.631	0.576	0.702	0.638	0.561	0.551
Facet (s=25%)	0.629	0.580	0.700	0.650	0.528	0.519
Facet (s=12.5%)	0.639	0.584	0.706	0.647	0.536	0.520
DeltaShaper $\langle 320 \times 240, 8 \times 8, 6, 1\rangle$	0.567	0.531	0.662	0.574	0.580	0.557
DeltaShaper $\langle 160 \times 120, 4 \times 4, 6, 1\rangle$	0.548	0.518	0.576	0.544	0.553	0.532

Table 2.5: Maximum and average AUC of OCSVM, Autoencoder and Isolation Forest when classifying Facet and DeltaShaper traffic. Search (min, max, step): OCSVM (ν(0.1, 1, +0.1), γ(0.01, 1, +0.01)); Autoencoder (hidden_layers(4,512,*2), compressed_representation(4,512,*2), learning_rate[0.001,0.01], epochs[1000]); Isolation Forest (n_trees(50,200,*2), n_samples(64,512,*2))

and between the compressed representation and the output layer. We conduct a grid search over the number of units populating each of these layers. As for Isolation Forest, we conduct a search over the number of trees composing the ensemble, as well as the number of samples for training each individual tree.

Experimental settings. For OCSVM and autoencoder, we use 90% of all labeled legitimate samples to learn the models. The remaining 10% legitimate samples are combined with 10% of a given covert traffic configuration's samples for creating a balanced testing set. For evaluating the model's performance, we compare each label output by the model with the ground truth. To prevent the fitting of results to a particular split of the data, we repeat this process 10 times while randomly choosing the samples making part of the training / testing sets. For Isolation Forest, we create balanced training and testing sets in a 90 / 10 proportion. The model's performance is evaluated following the same above procedure.

Our results reflect the use of the feature set based on the frequency distribution of packet lengths, with $K = 5$, as it was the one found to provide the highest AUC.

2.5.2 Main Findings

Table 2.5 depicts the maximum and average AUC obtained when identifying Facet and DeltaShaper traffic when using OCSVM, our autoencoder, and Isolation Forest. Next, we present our main findings.

1. OCSVMs possess a limited capability for correctly identifying covert traffic. This finding is supported by the fact that OCSVM attains an average AUC between 0.576 and 0.584 when detecting Facet traffic, and between 0.518 and 0.531 when detecting DeltaShaper traffic. Moreover, OCSVM achieves a maximum AUC=0.639 when classifying Facet s=12.5% traffic. This suggests that OCSVM achieves a poor classification performance, even after a search for optimal hyperparameters. Thus, from an adversary's point of view, a semi-supervised model based on OCSVMs shows little promise for performing the triage of covert traffic.

2. Autoencoders show promising results for the identification of covert traffic. The numbers in Table 2.5 show that our autoencoder achieves, in average, a higher or comparable AUC than the maximum AUC obtained by OCSVM when classifying Facet or DeltaShaper traffic. The choice of parameters for our autoencoder benefits its maximum AUC. For instance, a better parameterization of the autoencoder translates into a maximum AUC=0.662 when

classifying DeltaShaper traffic, approximately 0.1 higher than the average reported value for the same configuration. While an adversary making use of a classifier which exhibits an AUC=0.662 would sustain a large amount of false positives when attempting to detect covert traffic, we note that the obtained results have a wide margin of improvement. In particular, we use a rather shallow autoencoder structure for investigating the classification performance of this algorithm. For instance, it is possible that autoencoders with more sophisticated structures [127] may drive further improvements in classification accuracy.

3. An adversary has no advantage in using Isolation Forest for detecting covert traffic. The results in Table 2.5 show that the prediction output of Isolation Forest is close to random guessing when attempting to identify covert traffic. For Facet traffic, Isolation Forest obtains an average AUC between 0.519 and 0.551 across all steganography factors. When classifying DeltaShaper traffic, the average AUC sits on 0.532 and 0.557 for different encoding configurations. A closer observation of the confusion matrix reveals that Isolation Forest labels few traffic samples as anomalies. Informally, this observation suggests that anomalies are able to conceal their presence in the dataset in such a way that the number of partitions required to isolate them is similar to the number of partitions needed to isolate legitimate samples.

2.6 Discussion

We now discuss several relevant findings from our study.

Multimedia protocol tunneling. The outcomes of the experimental study conducted in Section 2.4 unveil that the unobservability claims of existing multimedia protocol tunneling systems were flawed. However, it is worth noticing that the vulnerability of such systems to supervised ML techniques, particularly decision tree-based algorithms, does not imply that multimedia protocol tunneling, as an approach, is fundamentally inviable. Our findings suggest that detecting covert channels built with conservative data modulation schemes (e.g., DeltaShaper $\langle 160 \times 120, 4 \times 4, 6, 1 \rangle$) while sustaining low FPR still represents a challenge for adversaries. Additionally, we provide details about the network behavior of currently deployed multimedia protocol tunneling tools which may be used for the construction of more robust implementations.

Legitimate traffic dataset. Adversaries face the non-trivial challenge of building a dataset which faithfully represents legitimate traffic. A naïve solution for building such a dataset would be for an adversary to take advantage of its privileged position in the network and collect all data originated by a given multimedia protocol. However, the very existence of multimedia protocol tunneling tools makes it hard for an adversary to know, before-hand, which data samples correspond either to legitimate or covert traffic. It is possible that covert data samples pollute the legitimate traffic model and bias the decisions of a classifier trained in such data [127]. A different alternative is the approach followed in the literature (and in our work), where datasets are synthesized by transmitting the media expected to be sent in such channels. However, such an approximation may fail to capture the underlying distribution of data in the wild.

2.7 Related Work

Freewave [82] was the first system designed to embed covert data in multimedia protocols through the modulation of audio signals sent through VoIP streams. However, a simple statistical

analysis of traffic patterns conducted by Geddes et al. [66] showed that FreeWave could be trivially detected by an adversary. Recent multimedia protocol tunneling systems such as Facet [105], CovertCast [121], and DeltaShaper [10] introduced new techniques for modulating data while striving to preserve the unobservability of the generated covert channels.

As noted earlier in the text, previous unobservability assessments performed on state-of-the-art multimedia protocol tunneling systems which rely on traffic classification make use of similarity-based classifiers. To the best of our knowledge, there is a limited body of work employing other machine learning techniques for the detection of covert channels in the Internet. Wang et al. [200] have resorted to decision tree-based classifiers to identify traffic flowing through Tor bridges. Their results have shown that this approach was promising for the identification of traffic obfuscated through domain fronting [60]. In our work, we perform the first systematic study of the unobservability of state-of-the-art multimedia protocol tunneling systems and find that such techniques are also effective for the detection of these covert channels.

Related to the problem of covert channel detection is the problem of creating fingerprints for encrypted traffic. Particularly, the fingerprinting of websites accessed through Tor [47] is an important research topic [2, 74, 151, 165, 202]. Multiple works dwell on creating fingerprints for encrypted traffic using different combinations of features and classifiers, for instance, Schuster et al. [175] have designed an attack which enables a passive observer to fingerprint YouTube video streams. However, fingerprinting is fundamentally different from covert channel detection: we do not aim to unequivocally fingerprint a given media according to its traffic pattern, but to distinguish two broader classes of media which may or may not carry covert data. It is unclear how fingerprinting techniques can be adapted to our purpose.

In this paper we have focused on covert channels based on multimedia protocol tunneling [10, 82, 105, 121], a popular approach at protocol tunneling. Other tunneling approaches have been attempted, including SWEET [233], CloudTransport [29], Castle [71], and *meek* [60]. It is worth mentioning that alternative approaches to build covert channels have been attempted in the past, such as protocol obfuscation [211]. However, obfuscation based on randomizing traffic fails in the presence of protocol whitelisting and is vulnerable to entropy analysis [200]. With protocol imitation, covert traffic is manipulated to mimic the behavior of protocols allowed across a censor's border [51, 52, 129]. Alas, the faithful imitation of all behaviors of a protocol behavior is a complex undertaking which lays protocol imitation systems prone to multiple network attacks [66, 80].

Finally, we would like to stress that although censorship circumvention is one of the main (and most noble) uses of covert channels, this type of channels can serve multiple purposes. Our work concentrates on covert channel detection and not on censorship circumvention *per se*. In fact, there are techniques to evade censorship, such as refraction networking [23, 53, 81, 90, 212, 213], which incorporates censorship resistance mechanisms in the network, rather than at end-hosts, that do not depend exclusively on the use of covert channels.

2.8 Conclusions

In this paper, we performed an extensive analysis over the unobservability evaluation of multimedia protocol tunneling systems. We proposed a novel method for assessing the unobservability of these systems, based on decision trees, which largely defies previous unobservability claims. Our work further explored the application of semi-supervised and unsupervised anomaly

detection techniques in the same context. Our results indicate that an adversary is required to possess labeled data for performing an effective detection of covert channels.

Acknowledgments

This work was partially supported by national funds through Instituto Superior Técnico, Universidade de Lisboa, and Fundação para a Ciência e a Tecnologia (FCT) via projects PTDC/EEI-SCR/1741/2014, SFRH/BSAB/135236/2017, and UID/CEC/50021/2013.

3 FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications

Publication Data

Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando M.V. Ramos, André Madeira. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. Proceedings of the 27th Network and Distributed System Security Symposium, San Diego, CA, USA, February 2021

Abstract

An emerging trend in network security consists of the adoption of programmable switches for performing various security tasks in large-scale, high-speed networks. However, since existing solutions are tailored to specific tasks, they cannot accommodate a growing variety of ML-based security applications, i.e., security-focused tasks that perform targeted flow classification based on packet size or inter-packet frequency distributions with the help of supervised machine learning algorithms. We present FlowLens, a system that leverages programmable switches to efficiently support multi-purpose ML-based security applications. FlowLens collects features of packet distributions at line speed and classifies flows directly on the switches, enabling network operators to re-purpose this measurement primitive at run-time to serve a different flow classification task. To cope with the resource constraints of programmable switches, FlowLens computes for each flow a memory-efficient representation of relevant features named "flow marker". Despite its small size, it contains enough information to perform accurate flow classification. Since flow markers are highly customizable and application-dependent, FlowLens can automatically parameterize the flow marker generation guided by a multi-objective optimization process that can balance their size and accuracy. We evaluated our system in three usage scenarios: covert channel detection, website fingerprinting, and botnet chatter detection. We find that very small markers enable FlowLens to achieve a 150 fold increase in monitoring capacity for covert channel detection with an accuracy drop of only 3% when compared to collecting full packet distributions.

The reproduction of this publication, following below, was slightly adapted to adhere to formatting requirements. The original version of this publication can be found at: `https://www.ndss-symposium.org/wp-content/uploads/2021-067-paper.pdf`.

3.1 Introduction

Recently, several systems have been proposed for tackling security concerns in modern high-speed networks [229, 87, 123, 215]. By leveraging the capabilities offered by programmable switches, these systems can process packets at line speed directly on the switch hardware, bringing relevant benefits for network security, such as decreased reaction times to attacks, avoidance of network bottlenecks, and decreased costs associated to equivalent centralized server-based infrastructures. So far, the proposed systems target very specific security-driven tasks. These tasks include the ability to mitigate DDoS attacks [229], enforce context-aware security policies [87], obfuscate network topologies [123], filter spoofed traffic [102], or detect data exfiltration through timing covert channels [215].

However, besides the specific tasks tackled by the previous work, there is currently a lack of support for a new range of security applications that resort to machine learning (ML) to classify flows in real time [232, 70]. This brand of applications has become more relevant as a result of a global trend towards encrypting all Internet traffic [43, 144], which has rendered deep-packet inspection (DPI) increasingly ineffective. As an alternative to DPI, the use of ML-based techniques has proved useful to classify flows with high accuracy for a wide range of scenarios, such as multimedia covert channel detection [12], website fingerprinting [107], botnet traffic identification [134], malware tracking [3], IoT device behavioral analysis [148, 191], or detection of DRM-protected streaming [69, 176, 162].

Most of these ML-based applications rely on supervised machine learning algorithms [12, 107, 176, 134] that need to collect flow features such as packet length and/or inter-packet time frequency distributions. However, both the set of features and ML algorithms used are highly application-dependent. As such, a general service to enable the implementation of ML-based security applications must be versatile enough to accommodate application-specific requirements without impairing its ability to produce accurate classification results. In addition, it must efficiently use the limited switch resources to maximize the number of flows that can be probed, scale to large networks comprising numerous switches, introduce minimal switch downtime caused by upgrades of switch programs, and require low maintenance effort.

We present FlowLens, a system that enables efficient flow classification for multi-purpose ML-based security applications. At the heart of our system lies a set of software components that run on the network switches' data plane and control plane. These components are responsible for collecting compact, but meaningful, features of the flows going past the data plane, and for running the ML-based algorithms responsible for classifying the flows on the control plane in real time. By performing both these tasks on the switches in a fully decentralized fashion, FlowLens does not depend on a centralized service that could introduce bottlenecks for operations in the critical path. To deliver the best performance, these software components must be fine-tuned for each specific ML-based security application. FlowLens includes the mechanisms to generate (and upload to the switches) application-dependent configurations that strike a good balance between classification accuracy and switch resource utilization efficiency. Because these configurations can be automatically generated, the maintenance effort of our system is greatly reduced.

A key challenge in fully offloading ML-based flow analysis onto the switches is tied to the hardware and programming restrictions of modern programmable switches. Ideally, we would like to collect the full packet length and inter-packet arrival time frequency distributions for every flow traveling through the switches. This approach would allow us to collect full per-flow information (on the data plane) which different applications could then process in order to

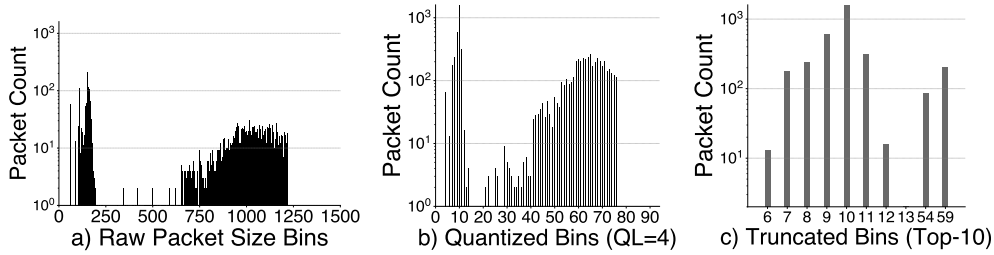


Figure 3.1: Histograms of packet size distribution for a single flow. A flow consists of a stream of packets identified by the same 5-tuple of TCP/IP header fields, at increasing degrees of compactness: a) the raw packet size distribution; b) the quantized representation, where packet sizes are aggregated into bins of size 2^4 bytes; c) the flow marker generated through truncation of the quantized representation which comprises the most relevant 10 bins for the detection of covert channels mounted through multimedia protocol tunneling.

extract relevant features and running specific ML classifiers (on the control plane). However, given that the amount of stateful switch memory is very limited, an information-lossless scheme for collecting flow data would considerably reduce the coverage of our system, i.e., the number of simultaneous flows that could be probed. In alternative, one could employ a lossfull scheme where the amount of dedicated memory allocated per flow is reduced thereby increasing flow coverage. Such a scheme, however, must be such that (1) the collected information does not deteriorate the accuracy of flow classification, (2) it can be implemented with a small set of basic hardware instructions and within few compute cycles as imposed by the switch, and (3) it precludes the need to frequently reprogram the switch as it would cause switch downtime in the order of seconds.

To address this challenge, we make two core technical contributions. First, we devised a new compact representation of packet length and inter-time packet arrival distributions which is small yet provides enough information to perform accurate application-specific traffic classification. We name such representations *flow markers*. We then developed a primitive named Flow Marker Accumulator (FMA) which generates flow markers while depending on simple and efficient operations that can be implemented on modern programmable switches. The FMA consists of a parameterizable data structure deployed on the data plane pipeline such that, for each incoming packet, it performs two simple operations, namely *quantization* and *truncation*, which adjust the granularity of the flow's frequency distribution intervals in bins (quantization), and select the bins considered to be the most relevant features for flow classification (truncation). As shown in Figure 3.1, the set of resulting bins for each flow constitutes the respective flow marker, which will then be processed by the classification algorithm.

Second, we developed an automatic profiler to find adequate quantization and truncation parameters of the FMA for a given application. Because there is a large space of configurations that present different trade-offs between switch memory savings and flow classification accuracy, manually setting up these parameters for each application would both be cumbersome and render sub-optimal results. Our profiler relies on well-known Bayesian optimization techniques [62] for finding suitable configurations by iteratively testing only a small subset of the possible FMA configurations. It can be tuned to find parameterizations according to different criteria, including (a) the maximization of a user-defined trade-off between space-efficiency and accuracy, (b) the smallest marker able to achieve a classification accuracy above a given threshold, or (c) the marker that maximizes the accuracy given some space constraint.

We have implemented FlowLens on a Barefoot/Intel Tofino programmable switch and evaluated our system in three use case applications: covert channel detection, website fingerprinting, and botnet detection. When comparing the classification scores achieved by FlowLens against those computed over raw packet length distributions, FlowLens offers similar accuracy scores while using significantly less memory, e.g., covert channels can be detected with at most 3% loss in accuracy using only a 20-byte memory footprint per flow. When compared with related methods for capturing compressed packet frequency distributions [45, 137], FlowLens consistently outperforms them in terms of the classification accuracy under similar memory restrictions. FlowLens also achieves considerable bandwidth savings when compared to network telemetry approaches [183] that rely on a server infrastructure responsible for flow analysis.

3.2 Motivation and Design Goals

This section motivates our work by characterizing a set of emerging ML-based security applications and discussing the technical constraints of modern programmable switches. It then provides an overview of FlowLens's design goals.

3.2.1 ML-based Network Security Applications

In recent years, generalized interest has grown in detecting atypical network flows using ML classification algorithms [144]. To deliver accurate flow classification results, these algorithms depend on a range of features that require the collection of the packet length/inter-packet timing frequency distributions. Below, we present three examples of applications in the realm of network security that rely on the analysis of such distributions for performing traffic classification. These examples are chosen to showcase the versatility of FlowLens in accommodating different classification algorithms. We will further use them to validate the classifiers' accuracy when deployed on FlowLens.

Covert channel detection: Capturing packet distributions makes it possible to detect covert channels, thereby providing a valuable asset for cyberforensic investigations. To achieve stealthy data transmissions, advanced covert channel tools tend to obfuscate covert flows such that their high-level features (e.g. packet lengths) resemble those of regular flows [210, 10, 105, 121]. However, recent work [12] has shown that these tools can be defeated or severely weakened due to subtle differences in packet distributions which can be detected by ML techniques.

Website fingerprinting: Privacy-enhancing technologies like OpenSSL or OpenVPN allow users to hide the destination address behind a proxy and the content of website visits from external observers through the use of encryption. However, it may still be possible to identify which sites they access by collecting the flows' packet length distributions [107, 77] and feeding them to ML classification algorithms for website fingerprinting purposes. This technique may help authorities respond against individuals engaged in illegal activities.

Botnet chatter detection: Botnets [93] can jeopardize the security of multiple organizations, emerging as a highly profitable activity for malicious actors [156]. Unfortunately, due to their decentralized P2P architectures and stealthy communication patterns, botnets have become incredibly resistant to takedown attempts. Nevertheless, state-of-the-art approaches to analyzing botnet traffic are able to identify the presence of bots through the combined analysis of packet

lengths and inter-packet timing distributions of network hosts [122, 134]. Being able to employ these techniques can help network administrators prevent and mitigate botnet threats to an organization's network.

3.2.2 Design Goals

In this work, our goal is to use programmable switches to collect packet frequency distributions and provide a multi-purpose flow classification platform for implementing a variety of ML-based security applications. By using our solution, a network operator will be able to scan local traffic in near real-time and look for specific flows that match a set of application-specific traffic patterns, such as those presented in Section 3.2.1. In summary, we are driven by the following design goals:

Scalability: We aim at monitoring flows in very large and fast networks (at the Tbps scale), comprised of many switches, while reducing the costs of the network telemetry infrastructure. To this end, we aim to avoid relying either on edge-based solutions, which capture the traffic through middleboxes [153], or solutions that collect packet features on the switches but offload them for further processing and classification on dedicated servers [182, 183]. Reducing the bandwidth consumed with the offloading of telemetry data is also a crucial point as the amount of collected data grows with the increasingly high link speeds and becomes substantial for large scale networks [224].

Accuracy: We aim at collecting a compact representation of packet distributions while retaining enough information about flows to enable high accuracy on classification tasks. Achieving such a representation requires the design of a flow compression scheme that is simple enough to be efficiently implemented by the primitives available in current P4-programmable switches, but which is able to retain meaningful features for classification purposes. Additionally, computing compact representations of packet distributions should not consume the majority of resources in the switch, enabling the system to co-exist with other typical applications, e.g. forwarding, or to be used in tandem with complementary network telemetry solutions.

Availability: Re-purposing our system to different traffic analysis tasks should not involve the deployment of a new P4 program. This is because deploying a new program involves a scheduled downtime during which the switch will be unable to perform its basic functions.

3.2.3 Constraints of Modern Programmable Switches

To efficiently collect and process packet distributions, we explore the programming capabilities of modern switches, such as Barefoot Tofino [6] and Broadcom Tomahawk II [28]. These switches include two types of processors which operate in two different planes of the network architecture. On the data plane, forwarding ASICs are able to quickly forward and perform simple computations on packets at line-rate, thus enabling the analysis of billions of packets at the Tbps scale. On the control plane, CPUs can be used for general-computing tasks such as controlling the packet forwarding pipeline, or for exchanging data with the ASIC through DMA.

Switching ASICs can be programmed in a hardware-independent language, such as P4 [25]. Figure 3.2 illustrates the architecture of our targeted switching ASIC: the Protocol Independent Switch Architecture (PISA) [35]. Packets arrive at the switch ingress interfaces and, after parsing, are processed by two logical pipelines of *match+action units* (MAUs) arranged in stages. Packet headers along with packet metadata may then *match* (M) a given table, triggering further

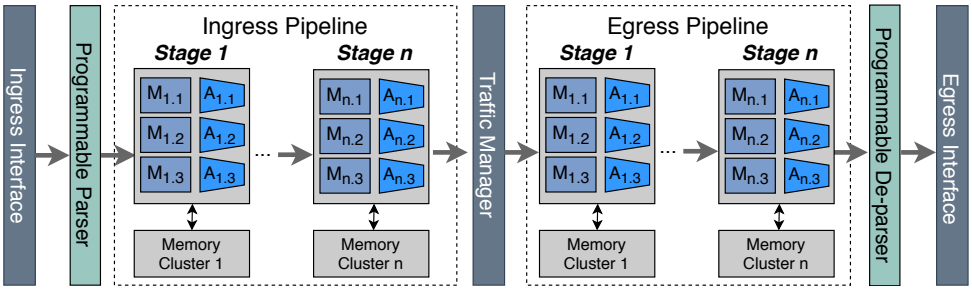


Figure 3.2: Protocol Independent Switch Architecture (PISA).

processing by the *action* (A) unit associated with the matching table's entry. These actions may modify packet header fields and change persistent state (e.g., increment a counter in stateful memory). Tables and other objects defined in a P4 program are instantiated inside MAUs and populated by the control plane at run-time.

Memory constraints: Several constraints in the memory architecture of switching ASICs may restrict the layout of the data structures that can be used by P4 programs. These ASICs are equipped with two high-speed types of memory: (i) TCAM, which is a content addressable memory suited for fast table lookups, such as for longest-prefix matching in routing tables [227], and (ii) SRAM which enables P4 programs to persist state across packets (e.g., using register arrays), and to hold exact-match tables. Unfortunately, switching ASICs contain a small amount of stateful memory (in the order of 100MB SRAM [125]), and only a fraction of the total available SRAM can be used to allocate register arrays. Moreover, accessing all available registers can be a complex task since the registers in one stage cannot be accessed at different stages [42]; this is because the SRAM is uniformly distributed amongst the different stages of the processing pipeline (see Figure 3.2).

Processing constraints: The P4 programs installed on the switch must use very simple instructions to process packets. To guarantee line-rate processing, packets must spend a fixed amount of time in each pipeline stage (a few ns [179, 177]) which restricts the number and type of operations allowed within each stage. Multiplications, divisions or floating-point operations, and variable-length loops are not supported. Moreover, each table's action can only perform a restricted set operations, like additions, bit shifts, and memory accesses that can quickly be performed while the packet is passing through an MAU without stalling the whole pipeline [180].

3.3 System Overview

This section describes FlowLens, a system for efficient flow classification that achieves the aforementioned goals. Figure 3.3 shows the architecture of FlowLens, illustrating how it can be used to monitor traffic on a single switch of a high-speed network. In general, it can be deployed across multiple other switches in the network at the system operator's discretion. FlowLens consists of the following components: a *P4 program* and two software components (*collector* and *classifier*) running on the switch, a standalone *profiler* server, and a software *client* that provides an interface to the system operator.

The components running on the switch are responsible for analyzing traffic and classifying flows as per ML-based security application. The P4 program runs on the data plane, and

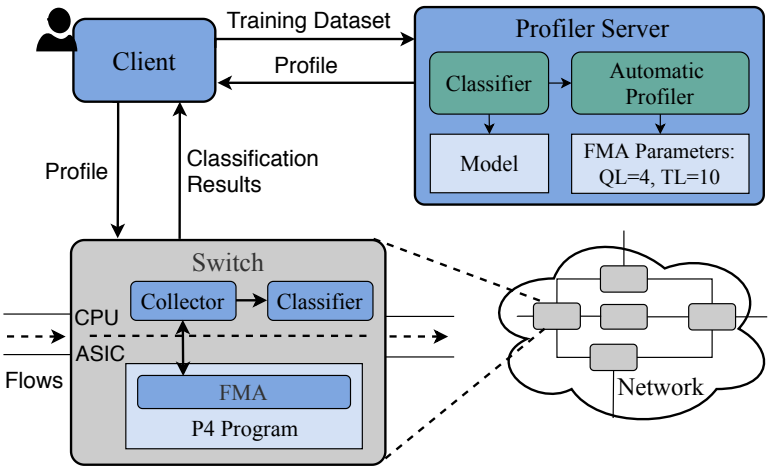


Figure 3.3: FlowLens architecture and components.

implements a tailor-made data structure named *Flow Marker Accumulator* (FMA). The FMA is used to collect concise encodings of the packet length or inter-packet timing distributions of flows named *flow markers*. Essentially, the FMA implements the necessary knobs for fine-tuning the memory savings and the classification accuracy of the system. It can accommodate the restrictions of the switch and generate flow markers at different compression levels by adjusting a set of configuration parameters that control (i) the size of the memory footprint allocated per-flow marker, and (ii) the loss of information in the flow markers due to compression.

Running on the local CPU of the switch, two additional FlowLens software components implement several control plane functions. The *collector* is responsible for loading the P4 program, configuring the FMA data structures in the forwarding pipeline, initiating the flow collection process, and collecting the resulting flow markers. The *classifier* runs the ML algorithms responsible for classifying flows based on the collected markers. FlowLens can generally employ any ML algorithm that can reason over flow markers, and whose memory and compute requirements can be accommodated on the switch control plane. After the classification step, results can be downloaded by the client and displayed to the system operator.

Since the classifier and the FMA configuration parameters depend on the application domain, FlowLens uses a standalone *profiler* to pre-configure the classifier models and FMA parameters (i.e., the application *profile*) onto the switch. The system workflow involves two phases: profiling and flow classification.

1. Profiling: FlowLens needs to be pre-configured by the system operator for specific applications. This operation involves the profiler server, which can automatically create profiles by using an application-specific classifier and a training set containing labeled flow samples provided by the system operator. To this end, the profiler runs an optimization process that explores the classification performance of different FMA quantization and truncation values, generating a configuration according to a given user-defined criterion (see Section 3.5).

2. Flow classification: As soon as the P4 program has been loaded into the switch (which happens only once when bootstrapping FlowLens), the collector takes the profile computed in phase 1 for a specific application, e.g., website fingerprinting, and configures the FMA accordingly. Afterward, the switch can start to process packets and compute flow markers. The collector then fetches the resulting flow markers from the data plane and the classifier processes them based on

the loaded model. The classifier results can be retrieved by the system operator, who can then take targeted actions about particular flows such as dropping flagged flows or scheduling further logging operations. The system operator can later reconfigure the system for other ML-based application profiles without the need to re-deploy the P4 program.

In the following sections, we present the relevant design details of FlowLens, namely the FMA data structure (Section 3.4) and the automatic profiling scheme aimed at choosing markers with good accuracy/memory saving trade-offs (Section 3.5).

3.4 Flow Marker Accumulator

The Flow Marker Accumulator (FMA) is the data structure responsible for computing flow markers on the switch data plane. Next, we present its internal operations by describing the FMA design for capturing markers of packet length distributions, and presenting the main changes when computing inter-packet arrival timing distributions. Then, we describe how the FMA is used by the control plane, and discuss alternative FMA setups.

3.4.1 Collecting Packet Length Distributions

To generate flow markers, the FMA provides two basic operators that can be implemented efficiently and be used to obtain a space-efficient encoding of a packet length distribution: *quantization* and *truncation*. Quantization consists of counting packet lengths in coarse bins that represent ranges of contiguous packet lengths. Truncation further trims the number of bins that need to be reserved for a certain classification task. These operators allow us to selectively collect the bin values which, in many cases, correspond to the most relevant features employed by the ML engine to yield accurate classifications [220, 12].

To perform these operations, the FMA is composed of several data structures shown in Figure 3.4. They consist of two match+action tables and one register array: the *flow table*, the *truncation table*, and the *register grid*, respectively. The register grid is a matrix of memory registers. Each line is used to store a flow marker. The index of each line (flow offset) is used to address the flow marker. Internally, a flow marker consists of a number of cells in a register (the grid's columns). These cells play the role of bins for storing samples of the flow's packet length frequency distribution. The flow table maps the monitored flows against the respective flow markers in the register grid. The truncation table identifies the bin that must be incremented for every incoming packet.

Next, we describe in detail the procedure for updating the flow marker for a given incoming packet. Consider the example shown in Figure 3.4, where an input packet arrives in the switch from source IP 162.2.13.42, source port 41065, with length 1024 bytes. The FMA performs the following four operations:

1. Lookup: First, the FMA's flow table matches the incoming packet with the corresponding flow ID, which is a 5-tuple of header fields $\langle IP_{src}, Port_{src}, IP_{dst}, Port_{dst}, Proto \rangle$ that is used as lookup key to return its associated flow offset. To be efficiently performed, we leverage the match+action units of the switch to accommodate specific rules for flow table indexing. Each rule in the flow table assigns a unique flow offset to each flow ID. For instance, in the running

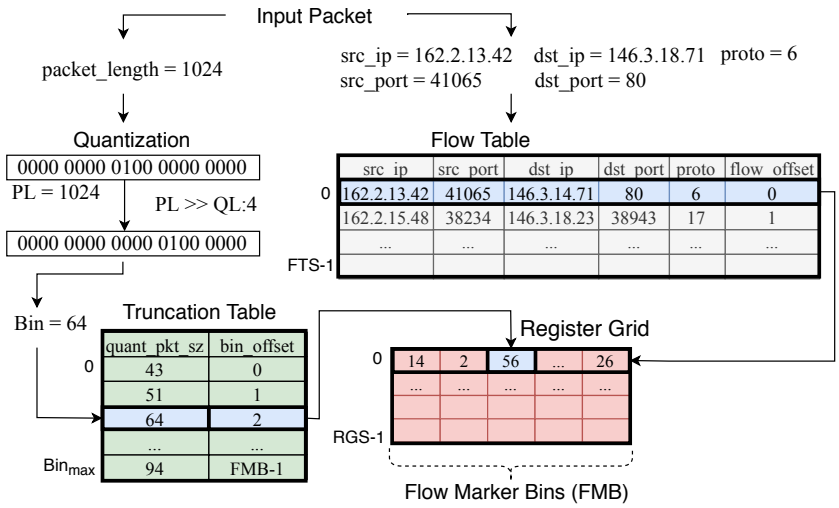


Figure 3.4: FMA internals: Flow Table, Truncation Table, and Register Grid.

example, the input packet is matched against the rule $\langle 162.2.13.42, 41065, 146.3.18.71, 80, 6\rangle \rightarrow 0$. (Section 3.4.3 describes how the flow table is populated.)

2. Quantization: The flow offset determined above locates the packet's flow marker in the register grid. Next, the FMA must increment the correct bin in the flow marker which is a function of the packet length. The first step to determine the right bin involves quantization, which aggregates, and so counts, a range of contiguous packet lengths into the same bin. To avoid complex instructions unsupported by the switch hardware (e.g., multiplications), the bin indexed by a certain packet length PL is computed by $bin(QL, PL) = length(PL) >> QL$, where QL denotes the *quantization level* and $0 \leq QL < log_2(PL_{max})$. For efficient lookup of a packet length's bin, FMA uses power-of-two bin sizes; this allows for computing the packet bin by right-shifting the packet length value by QL number of bits. In the shown example, applying $QL = 4$ to the packet length (1024 bytes) yields quantization bin #64.

3. Truncation: Based on the obtained quantization bin, truncation leverages an auxiliary data structure – the truncation table – which contains match+action rules exclusively for the bins that should be accounted for in the flow marker. Each rule is keyed by the quantized bin length, i.e., $bin(QL, PL)$, and indexes the flow marker's bin (bin offset) where the packet length frequency must be recorded. If no such rule exists for a given quantized bin, the packet is not counted. In this example, the current packet is considered because a rule exists for the packet's quantized bin (#64). In contrast, packets whose quantized bin values fall, e.g., within the bin range 52-63, will not be accounted for. This strategy allows for selectively filtering the most meaningful bins for flow classification.

4. Increment: Lastly, by combining the flow offset and the bin offset, the register grid can be indexed and the correct bin incremented. In our example, this entails incrementing bin 2 of the flow marker pointed to by the flow offset 0. These steps are repeated for every incoming packet.

3.4.2 Collecting Packet Timing Distributions

Gathering inter-packet timing distributions requires only minor modifications to the FMA design. This task does not affect the main FMA data structures and operators, but it requires additional resources in the switch processing pipeline.

To compute the arrival time difference between two consecutive packets of the same flow, the FMA stores the timestamp of the last packet seen per each flow. That information is available to the P4 program through device intrinsic metadata. With exception to the first packet of a flow, FMA computes the difference between the current packet timestamp and the last timestamp observed for that particular flow. That value can then be processed by the same quantization and truncation operators described for packet lengths, which produce the corresponding bin to be updated in the register grid.

3.4.3 Usage of the FMA by the Control Plane

To monitor flows on a switch, the FMA must be coordinated by the control plane's collector software. The collector predefines the FMA's quantization and truncation parameters, and determines: i) which amongst all flows traversing the switch at a given time will be monitored by the FMA, and ii) for how long the packets of the monitored flows will be considered in the respective flow markers. Next, we present the default FMA operational settings and then describe alternative customization policies that can be enabled by the FlowLens operator.

Default FMA measurement operations: By default, the control plane sets up the FMA to i) compute flow markers for all the flows on a first-come, first-served (FCFS) basis until they exhaust the register grid capacity, and ii) measure the flows' respective packets for a predefined time interval that we refer to as *collection window*. The control plane starts by clearing all registers of the FMA's register grid, and setting a timer for the duration of the collection window. Then, for every incoming packet for which a rule does not exist in the FMA's flow table (i.e., the packet belongs to a new flow), the control plane automatically installs one of two possible rules for that flow. If there is free space in the register grid, then it will install a new rule that points to a free flow marker, allowing the flow to be monitored by the FMA. Otherwise, if the register grid is full, the control plane will install a single wildcard *ignore rule* (featuring a reserved offset value) instructing the FMA to ignore that flow and all subsequent flows until the end of the collection window. When the timer expires, the control plane reads in batch all the computed flow markers. To prevent concurrent updates while reading, the control plane deletes the flow rules prior to reading the registers. Once the registers have been read, a new collection window round can then ensue. A side effect of this design is that FlowLens may skip the packets of a new flow until the flow's respective rule is installed in the switch. However, while this is a limitation of our system, such loss does not impair FlowLens's ability to trace most typical flows as they tend to last longer than a few milliseconds.

Discretionary flow monitoring: Prior to the enforcement of the FCFS flow marker allocation strategy, it is possible to filter which traffic should be monitored and therefore limit the amount of flows to inspect. For instance, ML-based security applications are frequently focused on traffic that can be identified by common target ports (e.g., HTTP). In other cases, the FlowLens operator may be interested in monitoring flows based on IP or network address ranges. Such a discretionary flow filtering stage can be implemented on the control plane by installing ignore rules for all uninteresting flows based on allow / deny policies provided by the FlowLens operator.

Ignore rules can be defined to target a single flow or to perform wildcard matching based on IP and port ranges, specific protocol fields, etc.

Fine-tuning of the collection window: Flow markers should not linger for an arbitrarily large amount of time inside FMA's data structures, as this would prevent the flow marker's memory from being used for other flows. To increase the number of flows that can be monitored at any given time, the collection window can be configured, based on three setups: i) the definition of a fixed window duration (explained above); ii) the use of a specific flag set in FMA data structures that enables the control plane to check for flow termination through a polling procedure; iii) a hybrid of both approaches. While option i) is arguably the simplest alternative, it may lead to memory waste since short-lived flows can occupy the FMA's data structures longer than necessary. In contrast, a polling approach allows FlowLens to pinpoint flows that will receive no new packets (e.g., after detecting FIN packets in the data plane pipeline which signals the termination of a TCP connection), but it may indefinitely keep flows which termination is not explicit (e.g., UDP-based multimedia flows). Thus, a hybrid collection approach allows us to proactively read and reset the FMA data structures in use by terminated flows while preventing long-lived flows to be monitored indefinitely. In other words, this approach fully refreshes the register grid every time the collection window is over and partially updates it whenever a particular row is in condition to be evicted.

Flow marker eviction: A high rate of new flows may saturate the capacity of the FMA data structures and prevent storing flow markers for all flows crossing the switch. In this case, as explained above, the FMA's default strategy is to not track new flows as long as existing flows are still being tracked. As an alternative behavior, it is possible to evict flow markers from the FMA according to an LRU policy. In this case, the control plane keeps track of the oldest flow markers stored by the FMA, and replaces them as new incoming flows cross the switch. The most suitable policy will greatly depend on the expected workload and topology of the network.

3.4.4 Distributed and Orchestrated FMA Operation

So far, we explained several design decisions of FlowLens when considering its operation to be contained within a single switch. In this section, we describe how FlowLens can benefit from a deployment in multiple vantage points.

Scaling the number of measured flows: Although the number of flows whose state can be kept by a single switch is limited, it is possible to take advantage of multiple vantage points in the network for monitoring a larger amount of flows. This is akin to the operation of other measurement frameworks [111, 84] and may be accomplished, for instance, by splitting packets coming from different IP address spaces between existing switches in the organization's network.

Increasing collection coverage: In the case that our system operates with a maximum collection window (see Section 3.4.3), reading and resetting FMA's data structures requires a non-negligible amount of time (a few seconds) [99]. This may prevent FlowLens from collecting flow information while these operations take place. To ensure visibility over the network traffic crossing an organization, FlowLens can be deployed in a cascade fashion across an additional switch to intertwine the collection windows of the different switches.

Increasing application coverage: The design of FlowLens is tailored for enabling a single profile to be loaded into a given FMA at any given time. However, a coordinated operation of FlowLens across several switches can provide support to multiple ML-based security applications.

For instance, when deployed across multiple switches, one FMA instance may be dedicated to the detection of covert channels, while other is dedicated to the identification of botnet behavior.

3.5 Automatic Profiling

The flow markers generated by FMA depend on the parameters, i.e., the quantization level and the truncation table, dictated by an application-specific profile which determines how efficiently the switch SRAM will be used and how accurate the flow classification will be. In general, finding the parameters that offer an optimal trade-off would require an exhaustive search of the parameter space. Unfortunately, this is a cumbersome task that requires non-trivial computational resources and time, e.g., automatically exploring the full space of configurations for the botnet detection task (Section 3.7.6) took one day.

To search on the parameter space for a configuration that offers a good trade-off between flow marker size and classification accuracy, the profiler implements optimization techniques that, albeit may fail to yield the optimal result, usually find near-optimal solutions quickly. Next, we describe the optimization criteria and algorithm employed in FlowLens. Note that FlowLens is not tightly coupled to a specific implementation of the profiler and nothing prevents the use of alternate optimization techniques [181, 18]. Investigating further optimization approaches is outside the scope of this paper.

3.5.1 Optimization Criteria

We expect that a FlowLens's operator will want to find a suitable FMA parameterization for any given ML-based security application. Because there is a space/accuracy trade-off in the FMA configuration, we are faced with a multi-criteria optimization problem that does not have a single optimal solution but, instead, has a number of Pareto optimal solutions [142]. The current version of the FlowLens's profiler can approximate three different pre-set points in the Pareto frontier, that can be selected by the system operator:

1. Smaller marker for target accuracy: In this mode, the system operator specifies a target accuracy value to be attained, and the profiler automatically chooses the quantization and truncation parameters that yield the smallest marker that is able to offer the target accuracy. Note that the profiler will not return a configuration if the accuracy set by the user cannot be achieved for the particular dataset under analysis.

2. Best accuracy given a size constraint: Here, the operator specifies the maximum size for the flow marker and the system automatically picks the quantization and truncation parameters that maximize the classification accuracy, among the configurations explored, without exceeding the target marker size. This constraint also allows us to reduce the search space, since the marker size generated by a set of quantization and truncation parameters is known beforehand.

3. Size vs. accuracy trade-off: Lastly, the profiler can work in a fully automated fashion. In this case, the profiler attempts to maximize an accuracy vs marker size trade-off that is expressed by the following reward function: $reward = \alpha \cdot accuracy + (1-\alpha)\frac{1}{marker\ size}$. A smaller α attributes less importance to the accuracy in favor of compactness, and vice-versa. Our prototype uses $\alpha = 0.5$, but the system operator can define the value of α as well as a different reward policy of its choosing altogether.

3.5.2 Optimization Algorithm

The search space is the product of the different quantization and truncation configurations; on its own, the number of configurations that result from truncation is combinatorial with the number of available bins. To guide the profiler's search, we use an optimization algorithm that consists of two phases. In the first phase, called *search space reduction*, we use domain knowledge to narrow the search, by excluding configurations that are unlikely to offer acceptable results. In the second phase, we resort to *Bayesian optimization* to find a suitable flow marker. We detail these two steps in the next paragraphs.

1. Search space reduction: To reduce the search space, we discretize the domain of quantization parameters, e.g., aggregate bins in powers of two. Then, we leverage a pre-training step for narrowing the truncation space: we first generate a coarser representation of the packet distributions of each sample in the training data according to a given quantization parameter, then we use a classifier to build a model based on these representations. We leverage the fact that most classifiers can output information regarding the top-N most relevant bins for accurate classification. Thus, for each quantization, we constrain the exploration to points that include an increasing number of features from the top-50 (i.e., the configuration that includes the top-10 bins, the top-20 bins, etc). When the classifier is unable to output the top-N features, we fall back to a simpler strategy to narrow the search space: we sample the input space, exclude bins that have not been observed in the sampled points, and feed the remaining ones to the optimizer.

2. Bayesian optimization: To reduce the manual labor required to explore a large space of configurations, we rely on Bayesian optimization, a well-known method for optimizing black-box functions and for finding near-optimal solutions with few function evaluations [62]. We optimize the combination of quantization and truncation parameters using the Python Hyperopt [19] software package. In each iteration, the profiler selects a parameterization, trains a classifier using flow markers accordingly generated, and records the classifier accuracy alongside the size of produced flow markers. The next parameterization to sample is selected by the optimizer which we run for a fixed number of iterations. The whole process took us a few hours to complete.

3.6 Implementation

We built a prototype of the FlowLens system. Excepting the FMA, which was written in P4, we implemented all other components in Python. The classification engine of the profiler server uses Python's `scikit-learn` [155] and the Weka [72] libraries. We implemented FlowLens's FMA [16] for a Barefoot Tofino ASIC [6] using about 500 lines of $P4_{16}$ code, which was compiled with the P4 Studio Software Development Environment (SDE) [7]. While the FMA's design presented in Section 3.4.1 is generically compatible with PISA architecture, its implementation required careful reasoning due to the specific intricacies of currently available switching hardware.

To implement the FMA code for a Tofino switch, we need to fit the FMA's data structures and operations into the specific pipeline and compute capabilities of the switch. To implement flow marker updates, it would be desirable to compute the flow offset and the bin offset of the target flow marker (see Figure 3.4) in a single pipeline stage to be able to use all the memory in upcoming stages to store flow markers. However, this cannot be achieved on our target hardware due to three major data dependencies: i) matching (i.e., indexing) the truncation table depends on the quantized packet length, but quantization and truncation are too complex to be realized

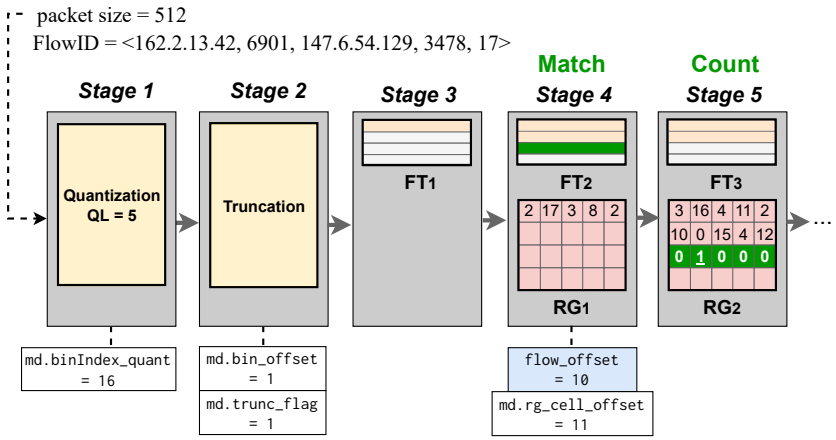


Figure 3.5: FMA implementation on Tofino switch. Each partition of the flow table (FT_m) records `flow ids` (marked in orange) and their corresponding flow markers are recorded in the register grid partition of the upcoming stage (RG_m). This packet is matched in FT_2 and the respective flow marker updated in RG_2. The white boxes in the bottom indicate the metadata values computed in each stage; the value in the blue box is loaded by the control plane.

together in a single stage; ii) indexing a flow marker's bin requires the result of truncation, but the truncation table and the flow table cannot be matched in the same physical stage; iii) matching the flow table and updating the respective flow marker are also too complex to perform in a single stage. Moreover, it is not possible to access all the switch memory from a single stage.

Laying out the FMA in hardware: To accommodate for the above requirements, we split the functioning of FMA across different stages, as depicted in Figure 3.5. To resolve dependencies i) and ii), we reserve the first and second stages of the pipeline to perform quantization and truncation. Then, we partition the flow table and register grid along the remaining stages to use up all the per-stage stateful registers across the processing pipeline. To overcome the inability to calculate the bin offset and increment the corresponding register cell in the same stage – dependency iii) – the flow table partitions and register grid partitions are placed in contiguous stages. Each flow table partition is responsible for managing flow markers in its corresponding register grid partition.

Figure 3.5 depicts in detail the operation of FMA when a packet for a new flow arrives. Notation FT_m and RG_m denote partition m of the flow table and register grid, respectively. Assume that the collector has installed a rule for `flow id` ⟨162.2.13.42, 6901, 147.6.54.129, 3478, 17⟩ in FT_2, and that the first incoming packet for this flow has a size of 512B. In stage 1, action `quantization_act` is triggered, quantizing the packet length using QL=5 and setting the resulting quantized packet length (`md.binIndex_quant`) to 16. The object `md` stores the metadata carried over across the pipeline stages.

In the second stage, the truncation table matches against the quantized packet length (refer to Figure 3.4) and triggers the `truncation_act` action, which returns the bin offset `md.bin_offset` within the flow marker and sets a truncation flag `md.trunc_flag` in order to inform the downstream stages that this packet's corresponding flow marker should be incremented. In case no match exists in the truncation table, the truncation flag is not set, and the packet is not accounted for.

Next, since the flow matching rule is not installed in FT_1, the packet is not matched until it reaches stage 4, where FT_2 is located. Upon matching the `flow id` and verifying that `md.trunc_`-

Listing 1 P4 code fragment that implements the actions performed per-packet by the FMA. The complexity of the truncation operator and of the computation of flow marker offsets is offloaded to the control plane and the resulting values are loaded through MAUs.

```
1  // triggered by the quantization table
2  // bin_width_shift depends on the Quantization Level (QL)
3  action quantization_act(bit<32> bin_width_shift){
4          md.binIndex_quant =
5          (bit<32>) (md.pkt_length >> bin_width_shift);
6  }
7  // triggered by the truncation table
8  action truncation_act(bit<32> new_index, bool flag) {
9          md.bin_offset = new_index;
10         md.trunc_flag = flag;
11 }
12 //triggered by the flow table (FT_2)
13 // compute the offset of the bin in the RG partition
14 action set_flow_data_act2(bit<32> flow_offset) {
15         md.rg_cell_offset = flow_offset + md.bin_offset;
16
17 //triggered after matching the flow table (FT_2)
18 action reg_grid_act2() {
19         bit<16> value;
20         reg_grid2.read(value, md.rg_cell_offset);
21         reg_grid2.write(md.rg_cell_offset, value+1);
22 }
```

`flag` is set, the `set_flow_data_act2` action is triggered. This action computes `md.rg_cell_offset` by adding `md.bin_offset` and `flow_offset` loaded by the control plane into FT_2. The resulting value is used to index a cell in RG_2 which is then incremented. To index the correct partition of the register grid, we use control flow logic to test which flow table partition was matched, triggering the respective `reg_grid_act2` action that updates the flow marker on the corresponding register grid partition in the next pipeline stage.

Optimizing per-packet computations: To reduce the complexity of the P4 program, we leverage the capabilities offered by the control plane to offload the computation of complex operations from the data plane. This results in a program sampled in Listing 1 which implements four simple actions that can be performed within single stages of the pipeline. Specifically, we offload two operations into the control plane:

a) Computing a flow offset within the register grid: The index of the register grid where a flow marker is located can be easily calculated in the control plane. Since the FMA parameterization is known prior to the loading of the P4 program on the switch, the control plane can compute the number of bins used by a flow marker in a given configuration. Thus, when a new flow is matched, the collector installs a rule where the flow offset is given by the number of flow rules installed in a particular flow table partition times the number of bins composing a marker. Upon matching, the flow offset is passed as an argument to action `set_flow_data_act2` (line 14).

b) Computing a bin offset within a flow marker: To index a bin within a flow marker two values must be added: the flow offset, and the bin offset. While the former can be computed as described in the previous paragraph, the latter is computed by the truncation operator. The quantized packet length passed as an argument to the action responsible for performing truncation (`truncation_act`, line 8) is computed by action `quantization_act` (line 3) using a simple bit shift. Then, the translation between a quantized packet length and the corresponding bin offset can also be computed offline once a specific FMA parameterization is known, and later loaded by the control plane into the truncation table (refer to Section 3.4.1). Pre-computing

these values in the control plane saves stateful memory and pipeline's stages for either monitoring more flows or executing other forwarding behaviors.

3.7 Evaluation

Here, we present our experimental evaluation of FlowLens aimed at analyzing the accuracy of ML-based flow classification tasks and the efficiency of the switch resources usage.

3.7.1 Metrics and Methodology

Our experiments aim at identifying a particular class of flows denoted as the *target* class. For instance, when using FlowLens for covert channel detection, the target class can be covert traffic. We assess the quality of FlowLens using the following set of metrics: *accuracy*, i.e., the percentage of flows that were correctly classified in their class, *false positive rate* (FPR), i.e., flows that do not belong to the target class but were erroneously classified as part of the target, and *false negative rate* (FNR), i.e., flows of the target class that were flagged as not belonging to the class. We also resort to related metrics such as *precision* – ratio of the number of relevant flows retrieved to the total number of relevant and irrelevant flows retrieved – and *recall* – ratio of the number of relevant flows retrieved to the total number of relevant flows.

We train our system to be able to identify specific target class flows within the context of three usage scenarios:

Covert channel detection: We train our system to identify Skype flows carrying covert channels encoded by two censorship resistance tools: Facet [105] and DeltaShaper [10]. We train two independent FlowLens applications, for Facet and for DeltaShaper traffic, using a balanced dataset including covert / legitimate samples of recorded flows. The traffic is classified using the XGBoost [12] classifier, based on the packet length distribution of the sampled flows.

Website fingerprinting: We train a second FlowLens application to identify webpages browsed through encrypted tunnels. We leverage the dataset made available by Herrman et al. [77]. This dataset has been widely used for the evaluation of novel website fingerprinting techniques [234, 152], and it contains traces of webpage accesses over OpenSSH. Websites are fingerprinted resorting to the Multinomial Naïve-Bayes classifier [77], which leverages the packet length distribution of the incoming and outgoing data in a connection as features. This classifier also allows us to illustrate how FlowLens can accommodate alternative truncation schemes whenever a given classifier does not return a ranking of feature importance (Section 3.7.5).

Botnet detection: Our last FlowLens application aims at detecting the presence of botnet chatter. We use the dataset produced by Rahbarinia et al. [158], which comprises traffic flows produced by four benign P2P applications (uTorrent, Vuze, Frostwire, and eMule), and two P2P botnets (Waledac and Storm). Malicious flows can be identified by analyzing packet length and inter-packet timing distributions resorting to a Random Forest classifier [158].

We simulate the classification of flows of a given target class in software based on a set of application-specific flow samples. We also configured all the classifiers to use the same hyperparameters suggested by the papers we drew our use-cases from. Throughout the evaluation, we assess the performance of different FlowLens configurations while exposing the system to a

Table 3.1: Scalability of FlowLens.

Use Case	FMA Configuration	Marker	Raw Dist.	Scaling
Covert Channels	$\langle QL_{PL}$=4, Top-N=10$\rangle$	20B	3000B	150×
Website Fgpt.	$\langle QL_{PL}$=5$\rangle$	94B	3000B	32×
Botnet Detection	$\langle QL_{PL}$=4, QL_{IPT}=6$\rangle$	302B	10200B	34×

workload that, to the best of our abilities, mimics those described in the literature. However, we highlight the adoption of a single holdout test instead of the cross-validation approach employed in other representative works [12, 134]. The reason is that, when applying truncation (Section-IV), FlowLens employs a pre-training step to obtain a feature ranking from the classifier. Then, it uses the top-N most important ones to fill the Truncation Table (Figure-4). Since cross-validation returns an average of the results obtained by multiple holdout models trained with different splits of the dataset, the resulting top-N features would not directly translate to be the top-N ones found in a particular model instance, namely in the model to be deployed on the switch for classification. Further, we chose a 50/50 holdout to increase the amount of unseen (test) data and better assess the generalization ability of the classifier.

3.7.2 Overall Performance

To give a general insight into the performance of FlowLens, Table 3.1 presents the scalability gains of our system when it is used to classify flows for covert channel detection, website fingerprinting, and botnet traffic detection while displaying an accuracy loss of at most 3% when compared with the use of complete packet frequency distributions. For these experiments, we generated the possible combinations of flow markers for the three considered use case scenarios, and assessed whether they allow for accurate flow classification. Packet lengths (PL) vary from 1 to 1500 bytes (MTU), and each cell of a flow marker has a size of 2 bytes.

These results show that, when the quantization and truncation parameters are properly fine-tuned (i.e., QL and truncation table), FlowLens can monitor at least 32 times more flows when compared to the baseline setup without compression, i.e., QL=0 and truncation disabled. Our system can also reach a 150 fold increase in its monitoring capacity when detecting covert channels. This is achieved for QL=4 and by selecting the top-10 most relevant bins for truncation. In this case, with a flow marker as small as 20 bytes, FlowLens manages to achieve a classification accuracy of 93%, only 3% shorter than the result obtained using raw packet length distributions. For website fingerprinting, the flow marker is larger (94 bytes) because we face a multi-class classification problem – different websites are better classified resorting to different bins. Thus, the truncation table is configured to map all quantized packet lengths. Lastly, for botnet chatter detection, we combine the quantization of packet inter-arrival time distribution (IPT) with the PL distribution. In this case, we achieve a marker size of 302 bytes which enables the bookkeeping of >30× flows.

In general, the number of flows that FlowLens can handle depends on the switches' available SRAM. The NDA we have signed with Tofino prevents us from disclosing the amount of switch memory but other sources [125] reveal that current switches feature hundreds of MBs of SRAM.

Table 3.2: Hardware resource consumption.

Resources	**Computational**			**Memory**	
	eMatch xBar	Gateway	VLIW	TCAM	SRAM
Usage	8.46%	5.21%	3.39%	0.00%	38.54%

3.7.3 Hardware Resource Efficiency

To evaluate the efficiency of FlowLens's hardware resource usage on the switch, we focus independently on the data plane and on the control plane. As for the data plane, Table 3.2 shows the average hardware resource consumption of FlowLens across all stages of the switch. The table shows that besides the SRAM required for the tables and register, the consumption of other resources is negligible. Since our flow matching logic entirely relies on exact matching, the FMA's flow table does not consume any of the TCAM resources on the switch. In tandem with the deployment of flow tables in SRAM, FlowLens leaves over 60% of SRAM available. Overall, these results suggest that FlowLens makes enough room for the concurrent execution of many other common forwarding behaviors, like access control, rate limiting or encapsulation, that do not necessarily require an extensive use of the stateful memory in the switch pipeline.

On the control plane, the switch has sufficient resources to fit all models used by FlowLens and to readily classify flows. In particular, the botnet chatter detection is our largest model, occupying only 140MB of memory, and 5.6MB of storage when compressed. In contrast, the model for covert channel detection uses only 64KB of memory and 24KB of storage. All these models comfortably fit within the control plane hardware resources, which has 32GB available RAM. Additionally, the flow classification step is very fast. For covert channel detection, once the flow markers have been collected from the data plane, the median of the time it takes for the classifier to output a label for a sample flow ranges approximately from 100 to 200 microseconds on the switch's Intel Broadwell 8-core general-purpose CPU operating at 2 GHz. These results indicate that flow classification can be efficiently conducted on the switch control plane.

Next, we present a set of micro-benchmarks which allow us to assess the benefits of our flow marker generation scheme. We will see that flow marker size (hence memory efficiency) tends to be more sensitive than classification accuracy to small variations in the quantization. The trend is the inverse for truncation, where accuracy is more sensitive to small variations than flow marker size. Our optimizer helps to find sweet-spot setups on the Pareto curve (Section 3.7.8).

3.7.4 Effects of Quantization

To study the effects of FlowLens's compression schemes, we first focus on the generation of flow markers for packet length distributions and start by analyzing the trade-offs of quantization. We present our main findings:

1. Multimedia covert channels can be detected with up to 92% accuracy using 188-byte flow markers. We leverage XGBoost to classify covert channels [12]. Figure 3.6 shows how the absolute values obtained for the accuracy, FPR, and FNR of the classifier vary when identifying Facet and DeltaShaper covert channels for different quantization levels (QL). For instance, for quantization level QL=4 FlowLens can correctly identify Facet and DeltaShaper flows with less than 5% and 1% decrease in accuracy, respectively. Table 3.3 shows that, for

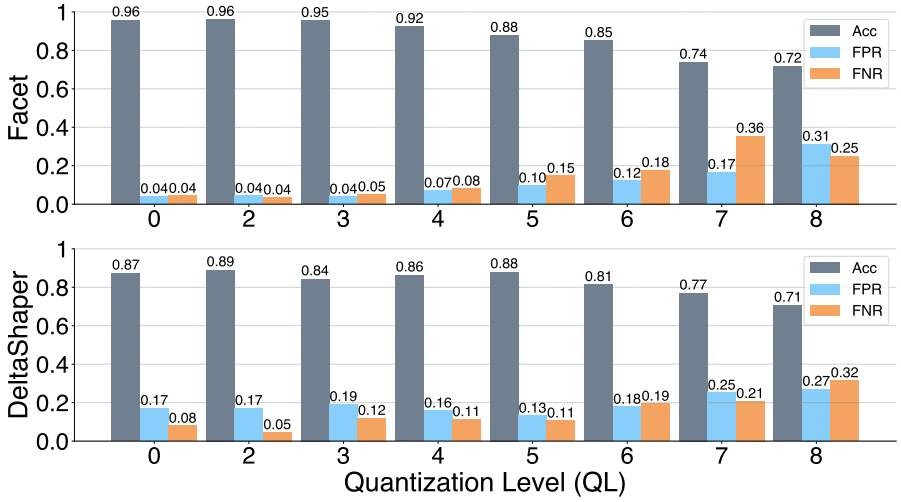


Figure 3.6: Accuracy, FPR, and FNR for multimedia protocol tunneling detection when using quantized packet length distributions.

Table 3.3: Flow marker size for different quantization levels.

Bins and Memory	**Quantization Level (QL)**							
`len(1 bin) = 2 Bytes`	0	2	3	4	5	6	7	8
Number of Bins	1500	375	188	94	47	24	12	6
Memory per Flow(B)	3000	750	376	188	94	48	24	12

QL=4, a flow marker can be represented in 94 bins instead of a full distribution composed of 1500 bins, amounting to an order of magnitude memory savings. While a single flow marker is then represented using 188B instead of 3000B, DeltaShaper classification scores are maintained with respect to those obtained when using full information (see Figure 3.6).

2. Accuracy of website fingerprinting is maintained when compressing flow markers by two orders of magnitude. For assessing the quality of FlowLens on website fingerprinting, we use the Multinomial Naïve-Bayes classifier [77]. We reproduced the multiclass closed-world website fingerprinting task for different quantization levels. Figure 3.7 shows that FlowLens is able to maintain the same classification accuracy up to a quantization level QL=3. Providing that classification accuracy can be relaxed in favor of memory savings, quantization can be further increased to QL=6, while still achieving over 90% accuracy and reducing a flow marker's memory footprint by two orders of magnitude.

3. Very coarse-grained flow markers are unsuitable for performing traffic differentiation. Figure 3.7 shows that flow markers can only be compressed to a given factor before causing a steep decrease in the quality of the models' predictions. For instance, in Figure 3.6, we see that for QL=7 the accuracy of the classifier is already over 20% and 10% away from the result obtained with full information for Facet and DeltaShaper, respectively. Thus, it is imperative to find the correct balance between memory savings and accuracy. FlowLens balances this trade-off, for different use cases, through a parameterization during the profiling phase.

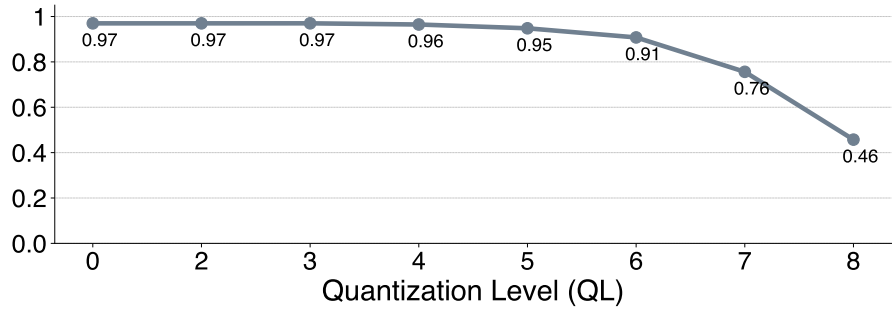


Figure 3.7: Accuracy of website fingerprinting when using quantized packet length distributions.

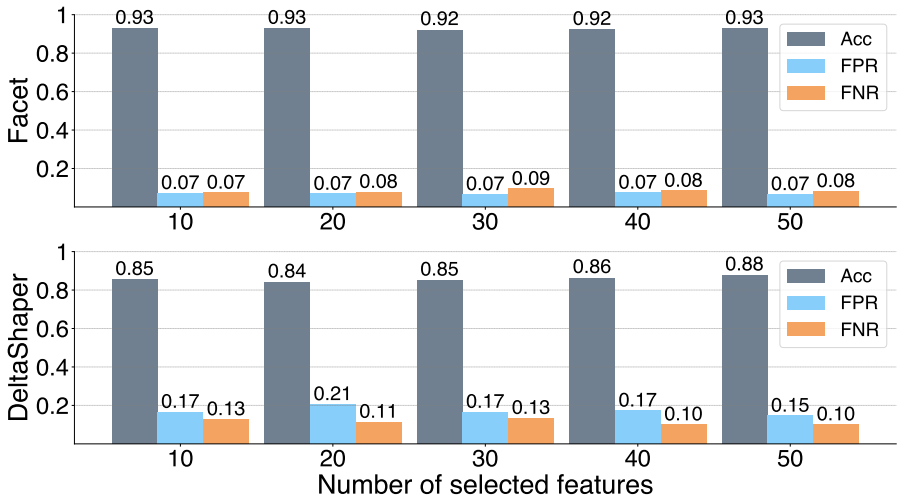


Figure 3.8: Accuracy, FPR, and FNR for covert channel detection with an increasing number of features for quantization level QL=4.

3.7.5 Effects of Truncation

The second mechanism to generate compact flow markers is that of truncating the flow marker to a subset of bins which make up for the most relevant features leveraged by the classifier. This is illustrated next, as we highlight our main findings after applying tailored truncation in different use cases.

1. Accurate detection of covert channels can be achieved using a flow marker of just 20 bytes. We elaborated a tailored truncation approach based on the importance of features computed by XGBoost. Figure 3.8 depicts the results obtained when performing quantization with QL=4 and truncating to the top-N most important features. The accuracy, FPR, and FNR rate of the classifier are practically identical when using the top-10 and top-50 features to classify flows (e.g., a difference of only 1% in FNR for Facet flows), and very similar to the results obtained when using full information (refer to QL=4 in Figure 3.6). Thus, truncation can not only maintain high accuracy, but further reduce the flow marker footprint from 188B (QL=4) to just 20B (QL=4, top-N=10).

2. 20-byte flow markers enable tracking 150× more flows. Covert flow markers can be reduced to just 20B using truncation. This corresponds to a 150× space-saving when representing

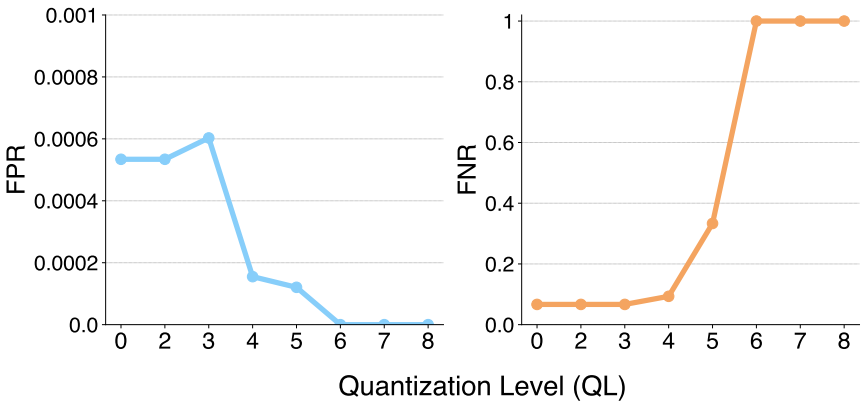


Figure 3.9: FPR and FNR for `www.amazon.com` at different quantization levels (QL). FNR shows the probability of identifying `www.amazon.com` as some other website. FPR shows the probability of some other website being classified as `www.amazon.com`. Truncation is applied at each QL.

Table 3.4: Number of bins used in `www.amazon.com` truncation.

Bins and Memory	**Quantization Level (QL)**							
`len(1 bin) = 2 Bytes`	0	2	3	4	5	6	7	8
Total Number of Bins	1500	375	188	94	47	24	12	6
Bins After Truncation	159	159	156	87	46	23	12	6

a flow (from 3000B to 20B). The space freed by compressing a single flow represents an increase in FlowLens's measurement capacity by two orders of magnitude.

3. Fingerprinting accesses to a website yields good results even when feature ranking is unavailable. The truncation method employed for covert channel detection is only applicable when considering classifiers able to output feature importance. To overcome the fact that Herrmann et al.'s [77] classifier is unable to output a rank of feature importance, we perform manual bin selection aimed at identifying a single website, e.g., `www.amazon.com`. Essentially, we first take a collection of access traces performed over a period of time to that particular website. Then, we simply discard the bins that correspond to packet lengths which have had zero counts of the sampled flows. Based on this selection, we then train our classifier accordingly. We can see in Figure 3.9 that the results obtained using this approach remain competitive. For instance, with quantization level QL=4, flows can be correctly identified with a 0.016% FPR and 9.333% FNR. As shown in Table 3.4, this flow marker footprint is not as small as with covert channel detection. Yet, it is practical to fingerprint website accesses with QL=4, yielding flow markers with a compression ratio of 1500:87, i.e., 17.2×.

3.7.6 Measuring Inter-Packet Timing

In this section, we concentrate on the ability of FlowLens to perform tasks that require both the inspection of packets' inter-arrival (IPT) and length (PL) distributions. To this end, we evaluate FlowLens in detecting P2P botnet chatter. Since the network traffic produced by bots tends to be stealthy and spread across time, packets sent in bot conversations are expected to

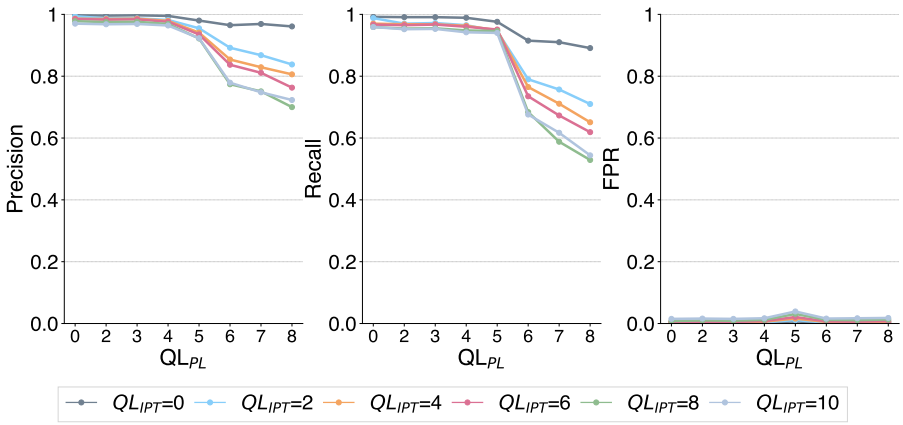


Figure 3.10: Precision, recall, and FPR for malicious P2P traffic.

have a higher IPT than those of legitimate P2P conversations. A conversation consists of the set of flows between any two hosts within a given time window, called *flowgap*. We resort to the Random Forest classifier originally employed by Narang et al. in PeerShark [134], and follow their recommendation to set flowgap to 3600s. Since the largest flowgap is set to 3600s, we vary the quantization of inter-arrival time down to a minimum of 4 bins ($\text{QL}_{IPT} = 10$).

Figure 3.10 depicts the precision, recall, and FPR obtained by the classifier when identifying botnet chatter for different QL applied to both PL and IPT distributions. While $\text{QL}_{PL} \leq 4$ slightly degrades precision and recall, we observe a sharp drop in both metrics when $\text{QL}_{PL} > 4$. The FPR, however, is not significantly affected by increasing quantization levels. Our experiments also reveal that precision and recall in the identification of legitimate P2P traffic are largely unaffected by the effect of quantization, whereas FPR takes a sharp increase for $\text{QL}_{PL} \geq 6$ (20% at $\text{QL}_{PL} = 6$ up to 50% at $\text{QL}_{PL} = 8$).

This figure also shows that it is possible to accurately identify botnet traffic with compact flow markers. For instance, $\langle \text{QL}_{PL} = 4, \text{QL}_{IPT} = 6 \rangle$ achieves a recall of 0.96, only 3% worse when compared to the result obtained when using full information (0.99). This accounts for a memory saving of 16× when storing a flow's packet length distribution, as well as occupying just 57 buckets×2B = 114B to keep an inter-packet timing distribution. These results suggest that FlowLens can offer different space-saving/accuracy trade-offs.

3.7.7 Performance of Automatic Profiling

To evaluate FlowLens's automatic profiling mechanism, we explore the parameter search space for each use case. For Facet and DeltaShaper, the search space includes the quantization and truncation parameters studied above (48 configurations). For website fingerprinting, the search space corresponds to 8 quantization configurations. For botnet detection, we consider 40 possible configurations based on packet length and IPT quantization, as we refrain from considering those whose $\text{QL}_{IPT} = 0$. We configure the optimizer to explore i=10 configurations for covert channel and botnet detection, and i=4 for website fingerprinting. For simplicity, we use no initial sampling for bootstrapping the optimizer, but techniques like Latin Hypercube sampling [184] may be also plugged in.

Table 3.5: Results of the profiling procedure, including the configuration output by the optimizer and the top-3 explored configurations (listed by decreasing accuracy, except for the case of Botnets which corresponds to malicious traffic recall).

Config. Rank	Facet (i=10)	DeltaShaper (i=10)	Website Fingerprinting (i=4)	Botnets (i=10)
#1	⟨QL=2, Top-N=all⟩ = 0.960	⟨QL=5, Top-N=all⟩ = 0.880	⟨QL=0⟩ = 0.970	⟨QL_{PL}=2, QL_{IPT}=2⟩ = 0.970
#2	⟨QL=3, Top-N=50⟩ = 0.951	⟨QL=0, Top-N=all⟩ = 0.873	⟨QL=4⟩ = 0.965	⟨QL_{PL}=0, QL_{IPT}=6⟩ = 0.969
#3	⟨QL=0, Top-N=30⟩ = 0.947	⟨QL=0, Top-N=20⟩ = 0.870	⟨QL=5⟩ = 0.948	⟨QL_{PL}=4, QL_{IPT}=6⟩ = 0.960
Output	⟨QL=3, Top-N=10⟩ = 0.944	⟨QL=5, Top-N=10⟩ = 0.840	⟨QL=4⟩ = 0.965	⟨QL_{PL}=3, QL_{IPT}=1024⟩ = 0.953

Fully automatic mode: Table 3.5 depicts the results obtained by our automatic profiler to choose an FMA configuration. In all cases, the profiler chooses a configuration that, albeit not the best accuracy wise, still provides a competitive accuracy while generating compact flow markers. For instance, for Facet, the top-3 configurations exhibit a marker size of 375, 50, and 30 bins, respectively. Our profiler chooses a configuration that provides a marker size of 10 bins while achieving an accuracy only 1.6% worse than the configuration with the best-found accuracy (and with a 37× smaller marker). Our reward policy leads the optimizer to perform good decisions over the explored configurations. In website fingerprinting, the profiler outputs the top-2 configuration rather than top-1 since the latter's marker size is too big in comparison (1500 vs 94 bins). The profiler also refrains from choosing top-3, a configuration whose marker is 2× smaller but less accurate. This trend can be observed for the remaining use cases.

Smaller marker for target accuracy: FlowLens can find a configuration that exceeds a minimum accuracy threshold, and that provides the smallest marker. For instance, we set a target accuracy of 0.85 for a DeltaShaper configuration. Among the 10 experimented configurations, the optimizer has found 3 candidate configurations with an accuracy larger than the set threshold. The system output ⟨QL=4, Top-N=30⟩ = 0.850, albeit finding ⟨QL=5, Top-N=40⟩ = 0.876 or ⟨QL=0, Top-N=40⟩ = 0.880, two other configurations which produced larger accuracy at the expense of a larger marker.

Best accuracy given a size constraint: The system is also able to find configurations with a larger accuracy value, given a maximum marker size. Additionally, and since the size of a marker can be computed offline without first trying a configuration, we achieve a reduction in the search space. In the case of DeltaShaper, setting a maximum marker size equal to 30 enables the reduction of the search space from 48 to 21 possible configurations. In this case, the optimizer outputs ⟨QL=2, Top-N=30⟩ = 0.890, albeit finding other smaller but less accurate alternatives such as ⟨QL=3, Top-N=20⟩ = 0.850.

3.7.8 Comparison with Related Approaches

In this section, we compare FlowLens against two related approaches: i) techniques which are able to produce compressed representations of packet distributions, and ii) techniques for collection of traffic features resorting to programmable switches.

Alternative feature compression approaches: Online Sketching (OSK) [45] and Compressive Traffic Analysis (CTA) [137] generate compressed packet length/inter-packet timing distributions using linear transformations. However, both approaches depend on matrix multiplications and/or floating-point operations unsupported by current switching hardware. Yet, we compare the classification accuracy of FlowLens against the accuracy obtained by OSK and CTA when using each technique to compress flow representations.

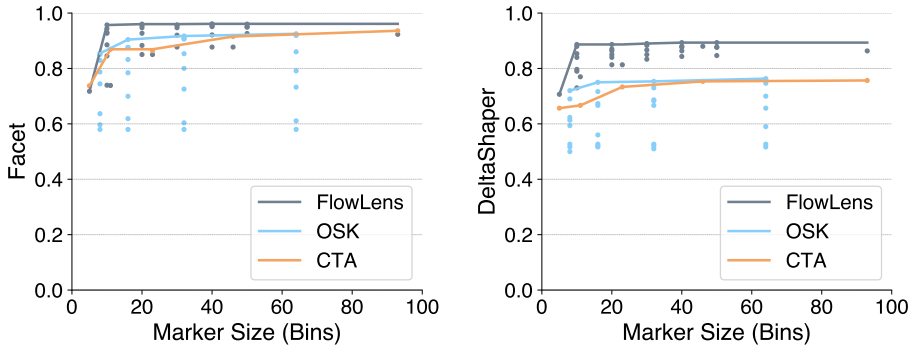


Figure 3.11: Pareto frontier for covert channel detection when using FlowLens, OSK, and CTA. Dots show individual configurations.

For evaluating the quality of the solutions yielded by the different compression techniques, we leverage the concept of Pareto optimality [142] which allows us to compare possible solutions to multi-criteria optimization problems (flow marker size vs. accuracy, in our case). A solution is said to be Pareto optimal if it cannot be improved in one of the objectives without adversely affecting the other. By generating the set of all of the potentially optimal solutions (Pareto frontier) for each approach, we can observe which approach delivers the best trade-offs between classification accuracy and marker size.

Figure 3.11 depicts the accuracy obtained in the classification of covert channels when using flow markers (with size up to 100 bins) generated by FlowLens, OSK, and CTA, while using different compression ratios. FlowLens configurations are achieved by combining the different quantization and truncation parameters. Solid lines represent the Pareto frontiers [113] that capture the best configurations for the three approaches. Overall, FlowLens produces flow markers that exhibit a better accuracy/memory trade-off and obtain the most accurate compressed representations of flows. For instance, in DeltaShaper, most FlowLens configurations achieve over 0.80 accuracy (and a maximum of 0.89 using a flow marker with a size of only 10 bins). In comparison, the most accurate OSK marker takes 16 bins and achieves an accuracy of only 0.76. A similar trend occurs in the case of Facet detection.

Alternative feature collection approaches: Systems such as *Flow [183] are able to collect fine-grained packet features at line rate from the switch and offload them to dedicated servers, where the packet distributions can be computed and analyzed by other dedicated systems for specific applications. FlowLens provides a complementary decentralized design where both the collection of packet distribution features and the application-specific analysis (i.e., flow classification) take place on the switches, thus achieving considerable savings in communication, compute, and storage hardware resources.

To estimate the potential gains of our design, we analyze the communication costs of both *Flow and FlowLens. Assuming the existence of 250k concurrent flows where each flow sends 15k packets during a collection window of 30 seconds, *Flow offloads data structures named *grouped packet vectors* (GPVs), each containing a flow key and a list of packet lengths, from a sequence of packets in a flow, on an average of 640ms [183] which totals 47 evictions. Since each GPV has a fixed header of 24 bytes, assuming 2 bytes to encode a packet length, *Flow must transfer $(24\text{B}\times47 + 2\text{B}\times15\text{k})\times250\text{k} = 7.78\text{GB}$ per collection window. This data would then need to be processed on a dedicated server. In contrast, FlowLens only transfers the classification score of

each flow at the end of the collection window which involves sending a fixed-size header per flow (13B for flow ID plus a 4B score value) times 250k flows, i.e., ≈4.25MB. Thus, FlowLens exhibits a communication footprint three orders of magnitude smaller than *Flow.

3.8 Security Analysis

We now analyze the security properties of FlowLens when functioning under an adversarial model. The overarching goal of the adversary is to be able to generate flows of a target application class without being detected by FlowLens. We consider three categories of increasingly sophisticated adversaries considering their knowledge about FlowLens and the models employed in ML-based security applications.

1. No knowledge about FlowLens nor the ML model: In the weakest threat model, the attacker knows nothing about the presence of FlowLens in the network, nor the details of the models being used by the ML-based security applications leveraging the capabilities of our system. In such a case, as shown in the sections above, ML-based security applications making use of the vanilla FlowLens setup can accurately identify different target classes of traffic.

2. FlowLens-aware adversary: In the second case, we consider an adversary that is aware of the deployment of FlowLens in the network infrastructure, but who is unaware of the particular machine learning models being used to filter the network for particular classes of traffic. In this case, the adversary may attempt to launch two particular types of attacks:

Flow aggregation attacks: An adversary may attempt to evade FlowLens's classifier by misusing the truncation and quantization steps to make the aggregation of flows of a given class of traffic indistinguishable from another class. In this sense, this type of attack is similar to our covert channel scenario (Section 3.7.4) where the adversary's goal is to mimic the distribution of legitimate traffic and evade a classifier. Figure 3.6 and Figure 3.8 show that a finer-grained aggregation of packet distributions does make it harder to evade the classifier. This suggests that increasing flow marker granularity makes FlowLens more robust against flow aggregation.

Evading collection windows: When analyzing long-lived network flows, FlowLens collects flow markers during a maximum pre-defined collection window. Once this window elapses, the FMA located on a given switch stops monitoring flows while the flow markers are read and FMA data structures are reset. An adversary may attempt to exploit this window of opportunity to transmit a class of traffic targeted by FlowLens during this period. However, as mentioned in Section 3.4.4, FlowLens can tolerate such attacks provided that multiple switches are used in an interleaved fashion to ensure that at least one switch can collect traffic pertaining to flows traversing the network.

DoS attacks: A FlowLens-aware adversary may also attempt to compromise the availability of our system. For instance, it may try to mount a DoS attack based on the transmission of packets with random IP addresses, forcing FlowLens to keep track of multiple dummy flows and waste the switch memory. To mitigate such a threat, FlowLens can temporarily prevent the installation of new rules in the FMA flow table when it detects unusual bursts of traffic, or reconfigure FMA parameters on-the-fly to store smaller (yet less accurate) flow signatures so as to increase the number of measured flows.

3. FlowLens and ML model-aware adversary: The third adversary we consider is cognizant of the operation of FlowLens and knowledgeable about the ML model used by a given ML-based

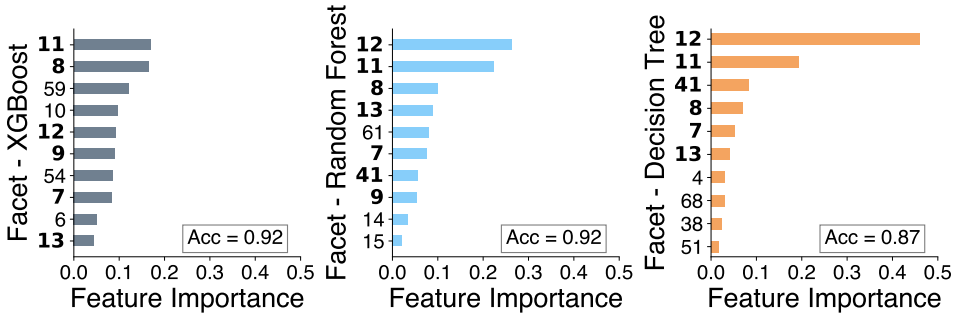


Figure 3.12: Features (PL bins) collected by the FMA for the ⟨QL=4, top-N=10⟩ configuration, when considering different classifiers to identify Facet covert channels. Features in bold are shared among at least two classifiers.

application. Apart from an adversary's attempts to evade or compromise the availability of our system, such an adversary aims to leverage adversarial ML techniques [4] to subvert the correct behavior of the model. These attacks can be grouped in two main categories [4]: i) *training-time*, where an attacker aims at manipulating the training set used by the ML model through the insertion of specific samples that alter the decision boundaries of the classifier; ii) *test-time*, where an attacker aims to evade classifiers by crafting traffic samples in such a way that these fool the classifier during its operational phase.

In general, providing defenses to such attacks is orthogonal to FlowLens's ability to collect flow markers and it concerns the particular models used by the different ML-based security applications. Nevertheless, FlowLens is compatible with various techniques aimed at increasing the robustness of the models used for traffic analysis. For mitigating *training-time* attacks, FlowLens's profiling phase can incorporate mechanisms aimed at filtering out contaminated instances upon training [114, 217, 68, 168]. Alternatively, FlowLens operators can leverage recent models whose training is explicitly hardened against the introduction of adversarial samples [33, 32, 190, 88]. For tackling *execution-time* attacks, FlowLens is compatible with the use of several techniques that increase the difficulty of an adversary to successfully evade network traffic classifiers. For instance, FlowLens can leverage classifier ensembles [1, 20, 110] or randomize the classifiers deployed at test-time [124].

Hardening FlowLens against adversarial ML attacks: We performed a simple experiment to understand whether FlowLens can leverage the above techniques to improve its robustness to adversarial attacks, while still collecting flow markers of small size. To this end, we profiled three different classifiers – XGBoost, Random Forest, and Decision Tree [12] – to identify Facet traffic in a ⟨QL=4, top-N=10⟩ FMA configuration.

Figure 3.12 depicts the importance of the top 10 features selected by the different classifiers after FlowLens's profiling step. Recall that, for a QL=4, there is a total of 94 features (bins), from which only the top-10 is considered. We draw two main observations from this figure. First, since all three classifiers share several features (marked in bold), crafting the traffic to subvert a given feature (e.g., feature 12), requires extra effort to collectively assess how it affects the classification accuracy not just of a single, but of all three classifiers. Second, each classifier selects a subset of features that are exclusive to it. Thus, while an adversary may shape a given flow to respect the features analyzed by a particular classifier, there may be another classifier that considers a different set of features. For instance, XGBoost leverages bins 59, 10, 54, and

6 to better inform a prediction, while the Random Forest classifier ignores these features and includes 61, 14, 15 in its top-10 instead.

To run multiple classifiers in execution-time, FlowLens must collect a superset of all meaningful features required by each model. Thus, it is expected that flow markers will increase their size. Figure 3.12 suggests that randomization, i.e., the random selection of one possible classifier, can provide a good compromise between robustness to adversarial ML and flow marker size. While a flow marker consists of 10 features for a given classifier (amounting to 20B), a flow marker that enables FlowLens to choose from three different models to classify flows uses a total of 18 distinct features, producing a flow marker amounting to just 36B. In a similar fashion, FlowLens could use all three classifiers to produce an ensemble which will ultimately classify a flow by majority voting [20].

Performance impact of the defense mechanisms: Although we have not empirically assessed the performance overheads caused by the proposed defense mechanisms, we argue that these mechanisms should not significantly impair the performance of FlowLens. Nevertheless, we reckon that they may require additional resources. The impact on resource allocation could be estimated, e.g., by measuring the memory consumed using ensembles, or by studying how many switches would suffice to plummet the risks of window evasion attacks.

3.9 Related Work

There is a considerable body of work proposing approaches for building efficient network telemetry systems for large scale networks [224]. Programmable switches can leverage TCAM-based flow tables for keeping flow data [195, 185, 133] and wildcard rules [26] to record a few statistics about a given flow [115]. The major drawback of this technique is tied to the limited size of TCAM which prevents the bookkeeping of more than a few thousand flows [225]. While multiple flows can be combined in the same table entry [230, 133], this aggregation jeopardizes the accurate representation of a large number of flows [225].

Traffic sampling techniques enable the collection of statistics for a large set of flows by recording a small number of packets of each flow [34, 141, 50]. Examples of general monitoring systems implementing sampling are OpenSample [186] or Planck [161]. Canini et al. [34] introduced a per-flow measurement technique that holds on the partial sampling of flows. However, the accuracy of sampling techniques is usually reduced when one aims at obtaining a faithful representation of a flow's distribution [106]. Moreover, increasing the sample rate is at odds with a larger memory footprint which can impact the overall performance of the network [83].

Probabilistic data structures known as sketches enable the error-bounded representation of flows' statistics within restrictive memory limits [225]. While multiple sketches allow for the extraction of flow's coarse-grained features [44, 222, 106, 111, 84, 221, 85], their strive for generality prevents recording fine-grained information such as approximations of flows' packet lengths and timing distributions. NetWarden [215] uses sketches to record approximate distributions of inter-packet timing distributions for a specific security task. In contrast, FlowLens can be broadly applicable to a range of ML-based applications. Coskun et al. [45] and Nasr et al. [137] explore additional ways to compress packet distributions based on the use of linear projections. Unfortunately, such techniques cannot be implemented efficiently in current switching hardware.

Recent systems relying on network query refinement [135], such as Turboflow [182] and *Flow [183], allow the data plane to offload simple packet features to servers for aggregation and processing. However, differently from FlowLens, such a strategy may increase the risks of network congestion and introduce scalability bottlenecks in large networks [108, 224].

3.10 Conclusions

This work proposed FlowLens, the first traffic analysis system for ML-based security applications that collects and analyses compact representations of flows' packet distributions – flow markers – within programmable switches. We evaluated our system for three use cases comprising the detection of network covert channels, website fingerprinting, and botnet chatter detection. FlowLens can accurately predict these classes of traffic flows with the help of compact flow markers, allowing for a reduction between one to two orders of magnitude of the memory footprint to represent packet distributions.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) via the SFRH/BD/136967/2018 grant, and the PTDC/EEI-COM/29271/2017, UIDB/50021/2020, and PTDC/CCI-INF/30340/2017 (uPVN) projects.

4 Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC

Publication Data

Diogo Barradas, Nuno Santos, Luís Rodrigues, Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 2020

Abstract

Many censorship circumvention tools rely on trusted proxies that allow users within censored regions to access blocked Internet content by tunneling it through a covert channel (e.g,. piggybacking on Skype video calls). However, building tools that can simultaneously (i) provide good bandwidth capacity for accommodating the typical activities of Internet users, and (ii) be secure against traffic analysis attacks has remained an open problem and a stumbling block to the practical adoption of such tools for censorship evasion.

We present Protozoa, a censorship-resistant tunneling tool featuring both high-performing covert channels and strong traffic analysis resistance. To create a covert channel, a user only needs to make a video call with a trusted party located outside the censored region using a popular WebRTC streaming service, e.g., Whereby. Protozoa can then covertly tunnel all IP traffic from unmodified user applications (e.g., Firefox) through the WebRTC video stream. This is achieved by hooking into the WebRTC stack and replacing the encoded video frame data with IP packet payload, while ensuring that the payload of the WebRTC stream remains encrypted, and the stream's statistical properties remain in all identical to those of any common video call. This technique allows for sustaining enough throughput to enable common-use Internet applications, e.g., web browsing or bulk data transfer, and avoid detection by state-of-the-art traffic analysis attacks. We show that Protozoa is able to evade state-level censorship in China, Russia, and India.

The reproduction of this publication was slightly adapted to adhere to formatting requirements. The original version of this publication can be found at: https://dl.acm.org/doi/abs/10.1145/3372297.3417874.

4.1 Introduction

State-level censors are known to apply techniques to prevent free access to information on the Internet. In fact, many countries have deployed a vast censorship apparatus to exercise control over available content, namely China [94], Russia [160], Iran [5], Bangladesh [132], India [219], Thailand [65], or Syria [37]. For instance, amidst the recent Coronavirus outbreak, the Chinese government has shut down news websites [76] and instructed Chinese social media platforms to censor references related to the infection [36, 170], in an attempt to handle the sharing of negative coverage within and outside the country. This control can be enforced through various techniques, such as keyword-based filters [218, 216], image filters [96], social platform monitoring [95, 78], and even the entire blocking of Internet destinations [146] or selected protocols [55].

To evade censorship, many circumvention tools have been proposed for enabling users to freely access/share information on the Internet [91, 192]. Typically, such tools rely on covert channels to allow for the stealthy transmission of sensitive data through an apparently innocuous carrier medium [91, 226], e.g., a multimedia streaming carrier application like Skype. The key idea is to encode the covert data in such a way that an adversary capable of inspecting the full packet exchange cannot distinguish between a legitimate transmission and one that subliminally carries covert data. This approach, which we call *multimedia covert streaming*, can be achieved in two ways: i) by entirely mimicking the carrier's network-level protocols [129] (*media protocol mimicking*), or ii) by embedding the covert data into the video (or audio signal) feed of the carrier application in the course of a regular video call [82, 105, 121, 10, 97] (*raw media tunneling*).

Given that all the carrier's traffic is encrypted, one way for an adversary to counter potential covert channels is to blatantly block all traffic generated by the carrier application. This method, however, can bring harmful side-effects even for a state-level adversary. In fact, considering how instrumental many media streaming applications are for the tissue of economic and social interactions within censored regions, the costs of shutting down popular applications can be overwhelming and erode even further the state's reputation in the eyes of its international peers. Leveraging on essential streaming applications can then serve as a strong deterrent for the enforcement of blocking policies, and constitutes the key insight that favors the effectiveness of multimedia covert streaming [58].

Nevertheless, an adversary can employ a second class of techniques based on traffic analysis. Essentially, it involves probing into the censored network region, inspecting the traffic generated by a presumable carrier application, and looking for discrepancies in the traffic (e.g., abnormal patterns) that might signal the presence of covert channels. Hence, it follows that an effective tool for multimedia covert streaming must be able to resist these kinds of attacks by exhibiting traffic patterns that will ideally be indistinguishable from legitimate traffic. In other words, when picking from two sampled flows – one legitimate flow and one crafted flow containing covert data – the adversary must not be able to distinguish them but by random guessing, i.e., with 50% chance of success.

Unfortunately, the existing tooling support for multimedia covert streaming is quite bleak. Several studies revealed that the presence of modulated covert traffic can be detected solely based on the analysis of traffic features such as packet sizes and packet inter-arrival times [66, 12]. In fact, machine learning (ML)-based traffic analysis techniques can effectively detect small changes in the packet frequency distributions caused by the embedding of covert data inside carrier video streams, namely deviations in the packet sizes and inter-packet arrival times when compared with legitimate traffic. Most existing tools fail this test and are prone to be detected

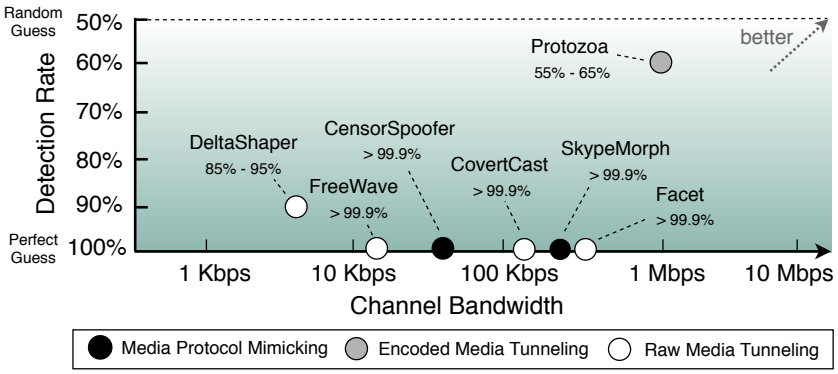


Figure 4.1: Design space of multimedia covert streaming tools along two dimensions: covert channel capacity (X-axis), and traffic analysis resistance (Y-axis). Darker shades indicate increasing chances of detection, i.e., tools are more insecure. Protozoa outperforms the existing systems in both dimensions. A detailed analysis of this plot is found in Section 4.9.1.

with high accuracy rates [12]. While some tools like DeltaShaper [10] can tolerate detection to some degree, they do so at the expense of reducing the amount of covert data embedded into the cover video stream, severely limiting the covert channel bandwidth capacity that can be attained. For instance, in DeltaShaper, the maximum achievable throughput is only 7 Kbps, which is clearly insufficient for sustaining the traffic generated by common Internet users, e.g., interactive web browsing, media streaming, or bulk data transfers.

This paper presents Protozoa, a new multimedia covert streaming tool that provides good performance for the covert transmission of arbitrary IP traffic while featuring strong resistance to detection when subjected to ML-based traffic analysis by a state-level adversary. In particular, Protozoa allows an Internet user (*client*) located in a censored region to access blocked content by leveraging the help of a trusted user in the free region who will act as a *proxy* on the client's behalf. Protozoa enables then to create a bidirectional *covert tunnel* between both endpoints. Henceforth, the client can start browsing the web freely: the local IP traffic will be transparently redirected through the covert tunnel to its final destination host in the free region, e.g., YouTube. This local application is not restricted to a browser: Protozoa can tunnel IP traffic from arbitrary unmodified applications, e.g., email or BitTorrent clients.

Protozoa advances the state-of-the-art by incorporating two new ideas in its design. First, to enable the transmission of covert traffic, it uses web streaming applications based on WebRTC which are very popular and widely disseminated. Concretely, to create a covert tunnel, all that two users – client and proxy – need to do is to establish a video call using a WebRTC-enabled web streaming website, such as Whereby (`https://whereby.com`). Protozoa uses the video call's associated WebRTC media stream to tunnel covert IP traffic between both endpoints. Second, to encode the covert signal into the carrier stream, Protozoa introduces a technique named *encoded media tunneling*, which allows for boosting the capacity of covert channels while offering strong resistance to traffic analysis. It consists of embedding the covert data into *encoded video frames*, i.e., right after the lossy compression has been applied by the video codec. This mechanism is implemented by modifying the WebRTC stack of Protozoa's Chromium browser component.

We extensively evaluated our Protozoa prototype both through a set of microbenchmarks resorting to media sessions established over Whereby, and by testing it in various realistic usage scenarios and workloads. Our results showed that, under normal network conditions, Protozoa

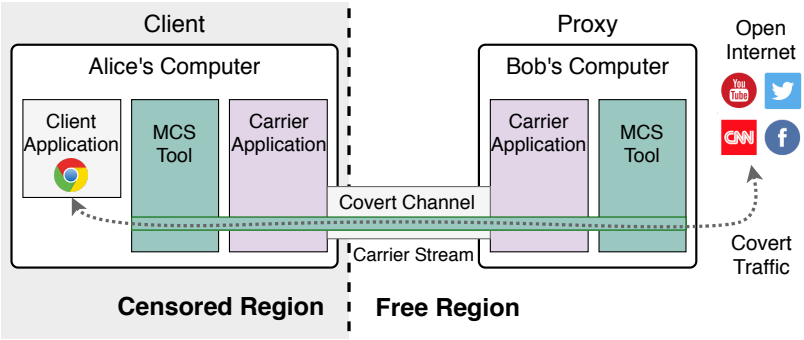


Figure 4.2: System model of a multimedia covert streaming tool.

can deliver covert channel bandwidth capacities in the order of 1.4Mbps and channel efficiency of 98.8%, while providing strong resistance to traffic analysis using state-of-the-art ML-based techniques [12]. As illustrated in Figure 4.1, these results represent a significant departure over existing media covert streaming techniques, dramatically improving the detection rate of the best performing tool, i.e., Facet, from >99.0% to 55%-65%, i.e., from nearly perfect guess to very close to random guess, while improving covert channel bandwidth by 3×. Additional experiments show that Protozoa can withstand a number of active network perturbations without jeopardizing its resistance against traffic analysis or breaking the covert channel connection.

To assess the portability of Protozoa, we also tested our system on alternative WebRTC services: `appr.tc` and `coderpad.io`. Our results showed that it can consistently achieve similar throughput and traffic analysis resistance properties when used over different applications, which makes it useful in scenarios where specific applications are blocked (e.g., `appr.tc` is blocked in China). Lastly, we deployed Protozoa in three regions that are known to enforce Internet censorship through several means: the Great Firewall of China (GFW) apparatus in China, and ISP censorship in Russia and India. We performed several experiments from servers deployed in each of these regions. First, we accessed blacklisted content without using Protozoa, and checked that it was indeed inaccessible. Then, using Protozoa, we were able to access this content, showing that our system can successfully breach through existing censorship mechanisms deployed in these regions, and provide free access to blocked Internet content.

4.2 Threat Model

The general system model of a *multimedia covert streaming* (MCS) tool is illustrated in Figure 4.2. It represents two users, one acting as *client* (Alice), and a second acting as *proxy* (Bob). The client is located in a censored region controlled by a state-level adversary, and the proxy is based in a free Internet region. The adversary is able to observe, store, interfere with, and analyze all the network flows within its jurisdiction, and block the generalized access to remote Internet services, such as CNN, Facebook, or Twitter, by the residents in the censored region. The censorship policies can be based on the IP address or the domain name of the target destination, the protocol used in the communication (e.g., BitTorrent or Tor), or blacklisted content (e.g., through keyword and image filtering).

An MCS tool aims at enabling the client to overcome the communication restrictions enforced by the adversary by leveraging (i) the cooperation of a proxy located in the free region operated by a trusted Internet user (Bob), and (ii) a carrier application consisting of an encrypted video-streaming service (e.g., Skype) whose traffic the adversary authorizes to cross the boundaries of the censored region. Both users – client and proxy – must run the MCS software on their local computers to create a covert tunnel through the media stream managed by the carrier application. This tunnel allows the client to contact remote hosts on the open Internet.

To defeat an MCS tool, the adversary can use deep packet inspection for pinpointing traffic indicators that lead to the detection of a covert channel. To increase its chances for successful detection, it may apply statistical traffic analysis techniques over collected network traces [12]. The adversary may also launch indiscriminate active network attacks aimed at perturbing the correct behavior of covert channels lurking under seemingly legitimate flows while ensuring that legitimate flows maintain a reasonable quality.

However, the adversary will seek only to rapidly disrupt and tear down those flows which are suspected of carrying covert channels, and it will refrain from blocking the carrier application altogether if such an application is reckoned to provide an important service to the population. The adversary is also deemed to be computationally bounded, and unable to decrypt encrypted traffic generated by the carrier application. The adversary's control is also limited to the network: it has no access to the persistent or volatile state of clients, proxies, or carrier application provider, and enjoys no privileges over the software that can be executed by each party.

4.3 Parasitizing on WebRTC Streams

In designing Protozoa, we elected WebRTC media streams as the carrier medium for deploying practical, efficient, and secure covert channels. WebRTC [112] is a W3C standardization initiative for protocols and APIs for enabling secure real-time communication between web browsers. All major browsers have built-in WebRTC implementations enabling the generalized use of this technology.

WebRTC creates new opportunities for building MCS services that can simultaneously be widely available and easy to use. By acting at this layer, any WebRTC-powered application can be transparently used for covert data transmission. WebRTC has been currently adopted by numerous services that integrate real-time communication capabilities[1]. This integration has been greatly facilitated by the simplicity of the JavaScript WebRTC API [67]. The generalized usage of web conferencing for professional dealings, in particular, will make it very deterring for a state-level adversary to block all WebRTC traffic due to the extensive collateral damage to the country's own sustainability [58]. This profusion of WebRTC services also gives Protozoa users a great deal of flexibility and options to choose from when it comes to picking the carrier medium for covert transmission. Creating covert sessions is in itself a user-friendly operation since establishing a Protozoa connection is in no way different from making a video call on a web streaming application (including the process of joining a chatroom URL).

[1]There are many different WebRTC-based web streaming applications, such as social applications like Whereby or Facebook Messenger, the gaming-focused chat Discord, professional video conferencing services like Amazon Chime and Slack, remote coding interview software like Coderpad, or even health monitoring through Vidyo.

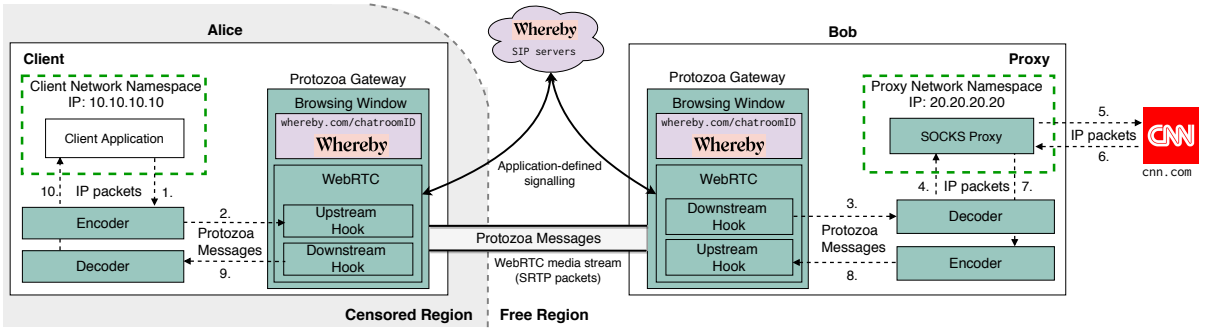


Figure 4.3: Architecture of Protozoa: The components of our system are highlighted.

Given that a widely-used WebRTC stack is openly available (in the Chromium web browser), we have full access to the WebRTC video streaming pipelines which allows us to develop a new efficient and secure covert data encoding technique named *encoded media tunneling*. In existing raw media tunneling tools, the covert data is encoded as pixels of the carrier video frames. Unfortunately, given that the raw input signal undergoes lossy compression by the video codec of the carrier application, a significant amount of redundancy must be included to enable the covert data recovery by the receiver which degrades the utilization efficiency of the covert channel. Moreover, using up all possible pixel area of the carrier frames for encoding covert data reduces the video codec's compression efficiency, which increases the network packet sizes and deforms the packet size frequency distribution when compared to that of legitimate flows. Because this discrepancy can be detected by an adversary, the covert channel tools must be parsimonious at using up the video stream's pixel area thereby throttling even further the bandwidth capacity of the covert channel. Encoded media tunneling overcomes these limitations and allows for boosting the capacity of covert channels while offering strong resistance to traffic analysis attacks. Next, we describe our technique as we present Protozoa.

4.4 Protozoa

This section presents Protozoa, a system that provides covert channels over WebRTC media sessions[2]. Next, we present its architecture and then describe its most relevant technical details.

4.4.1 Architecture

Figure 4.3 depicts the architecture of Protozoa. In Protozoa, the carrier application consists of a web application that uses the WebRTC framework for providing a point-to-point live streaming service between its users, e.g., Whereby. Typically, such an application consists of a backend that handles the signaling and session establishment of video calls between participants, and client-side JavaScript & HTML code that initiates video calls and manages the video transmission via the WebRTC API provided by the browser. The resulting media stream will be used by Protozoa as the carrier for a covert channel.

[2] "Protozoa" alludes to parasitic biological organisms which feed on other organisms.

The MCS tool is materialized by the Protozoa software bundle which targets the Linux platform, and it is set up differently to work as client or proxy by two participating parties. The client will be able to execute unmodified IP applications whose traffic can seamlessly be routed through the WebRTC covert channel by the proxy to destination hosts anywhere in the free Internet region. Figure 4.3 shows Alice (client) executing a local application for accessing `cnn.com` using Bob's computer as proxy.

The Protozoa software bundle comprises four main components: gateway server, encoder service, decoder service, and SOCKS proxy server. Both client and proxy run the gateway server and the encoder and decoder services. The proxy also runs the SOCKS proxy server. Both endpoints leverage network namespaces for transparent interception and manipulation of IP packets. All these components cooperate in forwarding covert IP traffic by implementing three cross-cutting functional layers. Next, we explain how these layers operate by introducing them in a bottom-up fashion.

1. WebRTC layer: This layer is responsible for the setup and management of a point-to-point WebRTC covert channel between two parties, and it is implemented by the gateway servers running on each communication endpoint. The covert channel is piggybacked on a WebRTC media stream instantiated by a carrier web streaming application. This channel supports full-duplex bidirectional communication and exchanges Protozoa messages defined in a specific format. These messages can contain arbitrary IP payload. The gateway server is built out of a modified Chromium browser, and instrumented with the placement of two hooks – *upstream* and *downstream* – in the WebRTC stack. We leverage Chromium's functionality to provide a web browsing UI and runtime environment which will allow for the execution of the client-side WebRTC application code. The hooks intercept the WebRTC streams so as to replace the payload of the WebRTC video frames with covert Protozoa messages. The gateway server opens two pipes for receiving upstream and downstream messages from the codec layer.

2. Codec layer: This layer performs two complementary encoding and decoding operations. The former is responsible for encoding streams of IP packets generated by local networked applications. Packets are read from a `libnetfilter` queue [140], and encapsulated into Protozoa messages, which are forwarded to the local gateway server and then delivered to the remote endpoint. Decoding performs the reverse operation, i.e., reads incoming Protozoa messages from the local gateway server, extracts the enclosed IP packets, and writes them to a raw socket to be routed to their final destination. These operations are coordinated by the encoder and decoder at both endpoints to sustain simultaneously two IP packet flows, i.e., upstream and downstream. Internally, these components maintain packet and message queues, and implement packet fragmentation and reassembly so as to efficiently use the covert channel capacity.

3. SOCKS layer: This layer enables the exchange of IP packets between the networked applications running on the client, and a remote Internet host through a SOCKS v5 proxy server running on the proxy. This is achieved by the use of Linux's network namespaces and configuration of `iptables`. Namespaces are implemented by the Linux kernel and allow for the creation of virtual network interfaces. In our context, we use namespaces for creating a virtual network environment for the client application and a second one for the SOCKS proxy server. Each environment features a virtual network interface that is exposed to the local processes with a specific IP address, e.g., 10.10.10.10, or 20.20.20.20, respectively (see Figure 4.3). Protozoa then configures the local `iptables` so as to route all (upstream) IP packets with destination address 20.20.20.20 to the namespace of the proxy, and all (downstream) IP packets with destination address 10.10.10.10 to the namespace of the client. Thus, by configuring a client application to use

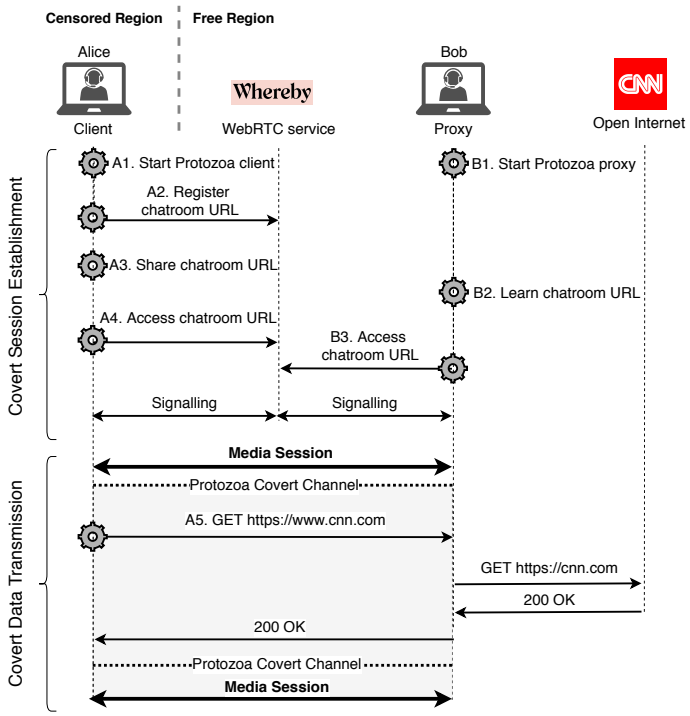


Figure 4.4: Covert session: gear symbol denotes user actions.

the IP address 20.20.20.20 as SOCKS proxy, all its IP connections will be transparently delivered to the SOCKS server proxy, which in turn will deliver the packets to its remote destination. So, for instance, the web request to `cnn.com` depicted in Figure 4.3 can be performed by running the `curl` command on a Linux terminal as follows:

```
$ ip netns exec PROTOZOA_ENV_CLIENT
  curl -x socks5h://20.20.20.20:1080 https://cnn.com
```

This means that, in order to use Protozoa's covert tunnels, the user must configure the client application to use a SOCKS proxy. For client applications that do not natively support the use of SOCKS proxy servers, the user can use an additional tool, `proxychains` [166], which provides the client application with SOCKS proxy support.

4.4.2 Execution Workflow

This section describes the execution workflow involved in a complete communication using Protozoa covert tunnels. Using the example depicted in Figure 4.3, we describe the full message exchange sequence that takes place in order for Alice to fetch a web page from `cnn.com` through a WebRTC covert tunnel facilitated by Bob, who is an individual volunteer trusted by Alice. This tunnel is created through a WebRTC video call between Alice and Bob using Whereby in the course of a Protozoa *covert session*, which is divided into two stages: covert session establishment, and covert data transmission. Figure 4.4 represents the messages exchanged.

1. Covert session establishment: The covert tunnel is set up between client and proxy, requiring both participants to agree on a common rendezvous point for a WebRTC media

connection. In our example, Alice and Bob use the web browsing interface of their Protozoa gateway to join a common video chatroom. They begin by bootstrapping the Protozoa software: Alice in client mode (A1), and Bob in proxy mode (B1). Then, Alice accesses `whereby.com`, creates a password-protected chatroom, and obtains the chatroom URL (A2). Similarly to using alternative MCS tools [10, 105], Alice uses an out-of-band channel, e.g., email, social network web site, or mobile app (e.g., Whatsapp) to share the chatroom URL and password with Bob (A3 and B2). Both users can now join the chatroom (A4 and B3) and initiate a video call by feeding a carrier video stream from their local cameras or (optionally) from a prerecorded video; this video will be replaced by covert payload. As the WebRTC video stream is initiated, Protozoa hooks into it, and sets up the covert tunnel.

2. Covert data transmission: Once the covert tunnel is ready, Alice can access remote Internet services. For instance, to access `cnn.com`, Alice can simply run the `curl` command listed in the section above to issue an HTTP GET request to `cnn.com`. The IP traffic generated from this request will be transparently tunneled through the covert channel. Protozoa will continuously stream video until the termination of the covert session, even when there is no covert traffic to be transmitted; in this case, dummy payload (chaff) is sent.

4.4.3 Network-level Security of Covert Sessions

At covert data transmission, standard WebRTC ensures that all exchanged packets are integrity-protected and the message payload containing sensitive video data is encrypted. Thus, an adversary cannot read its content, or modify it without detection. Nevertheless, we must ensure that the covert session has been securely established. In particular, an adversary may attempt a man-in-the-middle or an impersonation attack during the session negotiation phase (see Figure 4.4) enabling it to decrypt the message payload and inspect the covert data. To prevent these attacks, Protozoa leverages the security mechanisms implemented by WebRTC and by the carrier WebRTC web streaming application, namely the following ones:

a) HTTPS: Client and proxy run client-side code of the WebRTC web application which connects to its backend servers through HTTPS. This means that all messages involving interactions with the backend (i.e., A2, A4, B3) will be exchanged over TLS-enabled secure channels. In particular, this prevents an adversary from obtaining the URL that would allow it to join the chatroom, or to mount a MITM by advertising different URLs to client and proxy.

b) SIP / DTLS-SRTP: To establish a media session, WebRTC leverages the Session Initiation Protocol (SIP) to signal one endpoint's intention (e.g., the client's) to connect to its corresponding peer (e.g., the proxy). This protocol involves the communication between each endpoint (client/proxy) and a SIP server run by the WebRTC application provider (see Figure 4.3). This server is used to exchange media session parameters between endpoints, and it is combined with the DTLS-SRTP protocol [118, 223] to perform an initial key exchange so as to offer protection against man-in-the-middle attacks. The WebRTC application provider also runs a STUN server which helps the endpoints located behind a NAT to determine their respective public (NAT'ed) IP addresses, and share them with their peers. To ensure that the media sessions between endpoints are not hijacked and pointed to different IP locations, the connection attempts to the IP addresses of target endpoints are secured by a MAC, which is computed using the key exchanged in the signaling channel [169]. Once a WebRTC session has been established, WebRTC leverages the Secure Real-time Transport Protocol (SRTP) [17, 223] for encrypting and authenticating the content of the media in transit.

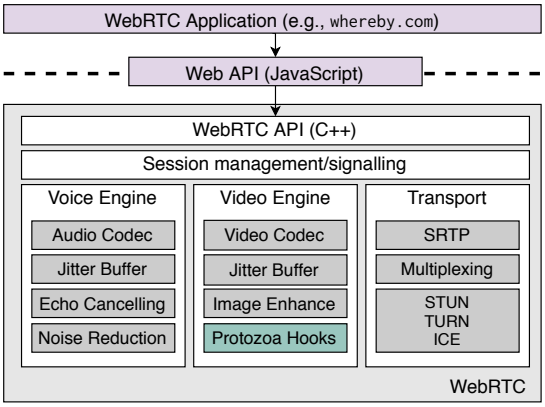


Figure 4.5: WebRTC software stack and Protozoa's hooks.

4.4.4 Encoded Media Tunneling

Protozoa uses the video streams generated by client and proxy as a medium for carrying covert IP packet data in both directions. To this end, we employ a new approach named *encoded media tunneling*. Similar to existing raw media tunneling techniques [10, 105], our method replaces carrier video information with a covert message. However, instead of replacing the pixels of the raw input video, it replaces the bits of the encoded video signal, i.e., *after* the input video has been compressed by the WebRTC video codec. This technique helps increase not only the capacity of the channel but also its resistance to traffic analysis. In our system, it is implemented by instrumenting the WebRTC stack of the Protozoa gateway.

Upstream and downstream hooks: Figure 4.5 illustrates the WebRTC stack as it is implemented in the Protozoa gateway. It is based on the WebRTC stack bundled into the Chromium browser. The WebRTC stack contains a built-in codec (VP8), which processes the video signal of local web applications that use the WebRTC API. To access the video frames generated by the WebRTC application and implement encoded media tunneling, the WebRTC stack includes two hooks that can intercept the processing of the media stream in different directions, i.e., upstream or downstream. The *upstream hook* intercepts outgoing frame data, i.e., from a local camera device to the network. It is placed after the raw video signal has been processed by the video engine, and right before the frame data is passed over to the transport layer where SRTP packets are created, and sent to the network. The *downstream hook* intercepts incoming frame data, i.e., from the network to the local screen. It is placed right after the transport layer has finished reconstructing an encoded frame sent in multiple network packets, and right before handing it over to the video engine to be decoded and rendered on screen. We strategically placed hooks in the WebRTC stack in order to manipulate a special data structure, named *encoded frame bitstream partitions* (EFBP), where we can embed Protozoa messages.

Using EFBP as a covert data mule: To help understand how the covert data is embedded into the carrier frame data, Figure 4.6 depicts the format of the SRTP packets, which is the means through which video data is exchanged. The bulk of SRTP packet space is reserved for the transmission of media payload in the field named *encoded frame bitstream* (EFB). This field contains the bits of an encoded (compressed) video frame as it is generated by VP8, the default WebRTC codec. An encoded frame contains a small (3-10 bytes) uncompressed header, and two partitions which carry compressed bitstreams containing actual carrier video data. We call EFBP

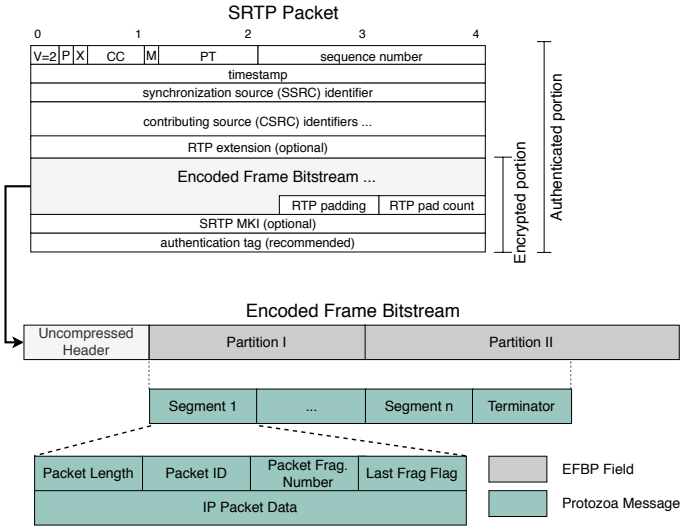


Figure 4.6: Format of SRTP packets: Protozoa replaces the EFBP payload containing carrier video bits with covert data.

the contiguous space occupied by these partitions. The EFBP has five extremely interesting properties for our problem domain:

1. The EFBP consists of a blob of bits that contains the actual video data of the carrier stream. Since this information is irrelevant for us, we can effectively use this field for carrying covert data by overwriting it with covert data.

2. The EFBP, once it is generated by VP8, is no longer modified by the WebRTC downstream pipeline. This means that the covert data bits placed in this field are not going to be corrupted, e.g., due to compression or other destructive operations, before being sent to the network.

3. The EFBP will be used as payload of the SRTP packet, and contains no relevant metadata that influences the transport layer logic, hence, modifying this field will not disturb the normal functioning of packet transfer over the network.

4. The EFBP, prior to being assembled into SRTP packets, will be encrypted, and protected with authentication markers. This means that covert data placed inside the EFBP will be encrypted and integrity-protected for free.

5. The EFBP will be encrypted resorting to a stream cipher that preserves the plaintext size, therefore, embedded covert messages do not change the size of encrypted EFBPs.

For all these reasons, we use the EFBP as a free storage space for transmitting covert data in the form of Protozoa messages.

Protozoa message transmission: To embed covert IP packets inside an EFBP, Protozoa uses the message format depicted in Figure 4.6. A single message consists of multiple segments followed by a Terminator (i.e., a zero-length segment) which delimits the EFBP area occupied by covert data. Each segment carries an entire IP packet or a packet fragment. IP packet fragmentation may be required at the sender's endpoint shall the next available IP packet be

larger than the available EFBP space on the frame; this process will help to use the covert channel in its maximal capacity. As a result, a covert IP packet can be transmitted in a single segment or span across multiple segments. Each segment has a small header that allows the receiving endpoint to reassemble IP packet fragments.

Message transmission works as follows. For every new frame generated by the video engine, there is an opportunity for sending a new message, which causes the upstream hook to be executed and given access to the encoded frame. The hook accesses the EFBP data structure, checks its size, and tells the encoder service how much free space exists in the frame for sending covert data. The hook waits for the encoder to assemble a new message containing locally queued IP packets. Then, the hook copies it into the EFBP and returns, letting the WebRTC pipeline to proceed with its normal execution until the resulting SRTP packets are transmitted. At the receiving endpoint, the reverse operation is performed. Whenever an encoded frame is reassembled, the downstream hook is executed and extracts the Protozoa message from the EFBP field. This message is sent to the local decoder service, which reassembles the ingress IP packets and forwards them to their rightful destination.

Prevent video decoding malfunction: While it is possible to fully replace the content of the EFBP field, the undisciplined corruption of a frame bitstream can prevent the video decoder in the WebRTC downstream pipeline from correctly decoding video frame data at the receiver's endpoint. In fact, we empirically verified that in such situations, WebRTC triggers congestion control mechanisms in the downstream pipeline for ensuring the reception of video. In particular, it advertises a Picture Loss Indication (PLI) in the accompanying RTCP control channel [163], aimed at requesting the retransmission of a key frame upon being unable to decode the corrupted frame data. In particular, VP8 produces two different types of encoded frames. Key frames can be decoded without any reference to previous frames and provide seeking points within a video stream. Delta frames are encoded with reference to the prior key frame and ensuing frames. By advertising PLIs and sending key frames upon detecting corrupted frames, the resulting traffic patterns produced by WebRTC applications would make Protozoa vulnerable to traffic analysis.

To overcome this problem, the downstream hook feeds the WebRTC video decoder with a pre-recorded sequence of valid encoded frames instead of the corrupted frames received over the network. Since the encoded frames may either be key frames or delta frames, the downstream hook uses the uncompressed header information kept intact after covert data embedding (see Figure 4.6) to decide which type of frame and corresponding resolution (e.g., 640x480) should be provided to the native WebRTC video decoder. Then, it restores the corrupted bitstream with a bitstream of a valid frame. This allows us to establish a covert channel where the size of egress frames on the upstream pipeline is maintained, and to deliver the decoder valid data so that it does not trigger congestion control.

4.4.5 Implementation and Optimizations

We developed a Protozoa prototype [8] by writing approximately 3,000 lines of C++ code. This includes the instrumentation of the native WebRTC codebase of the Chromium browser v79.0.3945.117, a stable release from January 2020. Protozoa requires the proper establishment of a WebRTC video session for embedding data into encoded frames sent over the wire. To this end, WebRTC must be able to access a video feed that can be directly obtained from the physical camera device available in the system. Alternatively, it is possible to set up a camera emulator by using the `v4l2loopback` kernel module [193] and feed recorded video with the help

of the `ffmpeg` video library [57]. In Section 4.6, we leverage the latter method for evaluating Protozoa in light of different video profiles.

Fine-tuning of IP packet queues: We performed an important optimization related to the size of the packet queues maintained internally by Protozoa's encoding service. Specifically, we refer to the queue that holds intercepted IP packets generated in the upstream pipeline network namespace. A typical rule-of-thumb for managing packet queues suggests the parameterization of a buffer size according to the following formula: *Buffer Size* $\geq$ *RTT* * *Channel Bandwidth* [119]. According to our experimental results, we conservatively assume that each packet in the queue has a size equal to the MTU. Additionally, we empirically verify that the round-trip-time experienced by our system is $\approx$200ms when connected in a LAN network, i.e., when the latency between WebRTC hosts is sub-millisecond. Configuring our packet queue with the above parameters yields a queue size of 24 packets for the 200ms round-trip-time experienced by Protozoa and a bandwidth of approximately 1.4Mbps achieved when sending video at 640x480 resolution.

4.5 Evaluation Methodology

This section describes our evaluation methodology for assessing the quality and performance of our Protozoa prototype.

4.5.1 Evaluation Goals and Approach

The goal of our experiments is twofold: i) evaluate the performance of Protozoa's covert channel in face of different network conditions, and ii) assess the ability of our system to resist against detection from an adversary able to perform statistical traffic analysis attacks.

To measure the performance of the covert IP flows tunneled by Protozoa covert WebRTC session, we leverage `iPerf`. This enables us to stress the covert channel capacity.

When testing our system's ability to resist traffic analysis attacks, we aim to reproduce the ideal conditions for the adversary. Essentially, the attacker's aim is to analyze the statistical properties of WebRTC's media (SRTP) and control (RTCP) packet flows so as to identify Protozoa traffic among legitimate WebRTC media sessions. To this end, we apply a state-of-the-art traffic classifier [12], which leverages two different sets of features: i) quantized packet size distributions, and ii) summary statistics computed from packet size and inter-arrival time distributions. Then, we collect a balanced dataset composed of legitimate and Protozoa WebRTC packet traces, and measure the AUC achieved by the above classifier when performing binary classification using 10-fold cross-validation. Note that, in the wild, class imbalance is expected to be skewed towards the abundance of legitimate streams and would likely make the adversary's task harder than in a controlled lab environment [38]. To ensure that the collected traces reflect realistic convert traffic transmissions, we keep the channel busy by injecting artificial chaff into the covert tunnel (using `iPerf`) while collecting these traces.

4.5.2 Experimental Testbed and Datasets

Our laboratory testbed, illustrated in Figure 4.7, is composed of four 64-bit Ubuntu 18.04.5 LTS virtual machines (VMs) provisioned with two virtual 2,3 GHz Intel Core i5 CPU cores and

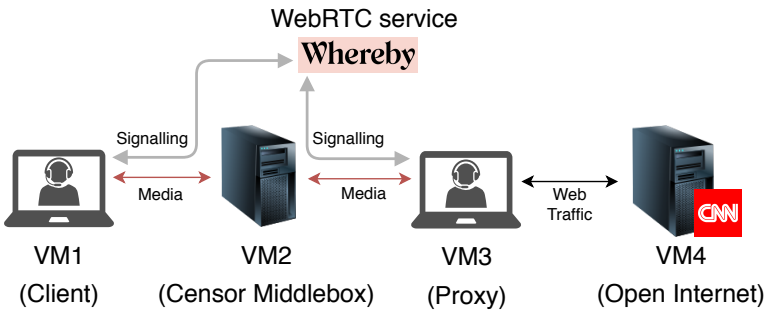


Figure 4.7: Laboratory setup.

16GB of RAM. VM1 and VM3 execute an instance of our prototype, operating as a Protozoa client and proxy, respectively. VM2 acts as the gateway and router for the two Protozoa VMs, and mimics the operation of a censor middlebox by collecting packet traces required for conducting statistical traffic analysis. Finally, VM4 is used to pose as a server in the open Internet which receives requests from the Protozoa proxy in VM3 acting on behalf of the client in VM1.

To conduct our experiments, we collected a total of 2000 YouTube video samples from four different categories (500 videos each) labeled by hand. These categories focus different video profiles assumed to be common in WebRTC services, and which we identify as *Chat*, *Coding*, *Gaming*, and *Sports*. For generating packet traces pertaining to legitimate and Protozoa media sessions, we split each of the four datasets (one for each video profile) in half. Then, we establish 250 legitimate WebRTC connections and 250 Protozoa connections while mirroring the video transmitted on each side of the connection. This allows us to avoid the contamination of the training data by mixing the same video samples in both legitimate and Protozoa connections. Video is set to be transmitted at 30fps and at a 640x480 resolution over the `whereby.com` WebRTC service, unless stated otherwise. Packet traces are collected for a duration of 30 seconds, a time interval shown in prior work to be sufficient for accurate detection of MCS streams using state-of-the-art statistical traffic analysis [12]. As described in Section 4.6, we validate that the use of longer traces did not significantly affect our results.

4.5.3 Metrics

We adopt a set of metrics for evaluating Protozoa's covert channel performance and resistance against statistical traffic analysis:

Performance metrics: In order to be able to compare Protozoa to existing work, we leverage *throughput* as the metric of performance of the covert channel. Additionally, we are interested in measuring Protozoa's covert channel *efficiency*, which provides the ratio between the total amount of data transmitted in the covert channel and the total available space in encoded video frame bitstreams.

Security metrics: Akin to earlier studies on the resistance of MCS systems to traffic analysis attacks, we use the following metrics to evaluate Protozoa's traffic analysis resistance capability: *true positive rate* (TPR), *false positive rate* (FPR), and the *area under the ROC curve* (AUC). The TPR measures the fraction of Protozoa flows that are correctly identified as such, while the FPR measures the proportion of legitimate flows erroneously classified as Protozoa flows. An adversary aims at obtaining a high TPR and a low FPR when performing covert traffic

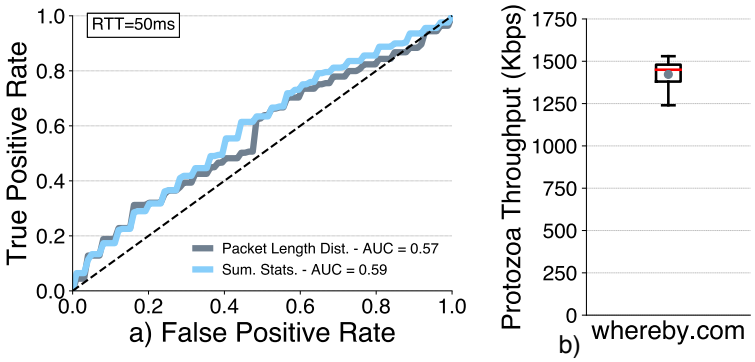


Figure 4.8: Baseline traffic analysis and performance results.

classification. The Receiver Operating Characteristic (ROC) curve plots the TPR against the FPR for the different possible cutout points for classifiers possessing adjustable internal thresholds. The AUC [56] summarizes this trade-off. Note that an AUC of 0.5 is equivalent to random guessing.

In the following sections, we evaluate our prototype by resorting to a set of microbenchmarks and by conducting a number of experiments when deploying Protozoa in real-world scenarios.

4.6 Evaluation using Microbenchmarks

In this section, we evaluate Protozoa using a series of microbenchmarks. We test our system on a baseline scenario and then study the effects of varying the network and carrier conditions.

4.6.1 Baseline Deployment

Protozoa can be evaluated in multiple scenarios that depend on many factors (e.g., carrier video, carrier WebRTC application, or network conditions). Since validating all these dimensions is a hard endeavor, we first present an analysis of Protozoa based on a *baseline* deployment which gathers a set of conditions expected to be found in a real-world deployment of Protozoa.

Our baseline deployment encompasses the following configuration. First, we select Whereby, a popular WebRTC application, as the carrier application for the Protozoa covert channel. Second, we select the videos comprising the *Chat* dataset as carrier media. Third, we assume that the round-trip-time (RTT) between Protozoa endpoints (VM1 - VM3) is in the order of 50ms, a typical value for connections established within the same continent [197, 147]. Lastly, we assume a 15ms RTT from the Protozoa proxy to an open Internet service (VM3 - VM4). This value is reasonable even when accessing foreign services due to the proliferation of CDN edge servers which may be regionally co-located with a Protozoa proxy [147, 194].

4.6.2 Baseline Performance Results

We now evaluate Protozoa's resistance against traffic analysis and assess the throughput and efficiency of the covert channel in the baseline deployment presented in the previous section.

Duration (s)	**10**	**20**	**30**	**40**	**50**	**60**
AUC	0.56	0.60	0.59	0.58	0.58	0.61

Table 4.1: Classifier's AUC for varying trace durations.

Traffic analysis resistance: Figure 4.8a) depicts the ROC curve of the classifier when attempting to identify Protozoa connections resorting to two sets of features: quantized packet size distributions, and summary statistics. Firstly, we see that summary statistics provide a better overall detection rate, enabling the classifier to obtain an AUC of 0.59 (for the remainder of our evaluation, we will limit ourselves to present the results corresponding to the use of summary statistics). Secondly, the ROC curve shows that a censor would incur in a large FPR when blocking Protozoa flows resorting to the state-of-the-art classifier. Essentially, the FPR represents the collateral damage that results from setting the TPR to a specific cutoff value. As an example, if we assume that the censor would like to block 80% of all Protozoa flows (TPR = 0.8), it would erroneously flag approximately 60% of all legitimate flows as covert channels (FPR = 0.6). Although the cutoff FPR value is determined in a discretionary fashion by each censor (i.e., possibly withstanding different TPR/FPR tradeoffs), the figure shows that distinguishing between Protozoa streams and legitimate media streams is close to random guessing.

To assess the robustness of Protozoa for packet traces of different durations, we repeated the same set of experiments using trace lengths up to 60 seconds as depicted in Table 4.1. These results suggest that the size of the traces has no meaningful impact on the AUC, given that the measured AUCs exhibit small fluctuations between 0.56 and 0.61. Thus, in the interest of scaling up our experiments, we conducted our remaining evaluation resorting to 30s traces.

Performance: Figure 4.8b) depicts a boxplot showing the throughput achieved by Protozoa's covert channels. We can observe that, under the baseline deployment conditions, Protozoa achieves an average throughput of 1422 Kbps, while the 90th percentile sits at 1510 Kbps, and the 75th percentile at 1480Kbps. This amounts to a throughput increase of 3× when compared to Facet, and a 3-fold order of magnitude increase when compared to DeltaShaper.

Additionally, we analyzed the efficiency of Protozoa's covert channel by measuring the ratio between the data embedded in each outgoing frame and the size of the frame. When using iPerf to stress the upstream covert channel link, we observed that Protozoa used 98.8% of the available frame space to transmit covert data. This suggests that our packet encoding scheme can use the majority of the encoded frame bitstream to transfer covert data.

Lastly, regarding resource consumption, the client and proxy VMs peaked at a 21.6% usage of their total CPU and at 596MB of memory usage. These numbers suggest that Protozoa can be executed on various commodity hardware platforms.

In the next sections, we evaluate our system beyond our baseline setup across multiple other network deployment scenarios.

4.6.3 Varying Network Conditions

Assessing the security of our prototype in face of different network conditions is paramount i) to understand whether Protozoa can remain undetectable in practical deployment scenarios, and ii) to ascertain whether our system can withstand active network perturbations introduced

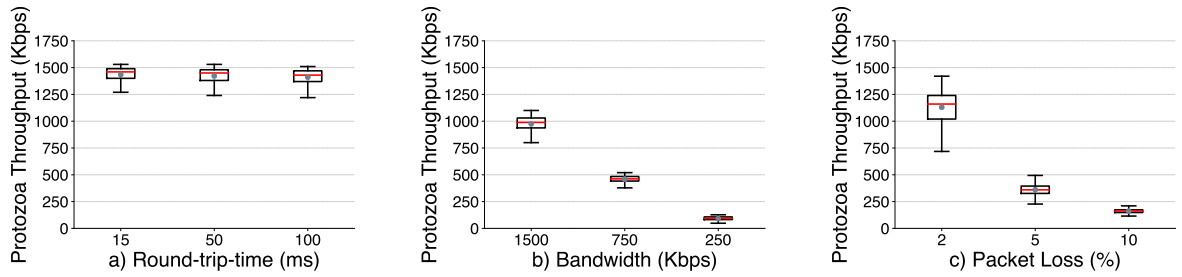


Figure 4.9: Throughput of Protozoa's covert channel on different network conditions.

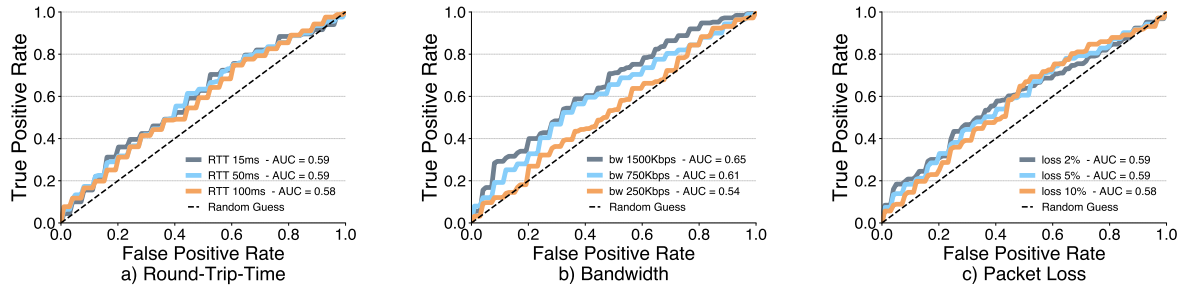


Figure 4.10: ROC AUC obtained when detecting Protozoa flows on different network conditions.

by a network adversary, aimed at disclosing the operation of the system or at breaking the covert channel connection. To manipulate network conditions, we leveraged the traffic control facility Linux NetEm [75] and varied the network conditions in the following dimensions: i) *latency*, ii) *bandwidth*, and iii) *packet loss*. A recent study [86] reported that WebRTC connections can withstand a limited range of network perturbations before being torn down due to QoS constraints. We bound the network perturbations to inject in the network according to the ranges suggested by this study. Next, we detail the results of our experiments.

Latency variation of the covert channel: We delay packets so as to achieve a round-trip-time (RTT) of 15ms, 50ms, and 100ms between Protozoa endpoints (VM1 - VM3). These values emulate regional, intra-continental, and inter-continental RTTs [197, 147], while being kept within the bounds of 300ms, recommended for establishing real-time multimedia sessions with acceptable quality [187].

Our results are as follows. Figure 4.9 a) illustrates the breakdown of throughput achieved by our prototype as the RTT between Protozoa endpoints increases. It shows that the latency introduced between endpoints does not impact the throughput obtained by Protozoa. In particular, the throughput remains at an average of about 1420Kbps in the three configurations tested. Figure 4.10 a) presents the ROC curves obtained by the classifier when attempting to detect Protozoa in a network with different RTT configurations. We see that the classifier obtains a maximum AUC of 0.59. Thus, irrespective of the latency introduced between endpoints, the adversary does not obtain an advantage at distinguishing Protozoa flows.

Bandwidth variation of the covert channel: We symmetrically limit the bandwidth of the link to 1500Kbps, 750Kbps, and 250Kbps, beyond the unrestricted bandwidth conditions assumed in our baseline case. In these conditions, WebRTC streams use approximately 80% of the available bandwidth, agreeing with other studies [86].

Figure 4.9 b) shows that the achievable throughput tends to decrease as bandwidth is more scarce. For instance, while Protozoa's throughput averages 975Kbps when bandwidth is capped at 1500Kbps, it drops to 460Kbps at 750Kbps, and attains an average 91Kbps when bandwidth is only 250Kbps. This effect is expected since the constrained amount of bandwidth leads to the decrease of frame rates and forces the downgrade of video resolution and encoded frame size, thus reducing the available space for embedding covert data. For instance, a 250Kbps bandwidth cap only allows a WebRTC stream to obtain ≈25 FPS at a low 480x270 resolution [86].

As for resistance to traffic analysis, Figure 4.10 b) reveals that the bandwidth variation does not provide sufficient information for the classifier to accurately distinguish between legitimate and Protozoa connections, peaking at 0.65 AUC when the bandwidth is limited to 1500Kbps.

Packet loss rate variation of the covert channel: We assess the properties of Protozoa when the media channel is subjected to packet losses. Following the experiments of Jansen et al. [86], we drop 2%, 5%, and 10% of the packets pertaining to WebRTC connections. Each of these loss rates causes WebRTC to increase sending rate (2%), slowly increase sending rate (5%), and converging data rate to values leading to the tear-down of the video stream (10%). Typical recommendations for real-time media traffic sit at no more than 1% packet loss [187].

Figure 4.9 c) shows that while Protozoa's throughput is negatively affected by increasing packet loss rates, our prototype is still able to sustain an average throughput of 1130Kbps for packet loss rates of 2% and of 360Kbps for a loss rate of 5%. While a 10% packet loss substantially decreases the throughput to 160Kbps, we note that Protozoa's covert channel connections remained active and did not break for the duration of the experiment.

Lastly, Figure 4.10 c) shows that Protozoa preserves high-levels of traffic analysis resistance when the network link between Protozoa endpoints is subject to variable packet loss rates.

Latency variation at the last mile: We now focus on the impact of the RTT between Protozoa's proxy (VM3) and open Internet services (VM4) to the network performance. Figure 4.11 b) shows the breakdown of throughput achieved by Protozoa when the RTT between VM1 and VM2 endpoints is set to 50ms and the last leg of the connection to the Internet service ranges from 15ms to 100ms.

Similarly to our earlier experiments when varying the latency between Protozoa endpoints, the throughput is not compromised when latency increases between the Protozoa proxy and the Internet service. The average throughput of our system is rather stable, at around 1410Kbps, for the three tested configurations.

4.6.4 Varying Carrier Conditions

We also evaluate our system varying two carrier-specific conditions:

Varying video profiles: This experiment aims to test whether different video profiles used as cover media affect the throughput of our system. Such a question arises since variable bitrate video encoders, such as the ones used in WebRTC (e.g. VP8), adjust the amount of output data according to the complexity of encoded video segments. To answer this question, we evaluate the performance of the covert channel when established through the *Chat*, *Coding*, *Gaming*, and *Sports* video profiles in our baseline deployment.

Performance-wise, Figure 4.12b) depicts the throughput achieved by Protozoa when using different video profiles as cover media. We see that our system achieves a similar average

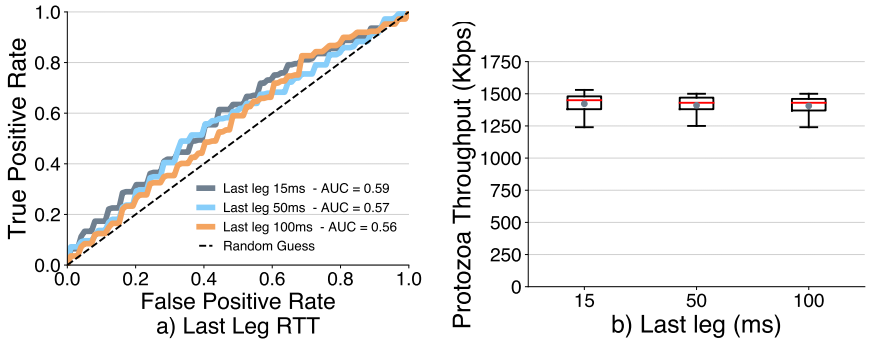


Figure 4.11: Throughput and traffic analysis resistance obtained while varying the RTT between VM3 and VM4.

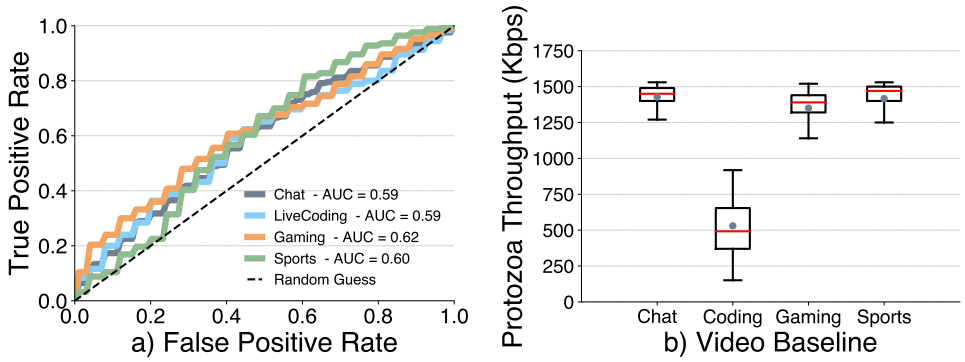


Figure 4.12: Throughput and traffic analysis resistance while using different video profiles.

throughput of approximately 1400Kbps for *Chat*, *Gaming*, and *Sports* media flows, while reaching an average throughput of 530Kbps when transmitting *Coding* media. These results concur to the observation that a large portion of video frames remain static in live coding videos. Additionally, these numbers suggest that the throughput is consistent within each baseline, achieving a maximum standard deviation of 157Kbps across the *Chat*, *Gaming*, and *Sports* baselines.

In turn, Figure 4.12a) shows the ROC curves for the classifier when attempting to distinguish Protozoa connections conducted over the different video profiles. The classifier achieves a similar AUC for all profiles (≈0.6 AUC), suggesting that the resistance against traffic analysis is preserved irrespective of the video profile used as cover.

Varying WebRTC services: To assess whether the properties of our system hold when media calls are established over multiple WebRTC applications, we conducted further experiments over two additional WebRTC services: a) `coderpad.io` – a live coding interview application – and b) `appr.tc` – Google's bare-bones demo application based on the simple WebRTC WebAPI.

Figure 4.13a) depicts the ROC curves for the classifier when inspecting Protozoa streams established through `coderpad.io` and `appr.tc`. The results show that the classifier obtains an AUC of 0.58 for streams established over `coderpad.io` and an AUC of 0.60 for streams established over `appr.tc`. In both cases, we see that Protozoa remains undetectable by a network eavesdropper.

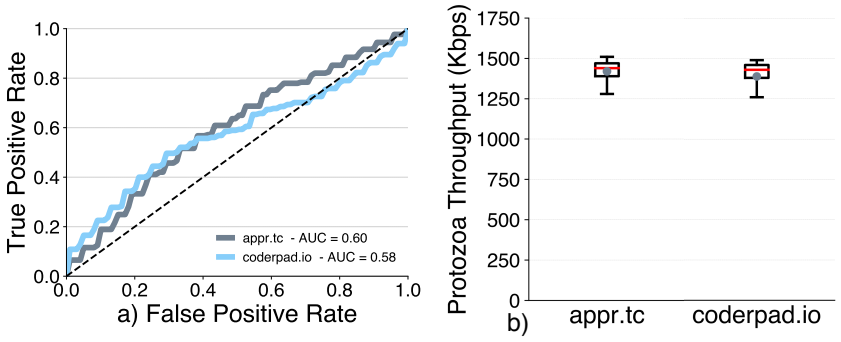


Figure 4.13: Throughput and traffic analysis resistance obtained with other WebRTC services.

Application	Protocol	Workload
A. Curl	HTTP, FTP	Page transfer (16 MB) in 89s
B. Transmission	BitTorrent	File transfer (2GB) in 3h5m
C. Mutt	SMTP	Send email (1KB) in 5.5ms
D. Irssi	IRC	Send text message (80B) in 0.44ms
E. VLC	HTTP	Video streaming at 20/14 fps in 240p/480p
F. Firefox	HTTP	Web-surfing session

Table 4.2: Real application workloads over Protozoa.

As for the throughput when establishing a covert channel resorting to the two alternative WebRTC applications, Figure 4.13b) shows that Protozoa achieves an average throughput of 1420Kbps for `appr.tc` and of 1388Kbps for `coderpad.io`, similar to what was obtained with `whereby.com`, as presented previously in Section 4.6.2.

4.7 Testing in the Wild

In this section, we test our system in multiple real-world settings. First, we present the results of an experiment comprising the execution of a set of typical workloads conducted by Internet users, over Protozoa. Then, as proof of concept, we show that Protozoa is able to evade the censorship apparatus of real-world adversaries by using it to access censored content in China, Russia, and India.

4.7.1 Testing with Real Application Workloads

We tested Protozoa with multiple networked applications as depicted in Table 4.2. We used the baseline setup presented in Section 4.6.2. To tunnel the traffic of applications that do not natively support a SOCKS proxy, such as `mutt`, we leverage `proxychains` [166], a SOCKS proxy wrapper, to redirect such traffic through Protozoa.

First, we used Curl (A) over the covert channel to download files with sizes ranging from 1KB to 256MB using both the HTTP and FTP protocols. We also verified that Protozoa is able to uniformly distribute the bandwidth of the covert channel among simultaneous Curl connections. In order to test Protozoa's covert channel on a different transport protocol, we configured the Transmission BitTorrent client (B) to download a popular Linux distribution ISO. We also found

Protozoa to be successful when operating email – Mutt (C) – and instant-messaging – Irssi IRC chat client (D) – applications. Additionally, we tested the ability to stream video content over VLC (E). Lastly, we ran Firefox (F) over Protozoa to navigate different web pages and to stream videos from YouTube, confirming that Protozoa enables interactive web-surfing tasks.

Overall, the results of our experiments show that Protozoa is able to accommodate a number of common Internet applications, including web-browsing, video streaming, or bulk data transfer.

4.7.2 Evading State-Level Adversaries

To test Protozoa's ability to circumvent real-world censors, we ran Protozoa in three locations known to be active targets of Internet censorship – China [22], Russia [160], and India [219].

First, we identified sets of web pages that are blocked for each country [149, 228, 188]. We selected web pages in several categories: gambling, pornography, news/politics, drug sale, and circumvention tools. For each set, we verified that the web pages could not be directly accessed (i.e., without using Protozoa) using Firefox running on a server physically deployed in the respective country. Then, we repeated this access after setting up Protozoa covert sessions. In each server, we configured a Protozoa client to establish a covert channel towards a Protozoa proxy located in LA, USA, and used this channel to access the blocked web pages. To run our experiments, we resorted to virtual private servers (VPSs) in Shanghai, Moscow, and Mumbai, respectively, and deployed the Protozoa bundle amounting to a total of ≈150 MB.

Blocking policies: We observed that browsing blocked websites in Russia and India resulted in ISP blockpages, whereas in China they would simply not load properly. A closer look at the traffic traces produced when trying to access blocked web pages revealed that the GFW performs packet drops on connections aimed at blacklisted hosts. This observation is consistent with the behavior of the GFW [22]. Browsing a blocked website from within Russia and India showed that blocking policies implemented within datacenters are less restrictive from those applied on typical ISP connections, i.e., we could access websites which would trigger the return of ISP blockpages when browsed over a VPN; therefore, given that our VPSes are located inside datacenters, for ensuring a reliable Protozoa testing, in Russia and India we route all VPS traffic through a VPN server hosted in the same country, where we obtain blockpages when visiting forbidden websites. This differentiation, however, did not occur on the VPS within China, where the pages found to be blocked when browsing over a VPN inside the country were also blocked when accessing from our VPS in a datacenter.

Availability of WebRTC services: Paramount to the functioning of Protozoa is the ability to connect to a foreign WebRTC service. Since Protozoa makes no assumption over the WebRTC application used as a vehicle for the covert channel, it is only necessary to find one unblocked application within the censored region. Table 4.3 shows that multiple WebRTC applications are available in the countries focused in our evaluation. Importantly, the table shows that, despite several WebRTC applications being blocked in China, a user still has plenty of alternative WebRTC media applications that can be used as a carrier for Protozoa covert channels.

Reaching censored content: To reach our blocked page sets, we leveraged `whereby.com` to establish Protozoa connections. We were able to access all such blocked websites in China, Russia, and India.

WebRTC Application	Reachability		
	China	Russia	India
appr.tc	-	✓	✓
aws.amazon.com/chime	✓	✓	✓
codassium.com	✓	✓	✓
coderpad.io	✓	✓	✓
discordapp.com	-	✓	✓
gotomeeting.com	✓	✓	✓
hangouts.google.com	-	✓	✓
messenger.com	-	✓	✓
slack.com	✓	✓	✓
whereby.com	✓	✓	✓

Table 4.3: Reachability tests on popular WebRTC services.

4.7.3 Ethical Considerations

The experiments conducted in this section involve the access to censored content from a number of vantage points within countries known to experience Internet censorship. These accesses raise important ethical concerns since they risk triggering reprisals from local authorities. We followed the best practices described in the Menlo report [49] to guide three major decisions of our experimental design. First, we did not recruit volunteers for our experiments. Instead, we rented VPSes from commercial VPS providers which understand the legal implications of offering network and computing services in each country they operate. Second, albeit using the signalling infrastructure of existing WebRTC applications, Protozoa does not compromise in any way the integrity of such applications. Covert traffic is exclusively forwarded by replacing user-generated video content. Lastly, we did not collect any sensitive user data.

4.8 Security Discussion

We now discuss some potential attacks to Protozoa and defenses:

Packet dropping: An adversary may instrumentally drop a small number of selected packets of WebRTC media streams in an attempt to dramatically slow down the covert data transmission or disrupt the functioning of Protozoa protocols causing, in either case, a denial of service. In contrast to other systems [66, 80], Protozoa is robust against these attacks since it does not rely on specific packets for managing covert channels. Moreover, Section 4.6.3 shows that applications that use Protozoa's covert channels are able to tolerate a large percentage of dropped packets without terminating.

Active probing: Active probing attacks aim at identifying Protozoa proxies, e.g., by attempting to join some chatroom and identify the transmission of corrupted video streams which telltale the presence of covert channels. By selecting WebRTC chatrooms that implement member admission controls, e.g., using passwords or contact list checks, Protozoa can evade this attack.

Fingerprinting of cover videos: If Protozoa is set up to stream a pre-recorded cover video, an adversary may attempt to identify a particular user by using that video for fingerprinting Protozoa covert channels. This threat can be countered by: i) rotating the pre-recorded video, ii) or feeding a live video from the local camera.

Long-term user profiling: An adversary may keep track of a user's interactions with WebRTC services so as to build a profile of interactions with multimedia applications. An accurate profile may enable an adversary to indirectly detect the usage of Protozoa through connections with out-of-ordinary duration or by detecting the placement of calls at unusual times of the day. Assessing the feasibility of this threat is an interesting direction for future work.

4.9 Related Work

We now describe past approaches aimed at evading Internet censorship and locate Protozoa in the spectrum of existing techniques.

4.9.1 Comparison with Similar Systems

Protozoa fits in the family of multimedia covert streaming systems. It stands out by introducing a new technique – encoded media tunneling. Next, we compare our system against two other branches of this family (Figure 4.1 puts all these systems in perspective.)

Media protocol mimicking: Previous systems have introduced traffic morphing [211] techniques for the transmission of covert data by imitating multimedia protocols. For instance, by entirely replacing the payload of media packets by encoded data, SkypeMorph [129] and CensorSpoofer [201] deliver a reasonable throughput of 344Kbps [129] and 64Kbps [201], respectively. However, due to the difficulty in mimicking the complete behavior of multimedia protocols, these systems are prone to be detected with 100% accuracy through a combination of passive and active attacks [80]. In contrast, Protozoa provides not only strong resistance against traffic analysis, but also higher throughput (around 1.4Mbps).

Raw media tunneling: Systems like FreeWave [82], Facet [105], DeltaShaper [10], and CovertCast [121] modulate covert data in the audio/video input of multimedia applications. Some of these systems can sustain a reasonable throughput. For instance, Facet can reach 471Kbps [105] and CovertCast 168Kbps [121]. However, these systems are vulnerable to statistical traffic analysis techniques [66, 12]: FreeWave, Facet, CovertCast are detected with over 99% accuracy, while DeltaShaper between 85%-95% [66, 12]. Protozoa outperforms these systems both performance and security wise.

4.9.2 Beyond Multimedia Covert Streaming

Protocol mimicking is a general technique for carrying covert data by imitating the behavior of a carrier protocol. However, most solutions [204, 51, 52] suffer from the same limitations as their multimedia protocol siblings and are prone to network attacks [80, 200].

Protocol tunneling has been used in other contexts. SWEET [233], CloudTransport [29], and Castle [71] tunnel covert data through steganographically marked email, cloud storage services, and real-time strategy games, respectively; *meek* [60, 178] leverages domain fronting to hide Tor traffic inside HTTPS connections to allowed hosts. However, unlike Protozoa, some of these systems have not been evaluated against state-of-the-art traffic analysis attacks, and others have already been shown to be vulnerable to detection [200].

There are many other related techniques. Ephemeral proxies like Snowflake [59, 58] (which uses WebRTC connections) redirect traffic through short-lived proxies provided by volunteers; however, unlike Protozoa, the covert traffic is fingerprintable and the presence of secret messages can be detected through traffic analysis. Protocol randomization [46] transforms traffic into random bytes to evade protocol blacklists, but it fails in the presence of protocol whitelisting and is vulnerable to entropy analysis [200]. Refraction networking [63, 23, 24, 53, 64, 81, 90, 212, 213] incorporates special traffic redirection routers inside cooperative ISPs which need to be carefully placed, otherwise a censor can avoid network paths containing such routers [136, 138, 174]. In contrast, Protozoa relies on trusted users located outside the censored region. Packet manipulation strategies [92, 203, 101, 22] aim at invalidating the state of censors' firewalls; Protozoa's covert channels can breach through such firewalls provided that WebRTC traffic is not blocked.

Lastly, some systems provide access to censored content cached in CDNs [79, 235]. Protozoa provides access to any publicly available content accessible to the Protozoa proxies. Mass-Browser [139] leverages cache browsing [79, 235] and volunteer proxies to reach censored content. However, since the connections between clients and proxies are protected with a variant of Obfsproxy [46], they are also affected by the limitations of protocol randomization.

4.10 Conclusions

This paper introduced Protozoa, the first multimedia-based censorship circumvention tool which generates secure covert channels by instrumenting the innards of the WebRTC multimedia framework. Our evaluation shows that Protozoa traffic cannot be distinguished from typical WebRTC flows by state-of-the-art traffic analysis techniques. Further, the results of our evaluation show that Protozoa enables an increase in throughput of up to three orders of magnitude when compared against similar (and less secure) tunneling tools. Currently, Protozoa requires active user support at the proxy's end and demands users to find trusted proxies for exchanging covert content. Devising a scalable solution for finding trusted proxies is an interesting direction for future work.

Acknowledgments

We thank our shepherd, Nick Feamster, and the anonymous reviewers for their comments. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) via the SFRH/BD/136967/2018 grant, and the PTDC/EEI-COM/29271/2017 and UIDB/ 50021/ 2020 projects.

Bibliography

[1] Aiken, J., and Scott-Hayward, S. Investigating adversarial attacks against network intrusion detection systems in SDNs. In *Proceedings of the 5th IEEE Conference on Network Function Virtualization and Software Defined Networks* (Dallas, TX, USA, November 2019), pp. 1–7.

[2] Al-Naami, K., Chandra, S., Mustafa, A., Khan, L., Lin, Z., Hamlen, K., and Thuraisingham, B. Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (Los Angeles, CA, USA, December 2016), pp. 177–188.

[3] Anderson, B., and McGrew, D. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 9th ACM Workshop on Artificial Intelligence and Security* (Vienna, Austria, October 2016), pp. 35–46.

[4] Apruzzese, G., Colajanni, M., Ferretti, L., and Marchetti, M. Addressing adversarial attacks against security systems based on machine learning. In *Proceedings of the 11th International Conference on Cyber Conflict* (Tallinn, Estonia, May 2019), vol. 900, pp. 1–18.

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

[6] Barefoot Networks. Tofino Switch `https://www.barefootnetworks.com/products/brief-tofino/`. Accessed: 2021-03-15.

[7] Barefoot Networks. P4 Studio, `https://www.barefootnetworks.com/products/brief-p4-studio/`. Accessed: 2021-03-15.

[8] Barradas, D. Protozoa code repository. `https://github.com/dmbb/Protozoa`, 2020. Accessed: 2021-03-15.

[9] Barradas, D., and Santos, N. Towards a scalable censorship-resistant overlay network based on webrtc covert channels. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good* (Delft, Netherlands, December 2020), pp. 37–42.

[10] Barradas, D., Santos, N., and Rodrigues, L. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams. *Proceedings on Privacy Enhancing Technologies 2017*, 4 (2017), 5–22.

[11] Barradas, D., Santos, N., and Rodrigues, L. DeltaShaper prototype. `https://dmbb.github.io/DeltaShaper/`, 2017. Accessed: 2021-03-15.

[12] Barradas, D., Santos, N., and Rodrigues, L. Effective detection of multimedia protocol tunneling using machine learning. In *Proceedings of the 27th USENIX Security Symposium* (Baltimore, MD, USA, August 2018), pp. 169–185.

[13] Barradas, D., Santos, N., and Rodrigues, L. Code for USENIX'18 paper: Effective Detection of Multimedia Protocol Tunneling using Machine Learning . `https://github.com/dmbb/MPTAnalysis/`, 2019. Accessed: 2021-03-15.

[14] Barradas, D., Santos, N., Rodrigues, L., and Nunes, V. Poking a hole in the wall: Efficient censorship-resistant Internet communications by parasitizing on WebRTC. In *Proceedings of the 27th ACM Conference on Computer and Communications Security* (Virtual Event, USA, November 2020), pp. 35–48.

[15] Barradas, D., Santos, N., Rodrigues, L., Signorello, S., Ramos, F., and Madeira, A. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *Proceedings of the 28th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2021).

[16] Barradas, D., and Signorello, S. FlowLens code repository. `https://github.com/dmbb/FlowLens`, 2020. Accessed: 2021-03-15.

[17] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and Norrman, K. The secure real-time transport protocol (SRTP). RFC 3711, March 2004.

[18] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Proceedings of the 25th Conference on Neural Information Processing Systems* (Granada, Spain, December 2011), pp. 2546–2554.

[19] Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, GA, USA, June 2013), pp. 115–123.

[20] Biggio, B., Fumera, G., and Roli, F. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics 1*, 1-4 (2010), 27–41.

[21] BigML. Which algorithm does BigML use for Anomaly Detection? `https://support.bigml.com/hc/en-us/articles/206746259`. Accessed: 2021-03-15.

[22] Bock, K., Hughey, G., Qiang, X., and Levin, D. Geneva: Evolving censorship evasion strategies. In *Proceedings of the 26th ACM Conference on Computer and Communications Security* (London, UK, November 2019), pp. 2199–2214.

[23] Bocovich, C., and Goldberg, I. Slitheen: Perfectly imitated decoy routing through traffic replacement. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security* (Vienna, Austria, October 2016), pp. 1702–1714.

[24] Bocovich, C., and Goldberg, I. Secure asymmetry and deployability for decoy routing systems. *Proceedings on Privacy Enhancing Technologies 2018*, 3 (2018), 43–62.

[25] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 87–95.

[26] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 99–110.

[27] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[28] Broadcom. Tomahawk II 6.4Tbps Ethernet Switch, https://www.broadcom.com/news/product-releases/broadcom-first-to-deliver-64-ports-of-100ge-with-tomahawk-ii-ethernet-switch. Accessed: 2021-03-15.

[29] Brubaker, C., Houmansadr, A., and Shmatikov, V. CloudTransport: Using cloud storage for censorship-resistant networking. In *Privacy Enhancing Technologies*, vol. 8555 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 1–20.

[30] Buczak, A. L., and Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials 18*, 2 (2015), 1153–1176.

[31] Calheiros, R., Ramamohanarao, K., Buyya, R., Leckie, C., and Versteeg, S. On the effectiveness of isolation-based anomaly detection in cloud data centers. *Concurrency and Computation: Practice and Experience 29*, 18 (2017).

[32] Calzavara, S., Lucchese, C., and Tolomei, G. Adversarial training of gradient-boosted decision trees. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China, November 2019), pp. 2429–2432.

[33] Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S., and Orlando, S. Treant: training evasion-aware decision trees. *Data Mining and Knowledge Discovery 34* (2020), 1–31.

[34] Canini, M., Fay, D., Miller, D., Moore, A., and Bolla, R. Per flow packet sampling for high-speed network monitoring. In *Proceedings of the 1st IEEE International Conference on Communication Systems and Networks and Workshops* (Chennai, India, December 2009), pp. 1–10.

[35] Cascaval, C., and Daly, D. P4 Architectures, https://p4.org/assets/p4-ws-2017-p4-architectures.pdf. Accessed: 2021-03-15.

[36] Cate Cadell - Reuters. Report says China internet firms censored coronavirus terms, criticism early in outbreak. https://www.reuters.com/article/us-health-coronavirus-china-censorship/report-says-china-internet-firms-censored-coronavirus-terms-criticism-early-in-outbreak-idUSKBN20Q1VS, 2020. Accessed: 2021-03-15.

[37] Chaabane, A., Chen, T., Cunche, M., De Cristofaro, E., Friedman, A., and Kaafar, M. Censorship in the wild: Analyzing Internet filtering in Syria. In *Proceedings of the*

14th ACM Internet Measurement Conference (Vancouver, BC, Canada, November 2014), pp. 285–298.

[38] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys 41*, 3 (2009).

[39] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd Conference on Knowledge Discovery and Data Mining* (San Francisco, CA, USA, August 2016), pp. 785–794.

[40] Chen, Y., Garcia, E., Gupta, M., Rahimi, A., and Cazzanti, L. Similarity-based classification: Concepts and algorithms. *Journal of Machine Learning Research 10*, Mar (2009), 747–776.

[41] Cherubin, G., Chatzikokolakis, K., and Palamidessi, C. F-BLEAU: Fast black-box leakage estimation. In *Proceedings of the 40th IEEE Symposium on Security and Privacy* (San Francisco, CA, USA, May 2019), pp. 835–852.

[42] Chole, S., Fingerhut, A., Ma, S., Sivaraman, A., Vargaftik, S., Berger, A., Mendelson, G., Alizadeh, M., Chuang, S.-T., Keslassy, I., Orda, A., and Edsall, T. drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA, August 2017), pp. 1–14.

[43] Cisco. Cisco Encrypted Traffic Analytics Whitepaper, `https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf`. Accessed: 2021-03-15.

[44] Cormode, G., and Muthukrishnan, S. What's new: Finding significant differences in network data streams. *IEEE/ACM Transactions on Networking 13*, 6 (2005), 1219–1232.

[45] Coskun, B., and Memon, N. Online sketching of network flows for real-time stepping-stone detection. In *Proceedings of the 25th Annual Computer Security Applications Conference* (Honolulu, HI, USA, December 2009), pp. 473–483.

[46] Dingledine, R. Obfsproxy: the next step in the censorship arms race. `https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race`, 2012. Accessed: 2021-03-15.

[47] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium* (San Diego, CA, USA, August 2004), pp. 303–321.

[48] Distributed (Deep) Machine Learning Community. https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions, 2018. Accessed: 2021-03-15.

[49] Dittrich, D., and Kenneally, E. The Menlo report: Ethical principles guiding information and communication technology research. Tech. rep., U.S.Department of Homeland Security, 2012.

[50] Duffield, N., Lund, C., and Thorup, M. Estimating flow distributions from sampled flow statistics. In *Proceedings of the ACM Special Interest Group on Data Communications Conference* (2003), pp. 325–336.

[51] Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 20th ACM Conference on Computer and Communications Security* (Berlin, Germany, November 2013), pp. 61–72.

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

[53] Ellard, D., Jones, C., Manfredi, V., Strayer, W., Thapa, B., Van Welie, M., and Jackson, A. Rebound: Decoy routing on asymmetric routes via error messages. In *Proceedings of the 40th IEEE Conference on Local Computer Networks* (Clearwater Beach, FL, USA, October 2015), pp. 91–99.

[54] ENISA. Encrypted Traffic Analysis: Use Cases & Security Challenges. https://www.enisa.europa.eu/publications/encrypted-traffic-analysis, 2019. Accessed: 2021-03-15.

[55] Ensafi, R., Fifield, D., Winter, P., Feamster, N., Weaver, N., and Paxson, V. Examining how the Great Firewall discovers hidden circumvention servers. In *Proceedings of the 15th ACM Internet Measurement Conference* (Tokyo, Japan, October 2015), pp. 445–458.

[56] Fawcett, T. ROC Graphs: Notes and practical considerations for researchers. *Machine Learning 31*, Jan (2004), 1–38.

[57] FFmpeg. `https://ffmpeg.org`, 2000. Accessed: 2021-03-15.

[58] Fifield, D. *Threat modeling and circumvention of Internet censorship*. PhD thesis, EECS Department, University of California, Berkeley, 2017.

[59] Fifield, D., Hardison, N., Ellithorpe, J., Stark, E., Boneh, D., Dingledine, R., and Porras, P. Evading censorship with browser-based proxies. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies* (Vigo, Spain, July 2012), pp. 239–258.

[60] Fifield, D., Lan, C., Hynes, R., Wegmann, P., and Paxson, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 46–64.

[61] Fifield, D., and Tsai, L. Censors' delay in blocking circumvention proxies. In *Proceedings of the 6th USENIX Workshop on Free and Open Communications on the Internet* (Austin, TX, USA, August 2016).

[62] Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[63] Frolov, S., Douglas, F., Scott, W., McDonald, A., VanderSloot, B., Hynes, R., Kruger, A., Kallitsis, M., Robinson, D. G., Schultze, S., Borisov, N., Halderman, A., and Wustrow, E. An ISP-scale deployment of TapDance. In *Proceedings of the 7th USENIX Workshop on Free and Open Communications on the Internet* (Vancouver, BC, Canada, August 2017).

[64] Frolov, S., Wampler, J., Tan, S. C., Halderman, J. A., Borisov, N., and Wustrow, E. Conjure: Summoning proxies from unused address space. In *Proceedings of the 26th ACM Conference on Computer and Communications Security* (London, UK, November 2019), pp. 2215–2229.

[65] Gebhart, G., Author, A., and Kohno, T. Internet censorship in Thailand: User practices and potential threats. In *Proceedings of the 2nd IEEE European Symposium on Security & Privacy* (Paris, France, April 2017).

[66] Geddes, J., Schuchard, M., and Hopper, N. Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 20th ACM Conference on Computer and Communications Security* (Berlin, Germany, November 2013), pp. 361–372.

[67] Google Developers - WebRTC. Getting started with WebRTC. `https://webrtc.org/getting-started/overview`, 2019. Accessed: 2021-03-15.

[68] Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[69] Gu, J., Wang, J., Yu, Z., and Shen, K. Walls have ears: Traffic-based side-channel attack in video streaming. In *Proceedings of the 37th IEEE Conference on Computer Communications* (Honolulu, HI, USA, April 2018), pp. 1538–1546.

[70] Habeeb, R., Nasaruddin, F., Gani, A., Hashem, I., Ahmed, E., and Imran, M. Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management 45* (2019), 289–307.

[71] Hahn, B., Nithyanand, R., Gill, P., and Johnson, R. Games without frontiers: Investigating video games as a covert channel. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy* (Saarbrucken, Germany, March 2016), pp. 63–77.

[72] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*, 1 (2009), 10–18.

[73] Han, J., Kamber, M., and Pei, J. *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann Publishers Inc., 2011.

[74] Hayes, J., and Danezis, G. k-fingerprinting: A robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Security Symposium* (Austin, TX, USA, August 2016), pp. 1187–1203.

[75] Hemminger, S. Network emulation with NetEm. In *Proceedings of the 6th Linux and open source conference for Australia and New Zealand* (Canberra, Australia, April 2005).

[76] Hernández, J. C. As China Cracks Down on Coronavirus Coverage, Journalists Fight Back. `https://www.nytimes.com/2020/03/14/business/media/coronavirus-china-journalists.html`, 2020. Accessed: 2021-03-15.

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

[78] Hiruncharoenvate, C., Lin, Z., and Gilbert, E. Algorithmically bypassing censorship on Sina Weibo with nondeterministic homophone substitutions. In *Proceedings of the 9th International Conference on Web and Social Media* (Oxford, UK, April 2015).

[79] Holowczak, J., and Houmansadr, A. CacheBrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security* (Denver, CO, USA, October 2015), pp. 70–83.

[80] Houmansadr, A., Brubaker, C., and Shmatikov, V. The parrot is dead: Observing unobservable network communications. In *Proceedings of the 34th IEEE Symposium on Security and Privacy* (San Francisco, CA, USA, May 2013), pp. 65–79.

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

[82] Houmansadr, A., Riedl, T. J., Borisov, N., and Singer, A. C. I want my voice to be heard: IP over voice-over-IP for unobservable censorship circumvention. In *Proceedings of the 20th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2013).

[83] Huang, Q., Jin, X., Lee, P., Li, R., Tang, L., Chen, Y.-C., and Zhang, G. SketchVisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (August 2017), pp. 113–126.

[84] Huang, Q., Lee, P., and Bao, Y. SketchLearn: relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary, August 2018), pp. 576–590.

[85] Ivkin, N., Yu, Z., Braverman, V., and Jin, X. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, FL, USA, December 2019), pp. 285–291.

[86] Jansen, B., Goodwin, T., Gupta, V., Kuipers, F., and Zussman, G. Performance evaluation of WebRTC-based video conferencing. *ACM SIGMETRICS Performance Evaluation Review 45*, 2 (2018), 56–68.

[87] Kang, Q., Xue, L., Morrison, A., Tang, Y., Chen, A., and Luo, X. Programmable in-network security for context-aware BYOD policies. In *Proceedings of the 29th USENIX Security Symposium* (Virtual Event, USA, August 2020), pp. 595–612.

[88] Kantchelian, A., Tygar, J., and Joseph, A. Evasion and hardening of tree ensemble classifiers. In *Proceedings of the 33rd International Conference on Machine Learning,* (New York, NY, USA, June 2016), vol. 48, pp. 2387–2396.

[89] Karev, D., McCubbin, C., and Vaulin, R. Cyber threat hunting through the use of an isolation forest. In *Proceedings of the 18th International Conference on Computer Systems and Technologies* (Ruse, Bulgaria, June 2017), pp. 163–170.

[90] Karlin, J., Ellard, D., Jackson, A., Jones, C., Lauer, G., Mankins, D., and Strayer, T. Decoy routing: Toward unblockable Internet communication. In *Proceedings of the 1st USENIX Workshop on Free and Open Communications on the Internet* (San Francisco, CA, USA, August 2011).

[91] Khattak, S., Elahi, T., Simon, L., Swanson, C., Murdoch, S., and Goldberg, I. SoK: Making sense of censorship resistance systems. *Proceedings on Privacy Enhancing Technologies 2016*, 4 (2016), 37–61.

[92] Khattak, S., Javed, M., Anderson, P. D., and Paxson, V. Towards illuminating a censorship monitor's model to facilitate evasion. In *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet* (Washington, D.C., USA, August 2013).

[93] Khattak, S., Ramay, N., Khan, K., Syed, A., and Khayam, S. A taxonomy of botnet behavior, detection, and defense. *IEEE Communications Surveys & Tutorials 16*, 2 (2013), 898–924.

[94] King, G., Pan, J., and Roberts, M. Reverse-engineering censorship in China: Randomized experimentation and participant observation. *Science 345*, 6199 (2014).

[95] Knockel, J., Crete-Nishihata, M., Ng, J. Q., Senft, A., and Crandall, J. R. Every rose has its thorn: Censorship and surveillance on social video platforms in China. In *Proceedings of the 5th USENIX Workshop on Free and Open Communications on the Internet* (Washington, D.C., USA, August 2015).

[96] Knockel, J., Ruan, L., and Crete-Nishihata, M. An analysis of automatic image filtering on WeChat Moments. In *Proceedings of the 8th USENIX Workshop on Free and Open Communications on the Internet* (Baltimore, MD, USA, August 2018).

[97] Kohls, K., Holz, T., Kolossa, D., and Pöpper, C. SkypeLine: Robust hidden data transmission for VoIP. In *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security* (Xi'an, China, May 2016), pp. 877–888.

[98] Kullback, S., and Leibler, R. A. On information and sufficiency. *The Annals of Mathematical Statistics 22*, 1 (1951), 79–86.

[99] Kučera, J., Popescu, D. A., Wang, H., Moore, A., Kořenek, J., and Antichi, G. Enabling event-triggered data plane monitoring. In *Proceedings of the 6th ACM Symposium on SDN Research* (San Jose, CA, USA, March 2020), pp. 14—26.

[100] Lever, C., Kotzias, P., Balzarotti, D., Caballero, J., and Antonakakis, M. A lustrum of malware network communication: Evolution and insights. In *Proceedings of the 38th IEEE Symposium on Security and Privacy* (San Jose, CA, USA, May 2017), pp. 788–804.

[101] Li, F., Razaghpanah, A., Kakhki, A. M., Niaki, A. A., Choffnes, D., Gill, P., and Mislove, A. lib• erate,(n): A library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the 17th ACM Internet Measurement Conference* (London, UK, November 2017), pp. 128–141.

[102] Li, G., Zhang, M., Liu, C., Kong, X., Chen, A., Gu, G., and Duan, H. Nethcf: Enabling line-rate and adaptive spoofed IP traffic filtering. In *Proceedings of the 27th IEEE International Conference on Network Protocols* (Chicago, IL, USA, October 2019), pp. 1–12.

[103] Li, S., Guo, H., and Hopper, N. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 25th ACM Conference on Computer and Communications Security* (Toronto, ON, Canada, October 2018), pp. 1977–1992.

[104] Li, S., Schliep, M., and Hopper, N. Facet prototype. `https://magicle.github.io/facet/index.html`, 2014. Accessed: 2021-03-15.

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

[106] Li, Y., Miao, R., Kim, C., and Yu, M. FlowRadar: A better NetFlow for data centers. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA, March 2016), pp. 311–324.

[107] Liberatore, M., and Levine, B. N. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (Alexandria, VA, USA, October 2006), pp. 255–263.

[108] Lin, Y., Zhou, Y., Liu, Z., Liu, K., Wang, Y., Xu, M., Bi, J., Liu, Y., and Wu, J. NetView: Towards on-demand network-wide telemetry in the data center. *Computer Networks 180*, 24 (2020).

[109] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining* (Pisa, Italy, December 2008), pp. 413–422.

[110] Liu, L., Wei, W., Chow, K.-H., Loper, M., Gursoy, E., Truex, S., and Wu, Y. Deep neural network ensembles against deception: Ensemble diversity, accuracy and robustness. In *Proceedings of the 16th IEEE International Conference on Mobile Ad Hoc and Sensor Systems* (Monterey, CA, USA, November 2019), pp. 274–282.

[111] Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., and Braverman, V. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communications Conference* (Florianópolis, Brazil, August 2016), pp. 101–114.

[112] Loreto, S., and Romano, S. P. Real-time communications in the web: Issues, achievements, and ongoing standardization efforts. *IEEE Internet Computing 16*, 5 (2012), 68–73.

[113] Luc, D. Pareto optimality. *Pareto Optimality, Game Theory And Equilibria* (2008), 481–515.

[114] Ma, S., and Liu, Y. Nic: Detecting adversarial samples with neural network invariant checking. In *Proceedings of the 26th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2019).

[115] Malboubi, M., Wang, L., Chuah, C.-N., and Sharma, P. Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP). In *Proceedings of the 33rd IEEE Conference on Computer Communications* (Toronto, ON, Canada, April 2014), pp. 934–942.

[116] Markou, M., and Singh, S. Novelty detection: a review—part 2: neural network based approaches. *Signal processing 83*, 12 (2003), 2499–2521.

[117] Massey Jr, F. J. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association 46*, 253 (1951), 68–78.

[118] McGrew, D., and Rescorla, E. Datagram transport layer security (DTLS) extension to establish keys for the secure real-time transport protocol (SRTP). RFC 5764, May 2010.

[119] McKeown, N., Appenzeller, G., and Keslassy, I. Sizing router buffers (redux). *ACM SIGCOMM Computer Communication Review 49*, 5 (2019), 69–74.

[120] McPherson, R., Houmansadr, A., and Shmatikov, V. CovertCast prototype. `https://www.cs.cornell.edu/~shmat/covertcast/`, 2016. Accessed: 2021-03-15.

[121] McPherson, R., Houmansadr, A., and Shmatikov, V. CovertCast: Using live streaming to evade Internet censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 3 (2016), 212–225.

[122] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. N-BaIoT – network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing 17*, 3 (2018), 12–22.

[123] Meier, R., Tsankov, P., Lenders, V., Vanbever, L., and Vechev, M. NetHide: Secure and practical network topology obfuscation. In *Proceedings of the 27th USENIX Security Symposium* (Baltimore, MD, USA, August 2018), pp. 693–709.

[124] Meng, D., and Chen, H. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 24th ACM Conference on Computer and Communications Security* (Dallas, TX, USA, October 2017), pp. 135–147.

[125] Miao, R., Zeng, H., Kim, C., Lee, J., and Yu, M. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA, August 2017), pp. 15–28.

[126] Michalski, R., Carbonell, J., and Mitchell, T. *Machine Learning: An Artificial Intelligence Approach*, vol. 1. Elsevier Science, 2014.

[127] Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. Kitsune: an ensemble of autoencoders for online network intrusion detection. In *Proceedings of the 25th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2018).

[128] Mishra, P., Varadharajan, V., Tupakula, U., and Pilli, E. S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys Tutorials 21*, 1 (2019), 686–728.

[129] Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. SkypeMorph: Protocol obfuscation for Tor bridges. In *Proceedings of the 19th ACM Conference on Computer and Communications Security* (Raleigh, NC, USA, October 2012), pp. 97–108.

[130] Molnár, S., and Perényi, M. On the identification and analysis of Skype traffic. *International Journal of Communication Systems 24*, 1 (2011), 94–117.

[131] Moore, A., Zuev, D., and Crogan, M. Discriminators for use in flow-based classification. Tech. rep., Queen Mary and Westfield College (University of London), Department of Computer Science), 2013.

[132] Morshed, M. B., Dye, M., Ahmed, S. I., and Kumar, N. When the Internet goes down in Bangladesh. In *Proceedings of the 20th ACM Conference on Computer-Supported Cooperative Work and Social Computing* (Portland, Oregon, USA, February 2017).

[133] Moshref, M., Yu, M., Govindan, R., and Vahdat, A. Dream: dynamic resource allocation for software-defined measurement. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 419–430.

[134] Narang, P., Ray, S., Hota, C., and Venkatakrishnan, V. Peershark: detecting peer-to-peer botnets by tracking conversations. In *Proceedings of the IEEE Security and Privacy Workshops* (San Jose, CA, USA, May 2014), pp. 108–115.

[135] Narayana, S., Sivaraman, A., Nathan, V., Goyal, P., Arun, V., Alizadeh, M., Jeyakumar, V., and Kim, C. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA, August 2017), pp. 85–98.

[136] Nasr, M., and Houmansadr, A. Game of Decoys: Optimal decoy routing through game theory. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security* (Vienna, Austria, October 2016), pp. 1727–1738.

[137] Nasr, M., Houmansadr, A., and Mazumdar, A. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 24th ACM Conference on Computer and Communications Security* (Dallas, TX, USA, October 2017), pp. 2053–2069.

[138] Nasr, M., Zolfaghari, H., and Houmansadr, A. The Waterfall of Liberty: Decoy routing circumvention that resists routing attacks. In *Proceedings of the 24th ACM Conference on Computer and Communications Security* (Dallas, Texas, USA, October 2017), pp. 2037–2052.

[139] Nasr, M., Zolfaghari, H., and Houmansadr, A. MassBrowser: Unblocking the censored web for the masses, by the masses. In *Proceedings of the 27th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2020).

[140] Netfilter Framework. `http://www.netfilter.org/`, 1998. Accessed: 2021-03-15.

[141] Netflow. `https://www.ietf.org/rfc/rfc3954.txt`. Accessed: 2021-03-15.

[142] Ngatchou, P., Zarei, A., and El-Sharkawi, A. Pareto multi objective optimization. In *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems* (Arlington, VA, USA, January 2005), pp. 84–91.

[143] Nguyen, T., and Armitage, G. A survey of techniques for Internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials 10*, 4 (2008), 56–76.

[144] Nguyen, T., and Armitage, G. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials 10*, 1-4 (2008), 56–76.

[145] Nguyen, T. T., and Armitage, G. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials 10*, 4 (2008), 56–76.

[146] Niaki, A. A., Cho, S., Weinberg, Z., Hoang, N. P., Razaghpanah, A., Christin, N., and Gill, P. ICLab: A global, longitudinal internet censorship measurement platform. In *Proceedings of the 41st IEEE Symposium on Security and Privacy* (San Francisco, CA, USA, May 2020), pp. 214–230.

[147] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[148] O'Connor, T., Mohamed, R., Miettinen, M., Enck, W., Reaves, B., and Sadeghi, A.-R. Homesnitch: Behavior transparency and control for smart home iot devices. In *Proceedings of the 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Miami, FL, USA, May 2019), pp. 128–138.

[149] Paige Leskin - Business Insider. Here are all the major US tech companies blocked behind China's "Great Firewall". `https://www.businessinsider.com/major-us-tech-companies-blocked-from-operating-in-china-2019-5`, 2019. Accessed: 2021-03-15.

[150] Panchenko, A., and Lanze, F. Website fingerprinting at Internet scale. In *Proceedings of the 23rd Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2016).

[151] Panchenko, A., and Lanze, F. Website fingerprinting at internet scale. In *Proceedings of the 23rd Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2016).

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

[153] Panda, A., Han, S., Jang, K., Walls, M., Ratnasamy, S., and Shenker, S. NetBricks: Taking the V out of NFV. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (Savannah, GA, USA, November 2016), pp. 203–216.

[154] Pearson, K. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 50*, 302 (1900), 157–175.

[155] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[156] Putman, C., Abhishta, and Nieuwenhuis, L. Business model of a botnet. In *Proceedings of the 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing* (Cambridge, UK, March 2018), pp. 441–445.

[157] Quinlan, J. R. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[158] Rahbarinia, B., Perdisci, R., Lanzi, A., and Li, K. PeerRush: Mining for unwanted P2P traffic. *Journal of Information Security and Applications 19*, 3 (2014), 194–208.

[159] Rambert, R., Weinberg, Z., Barradas, D., and Christin, N. Chinese wall or swiss cheese? Keyword filtering in the great firewall of china. In *Proceedings of the 30th The Web Conference* (Ljubljana, Slovenia, 2021).

[160] Ramesh, R., Raman, R. S., Bernhard, M., Ongkowijaya, V., Evdokimov, L., Edmundson, A., Sprecher, S., Ikram, M., and Ensafi, R. Decentralized control: A case study of Russia. In *Proceedings of the 27th Network and Distributed Systems Security Symposium* (San Diego, CA, USA, February 2020).

[161] Rasley, J., Stephens, B., Dixon, C., Rozner, E., Felter, W., Agarwal, K., Carter, J., and Fonseca, R. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review 44*, 4 (2014), 407–418.

[162] Reed, A., and Kranch, M. Identifying HTTPS-protected Netflix videos in real-time. In *Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy* (Scottsdale, AZ, USA, March 2017), pp. 361–368.

[163] RFC 1928 - SOCKS Protocol Version 5. https://tools.ietf.org/html/rfc1928. Accessed: 2021-03-15.

[164] Rimanic, L., Renggli, C., Li, B., and Zhang, C. On convergence of nearest neighbor classifiers over feature transformations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Virtual Event, December 2020), pp. 12521–12532.

[165] Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated website fingerprinting through deep learning. In *Proceedings of the 25th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2018).

[166] rofl0r - GitHub. Proxychains-ng. https://github.com/rofl0r/proxychains-ng, 2011. Accessed: 2021-03-15.

[167] Ronald Deibert, John Palfrey, R. R., and Zittrain, J., Eds. *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace*. MIT Press, 2010.

[168] Roth, K., Kilcher, Y., and Hofmann, T. The odds are odd: A statistical test for detecting adversarial examples. In *Proceedings of the 36th International Conference on Machine Learning* (Long Beach, CA, USA, June 2019).

[169] Roy, R. R. *Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony*. CRC Press, 2018.

[170] Ruan, L., Knockel, J., and Crete-Nishihata, M. Censored Contagion: How information on the Coronavirus is managed on Chinese social media. https://citizenlab.ca/2020/03/censored-contagion-how-information-on-the-coronavirus-is-managed-on-chinese-social-media/, 2020. Accessed: 2021-03-15.

[171] Rubner, Y., Tomasi, C., and Guibas, L. J. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision 40*, 2 (2000), 99–121.

[172] Sandulescu, V., and Chiru, M. Predicting the future relevance of research institutions - the winning solution of the KDD Cup 2016. *arXiv preprint arXiv:1609.02728* (2016).

[173] Schölkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation 13*, 7 (2001), 1443–1471.

[174] Schuchard, M., Geddes, J., Thompson, C., and Hopper, N. Routing around decoys. In *Proceedings of the 19th ACM Conference on Computer and Communications Security* (Raleigh, NC, USA, October 2012), pp. 85–96.

[175] Schuster, R., Shmatikov, V., and Tromer, E. Beauty and the burst: Remote identification of encrypted video streams. In *Proceedings of the 26th USENIX Security Symposium* (Vancouver, BC, Canada, August 2017), pp. 1357–1374.

[176] Schuster, R., Shmatikov, V., and Tromer, E. Beauty and the burst: Remote identification of encrypted video streams. In *Proceedings of the 26th USENIX Security Symposium* (Vancouver, BC, Canada, August 2017), pp. 1357–1374.

[177] Sharma, N., Kaufmann, A., Anderson, T., Krishnamurthy, A., Nelson, J., and Peter, S. Evaluating the power of flexible packet processing for network resource allocation. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation* (Boston, MA, USA, March 2017), pp. 67–82.

[178] Sheffey, S. R., and Aderholdt, F. Improving *meek* with adversarial techniques. In *Proceedings of the 9th USENIX Workshop on Free and Open Communications on the Internet* (Santa Clara, CA, USA, August 2019).

[179] Sivaraman, A., Cheung, A., Budiu, M., Kim, C., Alizadeh, M., Balakrishnan, H., Varghese, G., McKeown, N., and Licking, S. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Florianópolis, Brazil, August 2016), pp. 15–28.

[180] Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. Heavy-hitter detection entirely in the data plane. In *Proceedings of the 3rd ACM Symposium on SDN Research* (Santa Clara, CA, USA, April 2017), pp. 164–176.

[181] Snoek, J., Larochelle, H., and Adams, R. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (Lake Tahoe, NV, USA, December 2012), pp. 2951–2959.

[182] Sonchack, J., Aviv, A., Keller, E., and Smith, J. Turboflow: Information rich flow record generation on commodity switches. In *Proceedings of the 13th EuroSys Conference* (Porto, Portugal, April 2018), pp. 1–16.

[183] Sonchack, J., Michel, O., Aviv, A., Keller, E., and Smith, J. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow. In *Proceedings of the USENIX Annual Technical Conference* (Boston, MA, USA, July 2018), pp. 823–835.

[184] Stein, M. Large sample properties of simulations using latin hypercube sampling. *Technometrics 29*, 2 (1987), 143–151.

[185] Su, Z., Wang, T., Xia, Y., and Hamdi, M. FlowCover: Low-cost flow monitoring scheme in software defined networks. In *Proceedings of the 33rd IEEE Global Communications Conference* (Austin, TX, USA, December 2014), pp. 1956–1961.

[186] Suh, J., Kwon, T., Dixon, C., Felter, W., and Carter, J. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems* (Madrid, Spain, June 2014).

[187] Szigeti, T., and Hattingh, C. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs*. Cisco Press, 2004.

[188] Tarun Kumar Yadav - GitHub. Analyzing web censorship mechanisms in India. `https://github.com/tarun14110/Analyzing-Web-Censorship-Mechanisms-in-India/blob/master/possibly_blocked_websites.txt`, 2018. Accessed: 2021-03-15.

[189] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy* (Saarbrücken, Germany, March 2016), pp. 439–454.

[190] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. In *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, Canada, April 2018).

[191] Trimananda, R., Varmarken, J., Markopoulou, A., and Demsky, B. Packet-level signatures for smart home device events. In *Proceedings of the 27th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2020).

[192] Tschantz, M., Afroz, S., Anonymous, and Paxson, V. SoK: Towards grounding censorship circumvention in empiricism. In *Proceedings of the 37th IEEE Symposium on Security and Privacy* (San Jose, CA, USA, May 2016), pp. 914–933.

[193] v4l2loopback. `https://github.com/umlaeute/v4l2loopback`, 2005. Accessed: 2021-03-15.

[194] Vakali, A., and Pallis, G. Content delivery networks: status and trends. *IEEE Internet Computing 7*, 6 (2003), 68–74.

[195] van Adrichem, N., Doerr, C., and Kuipers, F. OpenNetMon: Network monitoring in OpenFlow software-defined networks. In *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium* (Krakow, Poland, May 2014).

[196] Velan, P., Čermák, M., Čeleda, P., and Drašar, M. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management 25*, 5 (2015), 355–374.

[197] Verizon Business. IP Latency Statistics. `https://enterprise.verizon.com/terms/latency/`, 2015. Accessed: 2021-03-15.

[198] Vines, P., and Kohno, T. 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* (Denver, CO, USA, October 2015), pp. 75–84.

[199] Walt, S. v. d., Colbert, S. C., and Varoquaux, G. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering 13*, 2 (2011), 22–30.

[200] Wang, L., Dyer, K., Akella, A., Ristenpart, T., and Shrimpton, T. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security* (Denver, CO, USA, October 2015), pp. 57–69.

[201] Wang, Q., Gong, X., Nguyen, G. T., Houmansadr, A., and Borisov, N. CensorSpoofer: Asymmetric communication using IP spoofing for censorship-resistant web browsing. In

Proceedings of the 19th ACM Conference on Computer and Communications Security (Raleigh, NC, USA, October 2012), pp. 121–132.

[202] Wang, T., Cai, X., Nithyanand, R., and Johnson, R. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium* (San Diego, CA, USA, August 2014), pp. 143–157.

[203] Wang, Z., Cao, Y., Qian, Z., Song, C., and Krishnamurthy, S. V. Your state is not mine: A closer look at evading stateful Internet censorship. In *Proceedings of the 17th ACM Internet Measurement Conference* (London, UK, November 2017), pp. 114–127.

[204] Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. StegoTorus: A camouflage proxy for the Tor anonymity system. In *Proceedings of the 19th ACM Conference on Computer and Communications Security* (Raleigh, NC, USA, October 2012), pp. 109–120.

[205] Wendzel, S., Zander, S., Fechner, B., and Herdin, C. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys 47*, 3 (2015).

[206] White, A. M., Matthews, A. R., Snow, K. Z., and Monrose, F. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy* (Oakland, CA, USA, May 2011), pp. 3–18.

[207] Winter, P., Hermann, E., and Zeilinger, M. Inductive intrusion detection in flow-based network data using one-class support vector machines. In *Proceedings of the 4th IFIP International Conference on New Technologies, Mobility and Security* (Paris, France, February 2011), pp. 16–20.

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

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

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

[211] Wright, C. V., Coull, S. E., and Monrose, F. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the 16th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2009), pp. 237–250.

[212] Wustrow, E., Swanson, C. M., and Halderman, J. A. TapDance: End-to-middle anticensorship without flow blocking. In *Proceedings of the 23rd USENIX Security Symposium* (San Diego, CA, USA, August 2014), pp. 159–174.

[213] Wustrow, E., Wolchok, S., Goldberg, I., and Halderman, J. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium* (San Francisco, CA, USA, August 2011), pp. 459–474.

[214] Xie, P., Zhang, H., You, W., Zhao, X., Yu, J., and Ma, Y. Adaptive VP8 steganography based on deblocking filtering. In *Proceedings of the 7th ACM Workshop on Information Hiding and Multimedia Security* (Paris, France, July 2019), pp. 25–30.

[215] Xing, J., Kang, Q., and Chen, A. NetWarden: Mitigating network covert channels while preserving performance. In *Proceedings of the 29th USENIX Security Symposium* (Virtual Event, USA, August 2020), pp. 2039–2056.

[216] Xiong, R., and Knockel, J. An efficient method to determine which combination of keywords triggered automatic filtering of a message. In *Proceedings of the 9th USENIX Workshop on Free and Open Communications on the Internet* (Santa Clara, CA, USA, August 2019).

[217] Xu, W., Evans, D., and Qi, Y. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings of the 25th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2018).

[218] Xu, X., Mao, Z. M., and Halderman, J. A. Internet censorship in China: Where does the filtering occur? In *Proceedings of the 12th International Conference on Passive and Active Network Measurement* (Atlanta, GA, USA, March 2011), pp. 133–142.

[219] Yadav, T. K., Sinha, A., Gosain, D., Sharma, P. K., and Chakravarty, S. Where the light gets in: Analyzing web censorship mechanisms in India. In *Proceedings of the 18th ACM Internet Measurement Conference* (Boston, MA, USA, October 2018).

[220] Yan, J., and Kaur, J. Feature selection for website fingerprinting. *Proceedings on Privacy Enhancing Technologies 2018*, 4 (2018), 200–219.

[221] Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., and Uhlig, S. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary, August 2018), pp. 561–575.

[222] Yang, T., Wang, L., Shen, Y., Shahzad, M., Huang, Q., Jiang, X., Tan, K., and Li, X. Empowering sketches with machine learning for network measurements. In *Proceedings of the ACM SIGCOMM Workshop on Network Meets AI & ML* (Budapest, Hungary, August 2018), pp. 15–20.

[223] Yoshimasa Iwase - NTT Communications. A Study of WebRTC Security. `https://webrtc-security.github.io/`, 2015. Accessed: 2021-03-15.

[224] Yu, M. Network telemetry: towards a top-down approach. *ACM SIGCOMM Computer Communication Review 49*, 1 (2019), 11–17.

[225] Yu, X., Xu, H., Yao, D., Wang, H., and Huang, L. CountMax: A lightweight and cooperative sketch measurement for software-defined networks. *IEEE/ACM Transactions on Networking 26*, 6 (2018), 2774–2786.

[226] Zander, S., Armitage, G., and Branch, P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials 9*, 3 (2007), 44–57.

[227] Zane, F., Narlikar, G., and Basu, A. CoolCAMs: Power-efficient TCAMs for forwarding engines. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies* (San Francisco, CA, USA, March 2003), vol. 1, pp. 42–52.

[228] zapret-info. Register of Internet Addresses filtered in Russian Federation. `https://github.com/zapret-info/z-i`, 2020. Accessed: 2021-03-15.

[229] Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., and Wu, J. Poseidon: Mitigating volumetric DDoS attacks with programmable switches. In *Proceedings of the 27th Network and Distributed System Security Symposium* (San Diego, CA, USA, February 2020).

[230] Zhao, G., Xu, H., Chen, S., Huang, L., and Wang, P. Deploying default paths by joint optimization of flow table and group table in SDNs. In *Proceedings of the 25th IEEE International Conference on Network Protocols* (Toronto, ON, Canada, Ocotober 2017), pp. 1–10.

[231] Zhao, M., and Saligrama, V. Anomaly detection with score functions based on nearest neighbor graphs. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems* (Vancouver, B.C., Canada, December 2009), pp. 2250–2258.

[232] Zhao, S., Chandrashekar, M., Lee, Y., and Medhi, D. Real-time network anomaly detection system using machine learning. In *Proceedings of 11th International Conference on the Design of Reliable Communication Networks* (Kansas City, MO, USA, March 2015), pp. 267–270.

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

[234] Zhuo, Z., Zhang, Y., Zhang, Z.-l., Zhang, X., and Zhang, J. Website fingerprinting attack on anonymity networks based on profile hidden markov model. *IEEE Transactions on Information Forensics and Security 13*, 5 (2017), 1081–1095.

[235] Zolfaghari, H., and Houmansadr, A. Practical censorship evasion leveraging content delivery networks. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security* (Vienna, Austria, October 2016), pp. 1715–1726.