Security Evaluation of Network Protocol Obfuscation Proxies

Chongyang Shi
Christ's College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: cs940@cam.ac.uk

June 5, 2018

Declaration

I Chongyang Shi of Christ's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

The majority of this report has been submitted as a conference paper to the 25th ACM *Conference on Computer and Communications Security* (CCS 2018) under the title **CovertMark – testing protocol obfuscation proxies**, which, at the time of submission is undergoing the conference's peer review process and has not received any review feedback. The listed authors of the paper are myself and the project supervisor, Prof Ross Anderson, who provided general guidance on this project's research directions and edited my initial draft of the paper.

The source code for the project is primarily in Python, with several dependent libraries reducing the implementation complexity and supporting certain features, which have been attributed in the report text and bibliography.

Total word count: 12,635

Signed:

Date:

Acknowledgement

Undertaking a postgraduate degree at Cambridge was in no way a trivial task. At the conclusion of this thrilling (and sometimes terrifying) adventure, I would like to express my gratitude and appreciation to many who have rendered their kind assistance:

First and foremost, I would like to thank my parents for their support and counsel during this particularly challenging year of academic work, without whom I would have conceded defeat a long time ago.

Within the Department of Computer Science and Technology, I would like to express my gratitude to Prof Ross Anderson, who recommended and supervised this project with vital guidances; and to Dr Steven Murdoch (now of UCL), who provided some recommendations during this project's formative stages, and verified some security observations from the results; and to the anonymous student volunteers for participating in the data collection session, making at-scale evaluations possible.

I would like to also express my appreciation to many of my friends, with whom I crossed path at Cambridge or York, for their support powering me through this challenging process.

Abstract

Internet censorship by nation states now involves distributed deep-packet inspection (DPI) systems using sophisticated classification techniques. The arms race between state censors and the developers of censorship-circumvention tools has made the detection of obfuscated traffic into a battleground. The growing use of protocols such as Tor has led censors to invest in passive analysis capabilities to detect such traffic. In response, toolsmiths have designed Tor Pluggable Transports and other DPI-resistant proxies that obfuscate a broad range of identifiable features to make passive classification harder, collectively known as network protocol obfuscation proxies (POPs).

The security offered by such proxies against classification, or their *covertness*, can be difficult to measure and compare. We designed CovertMark as an evaluation testbed for benchmarking the covertness of such proxies. We implemented a number of detection strategies including a new, efficient detection technique for meek; an SVM-based machine-learning classifier effective on different types of Pluggable Transport protocols as well as on the Shadowsocks proxy; and fast entropy distribution and estimation attacks on obfs4proxy. Our test strategies were evaluated on real traffic traces generated by volunteers, which will be released with this report. We find that the existing proxies are all easy to break with such tools; that our SVM classifier is a good all-round test for screening proxies; and that there may be interesting conflicts between anonymity and censorship resistance.

Total word count: 12,635

This word count has been reported by `texcount`. It excludes the declaration, acknowledgement, bibliography, photographs, diagrams and data listings, but includes the abstract and narrative text in tables, footnotes, and appendices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tor is a popular tool for anonymised and censorship-resistant network communications. While it is trivial for a network node to detect and block non-obfuscated Tor traffic [2, Tb. 6] through deep-packet inspection (DPI), the Tor Project provides a set of *Pluggable Transport* (PT) tools for clients to obfuscate their connections to a Tor bridge node, evading detection by censors. An arms race of obfuscation and detection between PT developers and state internet censors has been going on for some years [3].

Tor Pluggable Transports [4] are Layer-7 proxies tunnelling encrypted TCP packets between clients and PT-protected Tor bridges. A throttled distribution mechanism is publicly available to limit a censor's ability to query PT bridges en masse and block their IP addresses. Some non-PT proxy protocols also offer DPI-resistance, such as Shadowsocks [5], which provides a censorship-resistant proxy via thousands of privately-hosted servers located outside censoring states. For the rest of this report, we will collectively refer to Tor PTs and other DPI-resistant network proxies as *protocol obfuscation proxies* or POPs, and their underlying designs as *obfuscation protocols*.

PT protocols commonly tunnel through or mimic another protocol considered as "legitimate" by censors to evade classification by passive traffic analysis [6]. Most PT protocol designs also include randomisation and mitigation

against active attacks [7, Sec. 4.1] such as those used by the Great Firewall of China (GFW) to block Tor bridges and entry nodes. While widely considered as the most sophisticated internet censorship system to date, the GFW still primarily relies on active attack [8] and scanning through publicly-distributed server lists [9] to find and block servers acting as POPs.

While state censors often do not conduct passive analysis beyond the initial packets of a TCP session due to the heavy computational costs, there are signs that they may be increasingly motivated to expand their scope of analysis. Notably, the ineffectiveness of active attacks against meek bridges prompted recommendations by military-affiliated researchers [10] to the Chinese government, including a proposal to expand the GFW's passive analysis capabilities. Attempts have also been made to train statistical classifiers to detect Shadowsocks [11], another protocol obfuscation proxy on which the GFW has had limited success. We therefore sought to build a systematic evaluation scheme for analysing the covertness of protocol obfuscation proxies that will stand the test of time. This was inspired in part by the StirMark suite of tests for steganographic and copyright-marking systems, which has been in use for almost 20 years. Our contributions in this report include:

- We propose a new threat model for DPI-resistance evaluations of obfuscation protocols, with performance targets to simulate the goals of a current state censor more realistically, including a reduced emphasis on true positives and more pragmatic views on false positives.
- We designed the open-source CovertMark DPI framework for automated covertness analysis of network protocol obfuscation proxies (POPs).
- Along with this report, we release a range of positive and negative traces containing TCP flows produced from realistic internet usage by human volunteers under controlled conditions, as a contribution to future passive analysis research.
- A new detection technique based on TCP payload length distribution was developed. It is shown to be highly effective in detecting meek with minimal computational cost and state-keeping requirements, exploiting meek's

reliance on TLS semantics.

- A new machine-learning (ML) model based on stochastic gradient descent (SGD) was developed with refined feature sets to enhance resilience to overfitting and improve portability between network conditions.
- Detection techniques based on entropy distribution and byte uniformity by Wang et al. [12, Sec. 5] were also refined to reduce computational cost by refining inspection criteria based on each protocol tested. We further implemented a faster alternative through entropy estimation.
- We used our CovertMark framework to evaluate both our new detection techniques and existing ones on three representative protocols: meek, obfs4proxy (obfs4), and Shadowsocks.

With some anonymity-enhancing features of obfuscated protocols found to have hampered the covertness of obfuscation, our findings raise a question of whether a fundamental conflict exists between the goals of anonymity and censorship resistance when designing traffic obfuscation protocols.

The rest of this report is structured as follows. We first propose a new threat model based on realistic state censorship, and discuss the related limitations of the prior work in this area in chapter 2. We then describe how real browsing traffic was collected under controlled conditions (chapter 3). Design and evaluations detection techniques will be discussed individually: the new payload length distribution attack on meek (chapter 4), the refined SVM ML classification model with SGD training (chapter 5), and our refinements to entropy-based attacks by Wang et al. [12, Sec. 5] (chapter 6). Finally, we provide an overview on the CovertMark DPI framework developed in chapter 7. We conclude in chapter 8 with a summary of past efforts in obfuscation detection and other related fields, in contrast with our contributions, leading to some concluding remarks (chapter 9).

Chapter 2

Background and Threat Model

2.1 Censorship and protocol obfuscation

The expansion of Internet access has caused much anxiety to governments, and particularly to hybrid or authoritarian regimes [13]. Unlike traditional receive-only media such as television and radio – on which states can enforce editorial control [14, p. 152] – the Internet is a mass bidirectional medium, which poses a new challenge to censors. Most states have nevertheless allowed wide Internet access to maintain the competitiveness of their economies [15], while imposing varying levels of content filtering over local network traffic.

As material deemed undesirable by state censors and hosted within their jurisdictions can be easily taken down [16], Internet users hoping to share and access such content are forced to host them outside censoring jurisdictions. With traffic crossing the state boundaries subject to observation and automated filtering [17], users have relied on encrypted tunnelling protocols carrying traffic through proxy servers to evade the censors [18, 19].

Increasingly powerful and stateful firewall systems have been deployed by state censors in recent years to detect and block tunnelling. Commercial VPN protocols using UDP have been selectively and periodically blocked [20]; TCP-capable VPN protocols such as OpenVPN have been effectively

fingerprinted through their TLS/SSL handshakes and blocked [21]; and along with SSH tunnels throttled or blocked altogether [19]. Many censorship systems use deep-packet inspection (DPI) technologies, and while DPI systems cannot decipher encrypted content, they can inspect outgoing and incoming packets for any telltale features of tunnelling protocols. The censor may then block the IP addresses of out-of-jurisdiction servers or even actively and selectively tear down tunnelling connections.

The next step in the arms race has been for censorship-circumvention tools to use network protocol obfuscation proxies (POPs). To facilitate connections to the Tor anonymity network [22] from states that use DPI filtering to block Tor traffic, a class of proxy protocols known as Pluggable Transports (PTs) [4] are shipped with the Tor Browser Bundle. Apart from tunnelling traffic between Tor clients and PT bridges, and being distributed in a manner resistant to en-masse blocking, PTs are equivalent in functionality to DPI-resistant, non-anonymous encrypted proxies like Shadowsocks [5] [1].

To avoid excessive overhead, POPs typically transform censored traffic to look like protocols considered legitimate traffic by censors, or unknown pseudo-random bytes. Examples of the former approach include the mimicry applied by format-transporting encryption (FTE) [23] and meek's TLS domain fronting [24]; while examples of the latter include ScrambleSuit [7] and its derivation *obfs4* [25]. Shadowsocks being a non-anonymous proxy protocol also applies pseudo-random encryption.

So far, most obfuscated payloads have been found to be distinguishable from payloads of the legitimate protocols they mimic, to which we will refer as *cover traffic*. An overview of detection techniques and common limitations follows in the next section. From here on, we will consider an obfuscation technique to be secure with decent covertness if it is difficult to distinguish from cover traffic, and broken if distinguishing it is easy.

[1] Shadowsocks also supports relaying UDP traffic over TCP, which is not required for tunnelling Tor traffic, but provides censorship-circumvention support for UDP-based applications.

2.2 Obfuscation detection and limitations

The most comprehensive study of detecting protocol obfuscation to date was published by Wang et al. [12] in 2015; this was also the first study in the field to consider the effects of false positives carefully. The study was centred on deployed Tor PTs. Entropy and entropy distribution classification techniques were used to detect obfs4 and format-transporting encryption (FTE), while TCP flow features from initial packets were used to train a CART-based decision tree classifier model to primarily detect meek. Both entropy-based techniques and the decision-tree model were deemed reasonably successful in detecting obfuscation with low false positive rates.

Wang et al. also raised doubts about the effectiveness of protocol semantics-based attacks developed previously by Houmansadr et al. [26] due to the non-compliant nature of the web. Other machine-learning models have been trained on TCP flows to achieve reasonable effectiveness at detecting specific obfuscation protocols, including an SVM/C4.5 model on initial session packets by Soleimani et al. [27] for detecting ScrambleSuit/obfs4 and a Random Forest model with broader scope by Deng et al. [11] for detecting Shadowsocks.

However, we believe that these existing detection techniques are not sufficient for systematic covertness analysis of a protocol obfuscation proxy operating under realistic state-censorship conditions, for the following reasons:

- **Detection techniques tailored for individual protocols**: In order to achieve an acceptable false positive rate for each protocol, Wang et al. refined entropy-based techniques and the decision-tree model with features from individual PTs' protocol specifications. They exploited the minimum handshake payload length to spot obfs4, and made meek's client-to-server packet frequencies a feature for their CART classifier. While they achieved effective classification of the target PT protocols, it is harder to replicate a collection of specialist tools on a DPI box filtering real world traffic with a heavy imbalance between legitimate and obfuscated traffic, and varying

degrees of protocol non-compliance.

- **High computational cost per packet**: Entropy distribution tests on uniform distributions are computationally expensive due to the overhead involved in calculating entropy segment-by-segment. Testing all possible combinations of input features to an ML classifier for each individual obfuscation protocol will also generate a significant amount of setup cost.

- **Limitations in fingerprinting protocol handshakes**: Past efforts [12, 27, 28] in training machine-learning models to classify protocol obfuscation proxies have generally relied on the first few packets in each independent TCP session. For traditional DPI boxes, this is the default design: to inspect TCP sessions as they start up. To improve network efficiency, nearly all deployed PT protocols and Shadowsocks reuse the same TCP session (and the TLS session if applicable) between each client and the server. Figure 2.1 shows the number of public PT bridge users accessing the Tor network from Russia immediately following the April 2018 blockage of Telegram [29]. With fewer than 12,500 users between public obfsproxy and meek bridges, the Russian censors face a hugely imbalanced input dataset under the backdrop of more than 80 million daily internet users [30], which quantitatively amplifies the effects of even the lowest false positive rates. Detection techniques effective on arbitrary segments of live TCP sessions would see more positive samples and alleviate the input imbalance issue. Inspecting arbitrary segments also improves the detection of stateless proxies with no distinctive flow features, such as Shadowsocks.

- **Training on synthetic traffic damages portability**: While the negative traces captured by Wang et al. were real internet traffic from a university campus, all their Tor PT traffic (the positive samples) was synthetic [12, Sec. 3.1]. The automated collection process involved scripted visits to the front pages of the top 5,000 Alexa-ranked websites, presumably to generate large amounts of handshakes (PT clients were restarted after each page visit to force new TCP sessions [31]). While the negative impact of deviating from typical user behaviour may be minimal for fingerprinting handshakes, the limited information from handshakes may

have contributed to the significantly worsened performance when testing decision-tree models, that were trained on synthetic traffic, under other network conditions [12, Tb. 8] [6, p. 49]. For passive analysis techniques to be effective on arbitrary segments of TCP sessions, training needs to be on positive traffic traces resembling realistic user behaviour, such as irregular timings between visiting new pages, and streaming media content generating high volumes of traffic.

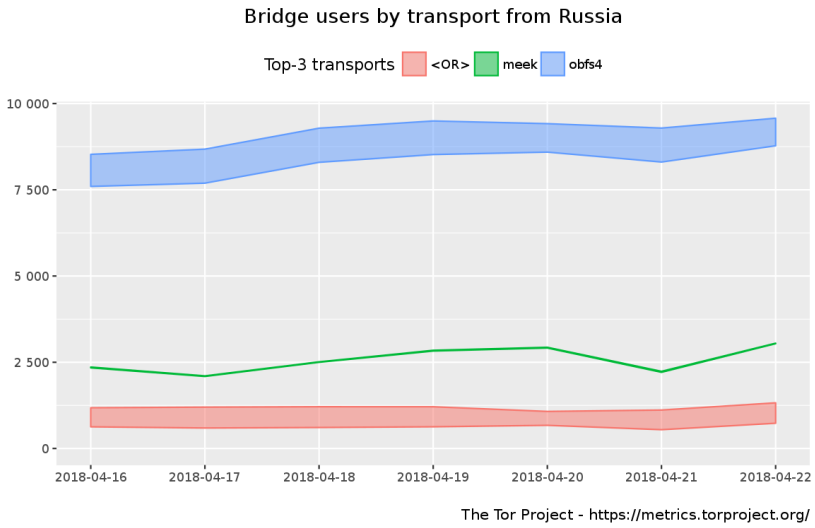

Figure 2.1: Daily users of Tor's public PT bridges in Russia during the week immediately after the government's block of Telegram and popular hosting services for VPN servers. [1]

2.3 Threat model and limitations

Threat model: Our threat model assumes a more aggressive state censor than considered by Wang et al. [12, Sec. 2], or by Winter et al. [7, 3.1] in designing ScrambleSuit. Given a very active domestic internet economy under tight regulation, the state censor can afford a reasonable level of false positives in deploying detection techniques against foreign IP addresses, especially during sensitive national events [32]. Another significant difference between our model and those of prior studies is that the true positive rate

does *not* need to be close to a perfect 100%, since the ultimate purpose of a state censor is to block the IP addresses of the servers. This means that the censorship system can also afford to miss some obfuscated packets or flows, as long as their servers will eventually be flagged and blocked.

We also assume that the censor is aware of the popular circumvention tools in use, such as deployed Tor PTs, but hopes to develop generalised detection techniques that can detect more than one protocol obfuscation proxy for reasons of efficiency. The censor will also want a rapid response to new circumvention tools, so we follow Winter et al. [7, Sec. 3.1] in assuming that manual offline analysis will take place. The censor's researcher staff will typically train statistical classifiers, then deploy new classifiers in the existing online framework.

Metrics: Based on these practical considerations for a state censor, in evaluating CovertMark detection techniques and automatically selecting their best parameters, we opted for a hybrid scoring scheme to measure the effectiveness of each technique with each parameter set (or *configuration*). Four metrics are used across all techniques and calculated individually for each parameter set: the true positive rate (TPR), the false positive rate (FPR), the false block rate of negative IPs (FBR), and the execution time of the technique. Specific implementations of techniques (known as *strategies*) can apply parameter-specific penalisation on the shared metrics to reflect the parameter's effect on implementation cost. We therefore opted to consider metrics individually to compare performance between difference detection techniques rather than using a uniform measure such as the PR/ROC AUC [33] or the F_1 score.

We introduced the false block rate of negative IPs (FBR) as a distinct metric to represent the per-foreign-IP frequency of false positives, regardless of how many false positives were triggered with each foreign IP address. If a detection technique triggers false positives frequently, but only at specific rare segments of a protocol, we expect a significant proportion of unique foreign IP addresses to be incorrectly flagged by the technique thanks to packets associated with these segments. This will cause significant practical difficul-

ties to the censor's minimisation of false positives, despite the overall low per-packet FPR.

Limitations: Because of the need for generalisation, CovertMark does not measure active probing detection techniques, which have been substantially studied elsewhere [8, 9, 34]. Active probing mitigation should nevertheless remain a design consideration of any protocol obfuscation proxy, and we leave its evaluation as future work. Similarly, detection techniques based on verifying protocol semantics are not considered here, due to the non-compliant nature of much web traffic [12, Sec. 4] and consequentially the high false positives involved.

While most of our detection techniques can support UDP with minimal modification, no currently deployed Tor PT tunnels through UDP, and whitelisting could also be applied to UDP traffic more easily by censors without causing serious collateral damage, as the GFW has done occasionally on UDP-based VPN protocols [35]. With limited resources, we suggest the development of generalised detection techniques against UDP obfuscation protocols such as Skypemorph [36] as an area of future work.

Chapter 3

Traffic Collection

Unlike state censors, researchers cannot easily arrange legal research access to captured network traffic for traffic analysis. For negative traffic traces, Wang et al. [12, Sec. 3] were able to obtain ethical review approval to evaluate their detection techniques on campus network traces. This is not possible within our jurisdiction. Because entropy-based attacks and the ML model require access to encrypted TCP payloads, it is also not possible to use research traces from CAIDA [37] or LBL [38], as payloads were stripped away as part of the anonymisation process.

For positive traffic traces obfuscated by different POPs, as discussed in section 2.2, we also could not rely on synthetic traffic generated through browser automation, as our threat model considers censors deploying broad-scope passive analysis not limited to the initial packets of a TCP session. Therefore our only option was to invite volunteers to create browsing traffic under controlled conditions.

We obtained approval from our research ethics committee to conduct a traffic collection session under strict limitations. Five medium-spec Ubuntu 16.04 machines with wired connections were set up with *tcpdump* and Firefox 48.0 in a software laboratory. The five student volunteers invited to browse the internet and generate negative traces were instructed to avoid entering any

truthful personal information into the computers used for packet capture, and avoid logging into any internet accounts they held. During the hour-long session, they browsed a wide range of websites commonly used by modern web users, while streaming varying music or high-bandwidth video content, simulating the typical behaviour of internet users. Traffic generated in both directions was captured as pcap files.

To further protect the privacy of volunteers, the pcap files were filtered to strip away packets with unencrypted payloads, such as HTTP packets. The IP and MAC addresses of packet sources and destinations were anonymised with Crypto-PAn [39]. It was explained to the volunteers before the session that while payloads will be encrypted and unreadable, some Layer-4 and Layer-7 meta-data such as TLS server name identification (SNI) hostname will remain readable in the released traffic. The nature of the websites visited could also sometimes be inferred from encrypted payloads through website fingerprinting [40]. The ethics committee allowed us to retain these metadata on the grounds that they will not endanger the privacy of volunteers due to the strict privacy-related instructions.

With some assistance from the author, the volunteers generated a total of 1,566,737 non-obfuscated packets, which were merged as the *multi-client* trace. Unfortunately, due to technical limitations, these packets were captured on the operating system rather than on the wire, before hardware large segment offloading (LSO) was applied to break down longer-than-MTU TCP payloads into smaller packets. While this had negligible impact on our detection techniques, the author additionally recorded a single-machine wireless trace with LSO applied (*lso*), to cross-validate the detection techniques. It is also necessary for a DPI analysis framework to support user-supplied traces without LSO.

By performing browsing activities similar to the volunteers', we recorded various obfuscated traces for meek, obfs4, and Shadowsocks as positive samples for evaluating our detection techniques. All traces other than *multi-client* were captured on an Apple laptop on macOS 10.13 through various wireless connections, with deployed PTs shipped with Tor Browser Bundle 7.5, and

Shadowsocks on version 2.9.1 with the AES-256-GCM cipher. For each obfuscation protocol, we captured three hour-long traces over wireless connections of moderate quality. Between the three traces, two that were captured over a connection with better signal-to-noise ratios (SNRs) were merged into a trace with timing adjustment, to the effect of having two concurrent users in the trace. The other trace captured over a connection with worse SNR was set aside for separate validation of detection techniques. All captured traffic traces involved in this study are summarised in Table 3.1.

Name	Protocol	Clients	# Packets	LSO Applied	Interface	Quality
ss1	Shadowsocks [41]	2	674,458	Yes	Wireless	Medium
ss2	Shadowsocks	1	286,887	Yes	Wireless	Low
meek1	meek [24]	2	756,548	Yes	Wireless	Medium
meek2	meek	1	602,134	Yes	Wireless	Low
obfs4-1	obfs4proxy [25]	2	619,754	Yes	Wireless	Medium
obfs4-2	obfs4proxy	1	373,504	Yes	Wireless	Low
multi-client	Regular Traffic	5	1,566,737	No	Wired	High
lso	Regular Traffic	1	200,405	Yes	Wireless	Medium

Table 3.1: Positive (obfuscated) and negative traces collected for the study, later anonymised and released.

Chapter 4

The meek Payload Length Attack

The first detection technique we implemented in the CovertMark framework was accidentally discovered while examining the distribution of client-to-server packets in meek-obfuscated traces. This attack leverages meek's genuine TLS client-server semantics to detect a distinctive cluster of TCP payload lengths, created by the client's need to regularly send valid TLS packets to poll the server and retrieve downstream data.

This attack may also be applicable to other POPs with distinctive payload distributions, which can be automatically detected with mean shift clustering [42] and verified on positive and negative traces. To further improve accuracy and reduce computational cost, the detection technique supports optional filtering based on TLS, or any other record-evident protocol leveraged by a known POP.

4.1 Technique design

Since meek portrays Tor PT traffic as semantically-valid TLS sessions [24, Sec. 5.1], packets between meek clients and servers through reflectors have

valid TLS records. As commented in meek's source code, it is not possible for the server (PT bridge) to send the client any data without the client polling for it through the reflector. In meek's implementation, the meek client polls the server regularly [43], up to several times per second. Each time such a TLS packet is sent by the client, the packet's encrypted TLS application data will carry a fixed protocol header and any upstream bytes that happen to be in the client's buffer.

However, under regular web browsing conditions, the user will likely stay on a page for some time before loading the next one. This results in periods of time during which nothing will be present in the client buffer waiting to be sent with an upstream packet, causing the outbound TCP payload length to be of a fixed value, with minor runtime variations. This in turn causes a substantial proportion of meek client-to-server packets to carry valid TLS records that contain the fixed-length protocol header only.

In the accompanying publication, the maintainers of meek concluded that while there were minor distinctions between raw distribution of TCP payload lengths of meek packets and negative traffic, these distinctions were not sufficiently distinguishable for efficient classification with low false positives [24, Sec. 8.1]. However, if TCP packets with empty payloads or not carrying valid TLS records are filtered out (a stateless process easily applied by firewalls), the TCP payload length distribution of meek's client-to-server packets paints a very different picture, shown in Figure 4.1.

To determine what payload lengths are effectively distinguishable, we applied unsupervised mean shift clustering to the meek-obfuscated positive traces, using the algorithm described in [42]. To account for small runtime variations of the empty request upstream payload size, we varied the bandwidth used in clustering between 1 and 10. We also consider both the top cluster only (usually 64-73 bytes in length), and the top two clusters (64-73 bytes as well as ≈38 bytes in length). Not all meek client-to-server packets fall into the clusters, with those outside the clusters carrying meaningful upstream payloads. Therefore the per-packet true positive rate is limited to the number of empty polling packets within the cluster. By testing the same process

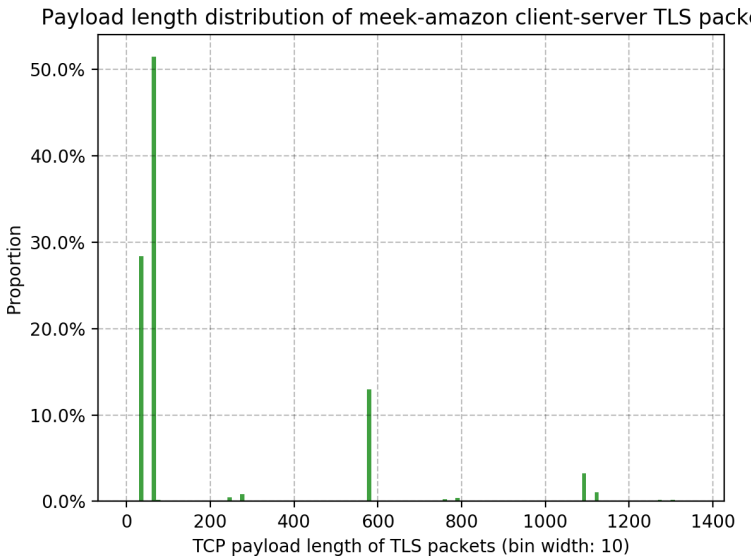

Figure 4.1: The binned TCP payload length distribution of meek's client-to-server packets carrying valid TLS records, from the *meek1* trace, with a bin width of 10.

on *meek1*, *meek2*, and smaller samples from meek runtimes on other platforms, we are confident that the top two clusters could always be located at approximately the same TCP payload-length ranges in meek traces.

With client-to-server TLS payload length clusters located – signaling meek TLS upstream packets – negative traces were then consulted to determine the proportions of client-to-server TLS packets with payload lengths matching each cluster, which constitute false positives when this detection technique is applied to all TCP upstream packets carrying valid TLS records.

4.2 Results

We performed mean shift clustering on *meek1* and *meek2* separately, and then validated the distinctness of the mean shift clusters on *multi-client* and *lso* negative traces to estimate FPR and FBR.

The execution time on positive traces remained consistently proportional to the number of packets in the trace, as expected. All other shared performance metrics with the top mean shift cluster only are shown in Table 4.1.

Trace	Metric	BW=1	BW=2	BW=3	BW=5	BW=10
meek1	TPR	40.31	51.52	51.52	51.58	51.58
multi-client	FPR	0.000	0.001	0.001	0.004	0.011
	FBR	0.000	0.438	0.438	1.205	2.520
lso	FPR	0.061	0.125	0.125	0.164	0.247
	FBR	7.143	8.791	8.791	10.99	14.29
meek2	TPR	42.96	48.10	51.78	51.78	51.78
multi-client	FPR	0.0003	0.0003	0.0017	0.0017	0.0017
	FBR	0.110	0.110	0.548	0.548	0.548
lso	FPR	0.063	0.064	0.145	0.145	0.145
	FBR	7.692	8.242	10.44	10.44	10.44

Table 4.1: Detection performance (%) of the payload length distribution attack using just the top mean shift clusters from *meek1* and *meek2*, on a varying mean shift bandwidth (BW), with validations on both negative traces (*multi-client* and *lso*).

With the maximum TPR effectively equal to the proportion of client-to-server TLS packets with empty wrapped payloads, the censor will inevitably miss some packets outbound to meek reflectors. In the case of using the top cluster only, missed packets constitute approximately half of all outbound packets. However, as the client-to-server polling occurs at regular intervals up to several times a second [43], the censor can still detect connections to meek reflector servers quickly and with a high level of certainty. While the FPR is already negligible (0.0009% of 761,215 outbound packets in *multi-client* when BW$\leq$3), it can be further reduced in practice by network-wide counting of how many times each foreign IP address receives TLS packets with matching payload lengths, and blocking on reaching a threshold. As firewall states are only created when TLS packets of the rare sizes are sent, and more than a few users within the censored network will be using the bridge concurrently, we expect counting states to last no more than a second in this detection technique.

The FBR, however, was affected by the number of unique remote IP addresses in the input sample, which was substantially lower in the smaller *lso*

negative trace (182 IPs, 7.14-14.3% flagged) than the concurrent *multi-client* trace (913 IPs, 0-2.52% flagged). While a non-negligible (>0.5%) FBR will clearly be unacceptable for a detection technique under realistic scenarios, with vastly greater quantities of unique foreign IP addresses seen in outbound packets by a state censor, we expect the FBR to remain close to zero for this detection technique during real-world operations using the top cluster only.

Also shown in Figure 4.1 was a second peak at 38 bytes of TCP payload length among meek's outbound packets, which was consistently captured in the second cluster. With the inclusion of the second cluster, the positive payload lengths cover 68.7–80.0% of outbound TLS packets in *meek1* and *meek2*, raising the TPR. However, the frequency of this payload length was also non-negligible in non-obfuscated outbound TLS packets (1760 or 0.23%), causing FBR to rise substantially despite maintaining a reasonable FPR, as shown in Table 4.2.

Due to the sporadic nature of the 38 byte payload length, and these payloads being encrypted TLS application data, we were unable to pinpoint the exact nature of these client-to-server packets after the fact due to the browser's almost exclusive use of cipher suites with forward secrecy. Based on IP information, most of these packets' destinations were CDN providers, similar to those used as meek reflectors. While the second cluster is likely unusable for detecting meek, we retained it in the implemented detection strategy, in case it becomes useful for detecting another POP.

Trace	Metric	BW=1	BW=2	BW=3	BW=5	BW=10
meek1	TPR	68.69	79.90	79.90	80.04	80.05
multi-client	FPR	0.231	0.232	0.232	0.288	0.534
	FBR	28.15	28.15	28.15	28.37	48.19
lso	FPR	0.279	0.343	0.343	0.478	0.750
	FBR	46.70	46.70	46.70	54.40	67.58

Table 4.2: Detection performance (%) of the payload length distribution attack using the top *two* mean shift clusters from *meek1*.

4.3 Discussion

By exploiting the client-driven nature of TLS sessions, we were able to refine generic fingerprinting of TCP payload lengths to identify meek, a protocol obfuscation proxy fronting itself as genuine TLS through a headless browser, and thus needing regular polling to fetch downstream data. While we have not observed this feature on any other PT or DPI-resistant proxies in use, the general technique may well be applicable to any POP leveraging a client-driven protocol through fronting, and so it definitely belongs in any covertness test suite. The detection strategy implemented in CovertMark allows the user to choose at run time whether to observe only TLS packets or all TCP packets. The same choice can be offered with minimal implementation effort for other fronted Layer-7 protocols with an easily-identifiable per-packet record.

Our findings leading to the design of this detection technique were consistent with suspected vulnerabilities raised in the Tor Project's evaluation of meek [44], in particular those related to statistical attacks and the "client driven poll timing". While this detection technique does not leverage the close timing proximity of TLS polling packets, the statistical prominence of their payload lengths among all packets is relied upon.

Ahead of this report and its associated paper's submission, we informed the primary maintainer of meek about our findings. Integration of HTTP/2 Server Push [45, Sec. 8.2] into the fronting protocol was suggested as a fix to eliminate TLS polling. So long as no distinctive features arise from push responses wrapped in TLS payloads, this should mitigate our detection technique. Alternatively, random padding can be applied to upstream packets to mask the polling, albeit at the cost of some bandwidth overhead.

We also emphasise these findings as a cautionary tale for designers of POPs implementing tunnelling over legitimate protocols, highlighting that merely reproducing semantically-valid flows of the target protocol would not be sufficient to achieve effective obfuscation. A POP's lack of typical statistical

features exhibited by the target protocol can allow highly effective detection techniques, the mitigations for which can be difficult to produce.

Chapter 5

The SGD-based ML Detection Model

With traditional passive DPI systems inspecting the initial packets of a TCP session for protocol obfuscation proxies, deployed PTs and popular DPI-resistant non-anonymous proxies either minimise and obfuscate the negotiations required to maintain secure connections, or accurately simulate those conducted by a "legitmate" protocol such as TLS. The former class of proxies include obfs4 transmitting randomly-padded and fully-encrypted handshake packets between clients and servers, and Shadowsocks sacrificing convenience with pre-shared symmetric keys for fully stateless connections. The latter class includes meek, which deploys a headless browser to front TLS sessions.

While the ML-based attack by Wang et al. [12, Sec. 6] targeted all deployed PTs, many timing-based and packet-header features were designed with meek in mind. This was presumably due to meek being more effective at resisting prior entropy-based attacks (which we will confirm later). While multiple ML models can indeed be deployed on DPI equipment at network edges for simultaneous detection of various known POPs, the selection of these features into different classifiers may not be a stable process. We suspect this was why the only reported performance of feature combinations was of entropy-based together with header-based features [12, Fig. 7].

For ease of implementation, we sought to design a generalised ML feature set applicable to both known and unknown obfuscation protocols, while maintaining performance and portability between network conditions to levels comparable with past efforts [12, 27, 28]. We also sought the ability to classify packets drawn from any part of the obfuscated flow, rather than only the initial packets, to avoid a hugely imbalanced input dataset as discussed in section 2.2 above, as well as to reduce the workload of users creating positive samples for evaluation by CovertMark.

5.1 Model selection and input choice

To select a suitable ML model resistant to overfitting under a generalised feature set, with an original feature set (described in the next section), we performed preliminary evaluations of trained classifiers on traces of the same obfuscation protocol captured under different network conditions. While CART decision trees yielded good initial performance, automatically controlling the depth and sample thresholds to manage computational cost and overfitting proved tricky across different POPs. We also preferred classifiers supporting L1 penalisation for robustness against noise in captured traffic, and faster convergence [46]. This essentially narrowed the candidates down to logistic regression with a SAGA solver [47] and linear SVM with stochastic gradient descent (SGD) learning, both yielding similar performance during our preliminary evaluations.

While our current implementation does not use SGD's partial fitting, we consider its availability an advantage for future performance improvements as well as a desirable continuous-training feature for high-volume DPI classifiers. So we selected linear SVM with SGD as the generalised ML model out of all POPs studied, the implementation of which was based on the `SGDClassifier` class from `sklearn.linear_model` [48], with hinge loss and L1 penalisation.

Once we changed the scope of analysis from initial packets to all packets, we no longer needed to generate and capture large amounts of synthetic proxy

handshakes. Instead, long positive traces containing one or more pairs of clients and servers could be windowed into equal-sized segments for feature extraction. They will individually remain representative of the underlying POP's behaviour, as long as the traces feature a variety of typical human network activities. In our case, for each proxy protocol, we sampled from the positive trace containing two clients and servers for segments used in SGD training and validation. To ensure portability of trained classifiers, we also performed both positive and negative recall testing with traces captured under different network conditions from those used for training and validation.

5.2 Feature selection and pre-processing

As a starting point of our ML feature set, we largely re-implemented the three classes of features from Wang et al. [12, Sec. 6] for each window: entropy-based (maximum, minimum, and mean entropy in each direction), timing-based (binned intervals between ACK packets in each direction), and packet header features (common payload lengths and ACK ratio). As almost all TCP packets in the window will be ACK packets due to the change of analysis scope, we redesigned the ACK ratio to feature the proportion of ACK packets in the window with the PSH flag set. This was inspired by the fact that all protocols in our study (and most proxy protocols in general) feature longer-than-normal TCP sessions with tighter timing requirements than regular web traffic, significantly bolstering the presence of PSH flags (e.g. 25.1% in *obfs4-1* as compared to 7.6% in *multi-client*).

Pre-processing: In order to extract these features from traces, TCP packets were first chronologically separated into one-minute segments. This served two purposes: ease of implementing batching, and filtering out low-frequency packet exchanges spanning many time segments. Within each time segment, packets to and from each foreign IP address were further segmented into windows of fixed size to compute features. The windows from both positive and negative traces constitute the labelled input set to the classifier. Features

in the input set would then be independently centred to the mean and scaled to unit variance. Finally, the scaled input set will be sampled with a 1:1 ratio into training and validation inputs.

The window size is a variable that can be modified at run time. In our evaluations, the fixed window size was set at 50, after preliminary tests with the original feature set showed that smaller window sizes significantly increased computational cost, and larger window sizes carried increased risk of overfitting due to insufficient input samples. This is roughly consistent with the optimal window size (30) found by Wang et al. [12, 6.1].

Feature refinements: We discovered that whilst the original feature set produced very high TPRs and low FPRs on unseen windows from the same traces used in training, by applying SGD-trained classifiers on traces captured under different network conditions, the positive recall rate drops to under 50% for meek, and almost 0% for Shadowsocks and obfs4. This confirmed that the original feature set is not suitable when applied in its entirety, and did not perform as well on other obfuscation protocols, such as the stateless Shadowsocks. The lack of portability in classifiers trained with the original feature set also suggests that some features contribute significantly to overfitting and depend too much on the network conditions of the training traces, such as the bin widths used for packet intervals.

Following some cursory feature evaluation, we divided the packet-interval and common-payload-length features into binned and statistical (mean, maximum) variants. The binned packet intervals were simplified down to four bins with 1000, 10000, 100000, and 1000000 μs as upper bounds, packet intervals above 1000000 μs (1s) were discarded. Based on the default maximum transmission unit (MTU) of most end-user devices, we binned payload lengths in each direction of flow into fifteen equal-width bins between 0 and 1500 bytes. Both binning schemes were designed to capture protocol-specific features such as meek's frequent small-payload polling packets while maintaining generally-applicable bin intervals.

With a new feature set containing entropy statistics, interval statistics and

bins, payload length statistics and bins, and PSH/ACK ratio, we refined the feature set by ablating each type of feature in turn. We found that removing the statistical variants of both packet intervals and payload lengths in each direction, while preserving the binned variants, significantly improved the recall rates for classifiers trained on all protocols. The final feature set extracted from each window is shown in Table 5.1.

Features		Description
Type	Count	
up/down ratio	1	Ratio between upstream and downstream TCP packets.
entropy stats	6	Maximum, minimum, and mean payload entropy for both directions of stream.
interval bins	8	Proportions of windowed ACK packet intervals falling into four bins, for both directions of stream.
length bins	30	Proportions of windowed packet payload lengths falling into 15 bins, for both directions of stream.
PSH ratio	2	Proportions of TCP ACK packets with PSH flags, for both directions of stream.

Table 5.1: The final feature set for linear SVM with SGD.

5.3 Training

For each of the three protocol obfuscation proxies within this study, we trained and validated a linear SVM classifier on the protocol's two-client positive trace (*obfs4-1*, *meek1*, or *ss1*) and the wired five-client negative trace (*multi-client*). Automatic class weight adjustment was used in place of manual balancing of positive and negative training inputs, which avoids down-sampling. If the input traces contain flows under both high-bandwidth and low-bandwidth conditions, balancing class weights rather than down-sampling allows all conditions to be considered by the classifier and avoid input-induced bias.

The training process was repeated five times for each protocol to produce five trained classifiers for validation and recall testing. Positive recalls were performed on traces of the same obfuscation protocol captured under a different wireless connection (*obfs4-2*, *meek2*, and *ss2*), evaluated based on the resulting TPR. Negative recalls were performed on the wireless *lso* trace, evaluated based on the resulting true negative rate (TNR).

5.4 Results and discussion

Results for the three protocols studied are shown in Table 5.2, comprised of means of shared metrics (TPR, FPR, FBR) from validating the trained classifier on unseen packets from the same conditions, as well as recall results on traces captured under different network conditions, including positive recall (PR) on a trace of the same protocol, and negative recall (NR) on *lso*.

Protocol	obfs4	meek	Shadowsocks
TPR	99.35	99.80	99.51
FPR	0.076	0.004	0.080
FBR	1.503	0.173	1.214
PR	99.91	99.96	99.65
NR	93.94	79.18	85.65

Table 5.2: Classification and recall performance (%) of the SVM classifier on meek, obfs4, and Shadowsocks traces. Figures shown are averages over five runs.

With meek being considered one of the hardest for DPI detection [12, Sec. 6.2], it was surprising that the classifier was able to distinguish meek flows from regular traffic better than both obfs4 and Shadowsocks, despite the changes we made to the feature set to reduce the focus on timing features. However, while the classifier remained effective in identifying meek flows under different wireless conditions, the same did not hold for regular traffic – the negative recall TNR is significantly worse than classifiers trained on obfs4 and Shadowsocks, and the worst FPR seen by Wang et al. (12%). With negative flows seen during training comprising packets transmitted over wired connections, and the classifier trained on meek upstream packets with

constantly varying frequency, the less stable *lso* flows may have encouraged misclassification.

With comparable validation performance on positive and negative traces captured under the same network conditions, under different conditions the classifier mislabelled more negative windows from *lso* when trained on Shadowsocks windows than on obfs4 windows. This was expected due to the stateless design of Shadowsocks, substituting handshakes with a pre-shared key. While obfs4 applies random padding on handshake packets in both directions [25], ultimately more distinctive patterns were created than the unpadded Shadowsocks.

Our ML model is broadly comparable in validation performance under the same network conditions to those developed by Wang et al. [12, Sec. 6] and Soleimani et al. [27]. It also suffered a mostly-negligible penalty in positive recall under slightly different network conditions. While time limitations prevented us from recording wired positive recall traces for more conclusive testing, we expect the eventual TPR loss to be at a manageable level, as the single-client traces captured for positive recall were already captured under connections with worse wireless quality than those used in training and validation. TNR/FPR loss in negative recalls were generally greater than those observed by Wang et al., although at a level we consider reasonable for classifying arbitrary segments of connections. Overall, we believe our model retains an acceptable level of portability for its generalised feature set.

The stateless pre-shared key protocol design of Shadowsocks was made specifically to frustrate the GFW's strategy of detecting proxy handshakes, albeit with some uncertainty, then using active attacks to confirm and block proxy servers. Deng et al. [11] trained a Random Forest (RF) of CART classifiers on packet frequency and duration features of Shadowsocks flows, aiming to develop a broad-scope passive analysis method for accurate detection. While our passive ML model shares the broad-scope nature of their method – inspecting arbitrary segments of suspected proxy connections rather than just the initial packets, we could not find sufficient evidence from their publication that the false positives were properly evaluated either under the same or

different network conditions to evaluate overfitting. We are also concerned about the optimistic extrapolation they applied in evaluating true positives [11, Fig. 2].

Based on results from evaluating our ML classifier model and those of other researchers, it is apparent that a significant challenge remains for POP developers to design obfuscation protocols that can cause sufficiently high misclassification rates in ML models – thus deterring deployment by censors. Based on meek's greater capability in degrading negative recall performance than that of obfs4 and Shadowsocks, it can be said that tunnelling (or fronting) traffic through a legitimate protocol remains a strong obfuscation strategy against ML-based censors who do not block all connections of well-known encrypted protocols altogether.

A possible enhancement to tunnelling may be to implement *split routing*, under which TLS-tunnelled POP traffic from a single client can cross the network edges via a large number of paths leading to different relaying servers, who then "bounce" these traffic towards the same exit server, preventing changes in the user's apparent public IP address and thus improving usability. By reducing the density of traffic flowing between a user within the censored network and each relaying server outside it, the POP traffic will have less distinctive volumetric features and blend better into regular TLS traffic. The reduced quantities of per-foreign-IP proxy packets will also make it harder to train classifiers generating low false positive rates during real operation. Split routing of the tunnelled traffic can be done on a per-TCP-session or per-packet basis, the latter will cause greater overhead but also improved covertness. It will also be possible to return a TLS-tunnelled downstream response via a different route than the upstream request's, but it is important to avoid accidentally introducing identifiable features by implementing client-driven polling or other irregular behaviours of a TLS client, which enabled the payload length attack introduced in chapter 4.

Chapter 6

Improving Entropy-based Attacks

We now turn to the problem of how a censor can measure traffic entropy and entropy distribution efficiently. There is a small literature on how to detect ciphertext or cryptographic keys embedded in software, notably by Shamir and Van Someren [49] and a slightly larger literature on how an intrusion detection system or other middlebox can detect traffic that is encrypted or compressed, so as not to waste time on it; this is reviewed by White et al. [50]. We will first optimise the computational cost required by the entropy distribution and byte uniformity tests by Wang et al. [50, Sec. 5], then describe our implementation of a faster entropy estimation test with similar levels of effectiveness based on Shamir and Van Someren's method.

6.1 Background

The censor is, like an IDS, a 'middlebox', but her task is more complex. Part of it may include looking for encrypted connections, but by now these make up over 80% of TCP traffic [50, Sec. 3]. Part of it is checking encrypted connections that have initially low entropy, as setup payloads of TLS streams

contain substantial low entropy segments specifying unencrypted protocol information and these may disclose POP use. Part of it may be detecting encrypted streams that a POP has padded to look like plaintext, or like the ciphertext of a different protocol. The censor will however want to do the cheapest filtering operations first, so her likely workflow to detect a suspected POP will be to statelessly check whether a packet carries a valid TLS record, if so how long it is, and whether the packets fit a known cluster of POP payload lengths; only then will she run an entropy-based test on the packet. Wang et al. proposed using entropy tests against uniform distributions to detect obfs4's fully encrypted TCP payloads [12, Sec. 5]. The Kolmogorov-Smirnov (KS) two-sample test and the truncated sequential probability ratio test (SPRT) test [50] were effective. However, both are more expensive than using a sliding window, which will be described later in our faster entropy estimation test.

Using real traffic, however, we found that most regular client-to-server payloads have a non-uniform entropy distribution regardless of their position in a TCP session. This is likely due to most of these payloads being client handshakes and upstream requests for specific resources, such as HTTP URIs. While fully-encrypted high-volume upstream traffic does exist, such as encrypted P2P file sharing, this traffic is mostly UDP. It also tends to be largely domestic in most developing countries with Internet censorship, due to limited upstream bandwidth available to both residential broadband connections and also at ISPs' peering or transit points to the global network. As a result, applications requiring high upstream international bandwidth are undesirable and are often throttled. Therefore, TCP packets with uniformly distributed payload entropy are rare in regular traffic, and a passive censor may be inclined to inspect only outbound TCP packets with payload sizes above a certain threshold for entropy distribution or byte uniformity. We cannot rule out the possibility of state censors deploying specialised entropy-testing hardware, just as professional cryptocurrency miners now use high-performance hash processors [51]. However, with conventional hardware, the key to make entropy detection practical is minimising the number of packets

examined.

Limiting the scope of inspection to initial packets of TCP sessions will not be enough, as a POP can cause the client and server to exchange "cover" low-entropy packets while setting up a connection, and transmit legitimate clear-text HTTP messages, such as an improved variant of the FTE handshake [23]. Eventually, the censor will have to analyse arbitrary segments of TCP sessions.

6.2 Designs of improved entropy-based attacks

In integrating entropy-based detection techniques into CovertMark, we sought to manage the computational cost with two approaches: first, determining the minimum length at which each payload should be inspected for effective classification of each protocol; and second, cross-referencing multiple statistical measures to determine the feasibility of inspecting fewer payloads with sufficient confidence.

Minimum inspection threshold: Without reassembling the TCP segments of a large PDU, most segments will carry a payload of no more than ≈ 1450 bytes of payload, with the common MTU being 1500 bytes and Layer 2/3/4 headers taking up some space. The detection technique will lower the minimum payload size required for inspection in an iterative process, with the lower bounds set at 1024, 512, 256, and 128 bytes respectively. In order to correctly process traces captured before LSO, we only examine the first 1450 bytes from any payload. Based on Wang et al.'s findings, we varied the block size used in entropy-distribution and byte-uniformity tests between 16, 32, and 64 bytes.

Cross-referencing measures: While we re-implemented the KS 2-sample tests for both entropy distribution and byte uniformity, we did not re-implement the SPRT test due to its reported high FPR/FNR in the context of POP detection, and Wang et al.'s failure to document the parameter choices in their implementation [31]. However, we did add the Anderson-Darling (AD)

2-sample test for its greater sensitivity to fluctuations at the tails of distributions [52], which may help distinguish between high-entropy obfuscated and normal payloads. Given results from the AD and KS 2-sample tests and the KS byte-uniformity test on the same payload, we used three alternate criteria for fusion: conservative (all decisions are positive), majority (most are positive), and sensitive (any one is positive).

In our implementation, all possible combinations of payload sizes, block sizes, and fusion criteria are tested for each POP to determine the most effective classification configuration.

As this detection technique is applied on client-to-server TCP packets with non-empty payloads, we made an additional optimisation in analysing protocols fronting or mimicking semantically valid TLS or HTTP sessions. If the number of frames in the positive trace carrying valid TLS records or HTTP requests exceeds 10% of total client-to-server frames, the detection technique will automatically narrow the scope down to such frames only for both positive and negative traces, in order to reduce the computational cost while remaining effective in detecting the POP concerned. Note that this optimisation can only be applied to the least narrow configuration if multiple POPs need to be detected.

6.3 Results of improved entropy-based attacks

Neither obfs4 nor Shadowsocks triggered the TLS/HTTP optimisation, as we intended; both carried fully-encrypted payloads with no distinctive signature, while meek did trigger the 10% threshold for TLS, so only client-to-server TLS packets had to be examined. This will happen automatically for other PTs and non-anonymous proxies mimicking or fronting TLS or HTTP.

Consequentially, all client-to-server TCP packets with non-empty payloads in the negative traces were tested when the detection technique was applied on obfs4 and Shadowsocks, while only those with valid TLS records from the negative traces were tested when it was applied on meek. For meek, this

appears to have caused a trade-off between computational cost and FPR on the negative traces, as shown in Figure 6.1. However, at higher minimum test sizes, only client-to-server TLS packets with longer payloads (i.e. containing substantial upstream data rather than an empty polling packet) were tested, and the FPR penalty from reducing the overall sample size and only inspecting packets with valid TLS records was significantly reduced (+0.3% at 1024 bytes minimum test size, from +2.2% at 128 bytes minimum). For obfs4 and Shadowsocks, FPRs remained below 0.2% when only packets with longer payloads were inspected.

FBRs for both Shadowsocks and obfs4 were consistently between 4.27% and 4.38%, likely a result of falsely flagging certain high-entropy segments of TLS connections with different servers in the negative traces. This is slightly higher than those observed for *multi-client* in payload length clustering (Table 4.1), but also at a level we expect to diminish in real traffic inspected by censors. Greater fluctuations in FBR were observed when only TLS packets were tested under meek (between 9.89% and 80.8%), requiring careful selection of a fusion criterion to keep FBR at a manageable level.

Across different minimum test sizes and fusion criteria, the block size used in entropy tests had little to no influence on performance. As larger block sizes lead to shorter test inputs and lower computational cost, but should not be too large that there will not be sufficient number of blocks in most payloads, we assumed a censor would prefer entropy tests with a 64-byte block size during actual operations, so long as the TPR on packets inspected remains reasonable. Table 6.1 shows TPR results for all protocols tested under different fusion criteria.

Under either the Sensitive or the Majority criterion, meek client-to-server packets can be detected and blocked just as effectively as Shadowsocks and obfs4. Low-entropy leading bytes are present at the start of meek TCP payloads, due to their carrying TLS records, so meek payloads are identified more frequently as non-uniform by the AD test. The Conservative fusion criterion performed less well on meek packets with longer payloads, but a substantial improvement to FPR was observed. This was especially the case

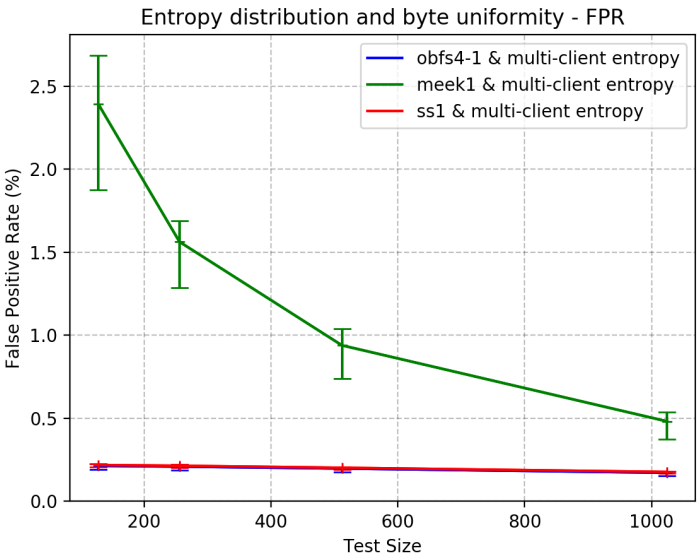

Figure 6.1: False positive rates from applying entropy-distribution and byte-uniformity tests on client-to-server payloads from negative frames in *multi-client*, with minimum test sizes determined from positive traces.

when the TLS-only optimisation was triggered by meek in Figure 6.1, for which the lower limits of error bars consistently represented FPR values obtained under the Conservative fusion criterion. This illustrates the trade-off for censors wishing to apply entropy-based attacks on fewer upstream packets: a more conservative fusion criterion will reduce TPR as well as FPR. As already discussed in our threat model, a reduced but reasonable TPR may not in some circumstances hinder a censor's ability to detect and block POP servers so long as the FPR is low. Therefore, some censors may be inclined to perform multiple statistical tests with a Conservative fusion criterion to confidently detect tunnelling-based POPs such as meek, while only testing packets fitting the profile of the fronted protocol (such as those with valid TLS records) to save on computational cost.

Protocol	Min Test Size	128	256	512	1024
Shadowsocks	Sensitive	99.67	99.79	99.88	99.93
	Majority	95.56	96.75	96.97	97.67
	Conservative	80.62	86.76	89.34	93.18
obfs4	Sensitive	99.89	99.89	99.88	99.86
	Majority	97.22	97.27	97.33	97.31
	Conservative	88.45	88.50	88.51	87.93
meek	Sensitive	99.74	99.78	99.77	99.65
	Majority	97.99	98.31	98.24	97.99
	Conservative	81.80	82.13	81.96	70.31

Table 6.1: True positive rate (%) from applying entropy distribution and byte uniformity test on client-to-server payloads of *ss1*, *obfs4-1*, and *meek1*, with block size 64.

6.4 Going faster: The fast entropy estimation test

Conventional Shannon entropy is expensive to measure because of its logarithmic nature. On an Intel Skylake CPU, floating point division can be done in one cycle, while a logarithmic operation requires 40-100 cycles [53, pp. 236-237]. Furthermore, while the most efficient implementation of the Kolmogorov-Smirnov distribution test can be done in linear time, the leading constant is high [54]. To detect the presence of fully-encrypted TCP payloads, we only need to estimate the distribution of entropy within these bytes, which can be conducted through applying a sliding window over the bytes and keeping track of the number of unique bytes within the window, as demonstrated by Shamir and Van Someren [49, Sec. 3.2].

Because of the computational cost of aforementioned entropy distribution and byte uniformity tests, we implemented a fast entropy estimation test adapted from Shamir and Van Someren's method, in which a sliding window between 32 and 96 bytes in size is applied to all TCP payloads. For each payload, we calculate the mean ratio between the number of unique bytes within each window and the fixed window size. Among the mean ratios for all client-to-server TCP payloads within the positive trace, a percentile threshold determines the mean ratio above which a payload will be considered encrypted, and hence likely to be from a POP. All payloads with mean ratios

above this value will make up the true positives. When the same mean ratio threshold is applied on negative payloads, any flagged by the threshold will be counted as false positives, giving the FPR.

We applied the fast entropy estimation test to positive traces of Shadowsocks, obfs4, and meek. For all three POPs, we found that the mean ratios of fully-encrypted payloads mostly remained consistent, allowing the percentile threshold to be set at 0.1th percentile of all mean ratios. The overall performance was also consistent between different window sizes used: 32, 64, or 96 bytes. With a constant TPR of 99.9%, average FPRs and FBRs from applying the 0.1th percentile mean ratio on negative client-to-server packets in *multi-client* are shown in Table 6.2.

Protocol	**Min Test Size**	**128**	**256**	**512**	**1024**
Shadowsocks & obfs4	FPR	0.225	0.220	0.208	0.181
	FBR	4.491	4.381	4.381	4.381
meek	FPR	3.247	1.868	1.216	0.539
	FBR	98.80	62.65	88.17	24.21

Table 6.2: FPRs and FBRs (%) from applying entropy estimation tests on client-to-server payloads of *multi-client*, with classification thresholds estimated from *ss1*, *obfs4-1*, and *meek1*.

FPRs and FBRs from percentile thresholds obtained from POPs with fully-encrypted payloads (obfs4 and Shadowsocks) remained consistent with those observed for conventional entropy tests, while being able to achieve a better TPR when the percentile-based classification threshold was used in place of statistical tests. While a similar FPR was achieved for meek's TLS-only inspections on longer payloads, other unreasonably high FBRs mean that narrowing the scope of inspection to TLS packets can only be done with statistical tests on more conservative fusion criteria. The execution time was the primary benefit of using the fast entropy estimation test: applying entropy estimation with a 64-byte sliding window on *obfs4-1* took only 21.9% of the entropy distribution tests' execution time on payloads longer than 1024 bytes, and only 21.2% of the execution time on payloads longer than 128 bytes.

6.5 Discussion

A censor's cost-benefit analysis of performing entropy-based detection techniques to detect fully-encrypted payloads can be complex: measuring entropy information on individual payloads require a very substantial amount of computational power with limited insight in return: whether potential cleartext portions exist at arbitrary segments of a long payload, while a ML model trained on additional flow information could perform classification far more cheaply (chapter 5). There are however several advantages pertaining to the deployment of entropy-based attacks: they are stateless and unaffected by fluctuations in traffic volume or changing network conditions; by tuning a few test parameters on small amounts of sample traffic, POPs deploying encrypted tunnelling rather than pseudo-randomisation can still be detected with some levels of effectiveness, such as in the case of meek. If optimised to reduce computational cost, there is a good case for deploying entropy-based attacks in censorship systems, especially when used on arbitrary segments of TCP flows to detect obfuscation protocols with low-entropy decoy handshakes.

In extending entropy-based attacks to arbitrary upstream packets, we maintained their effectiveness in detecting encrypted payloads typical of POPs, while minimising the amount of packets undergoing computationally expensive entropy-distribution and byte-uniformity tests. We also demonstrated a fast entropy estimation test, based on capture-recapture statistics [55], which is just as effective but runs significantly faster on commodity hardware. We hope that, taken together, the tests we have implemented will serve as a good simulation of a capable passive censor for the development of PTs and non-anonymous proxies carrying fully-encrypted payloads.

Chapter 7

The CovertMark DPI Framework

Inspired by the StirMark framework [56] for evaluating the robustness of copyright watermarking systems, we designed CovertMark as a framework for evaluating the covertness of protocol obfuscation proxies (POPs). The goal was to make DPI evaluations of POPs as simple as possible. In addition to substantial generalisations of techniques originally designed for a single protocol, our framework also aimed to standardise as much of the shared setup process for DPI detection techniques as possible, in order to streamline the implementation and integration of new detection strategies.

For users simply wishing to evaluate the covertness of a specific POP or PT, CovertMark requires only positive and negative traces of the protocol, which can be supplied through an interactive shell. Additional flow processing with tools like Bro [57] are not needed. Within the shell, the user interactively programs a *procedure*, which can comprise one or more detection strategies running on the same positive and negative traffic traces of a specific POP for finding the most effective strategy, or a single strategy running on traffic traces from different POPs for finding the most vulnerable POP to the specific strategy. Some detection strategies allow optional runtime configurations (such as whether to consider TLS packets only or all pack-

ets), which can also be automatically inferred from the input traces or have strategy-defined default values. CovertMark will then execute detection strategies in the programmed order and record performance of each. At the end of the benchmarking process, the framework provides the user with a detailed report of the best performance of each detection strategy, together with their optimal configuration parameters on the POP. The framework also offers a range of convenient functions, such as plotting basic graphs and exporting Wireshark display filters for manual inspection of false positives.

The framework is implemented in Python 3, utilising the `dpkt` [58] library for processing PCAP files. Analytic methods associated with detection techniques are primarily implemented with the `SciPy` library [59], and `scikit-learn` [48] for machine-learning models. `Matplotlib` [60] is used for the embedded plotting function. A diagram of the framework's operation flow is shown in Figure 7.1. Note that recall tests are not part of the operation flow, but manually applied as necessary when verifying individual strategies.

7.1 Strategy integration

As discussed in chapters 4, 5, and 6, we integrated four default strategies into the framework: payload length clustering, the SVM ML model with SGD training, the entropy distribution and byte uniformity tests, and the faster entropy estimation method. We opted not to integrate semantic-based detection techniques into CovertMark due to their lack of practicality within a generally non-compliant and noisy web environment [12, Sec. 4]. For developers hoping to integrate their own Python-based detection techniques into CovertMark, the framework includes an abstract detection strategy class that interfaces with existing facilities for traffic data management, result reporting, and scoring to minimise the development effort required.

Strategies share a common data source, consisting of MongoDB documents each storing the record of a packet parsed from a PCAP file, which contains all packet-level information necessary to implement a variety of detection

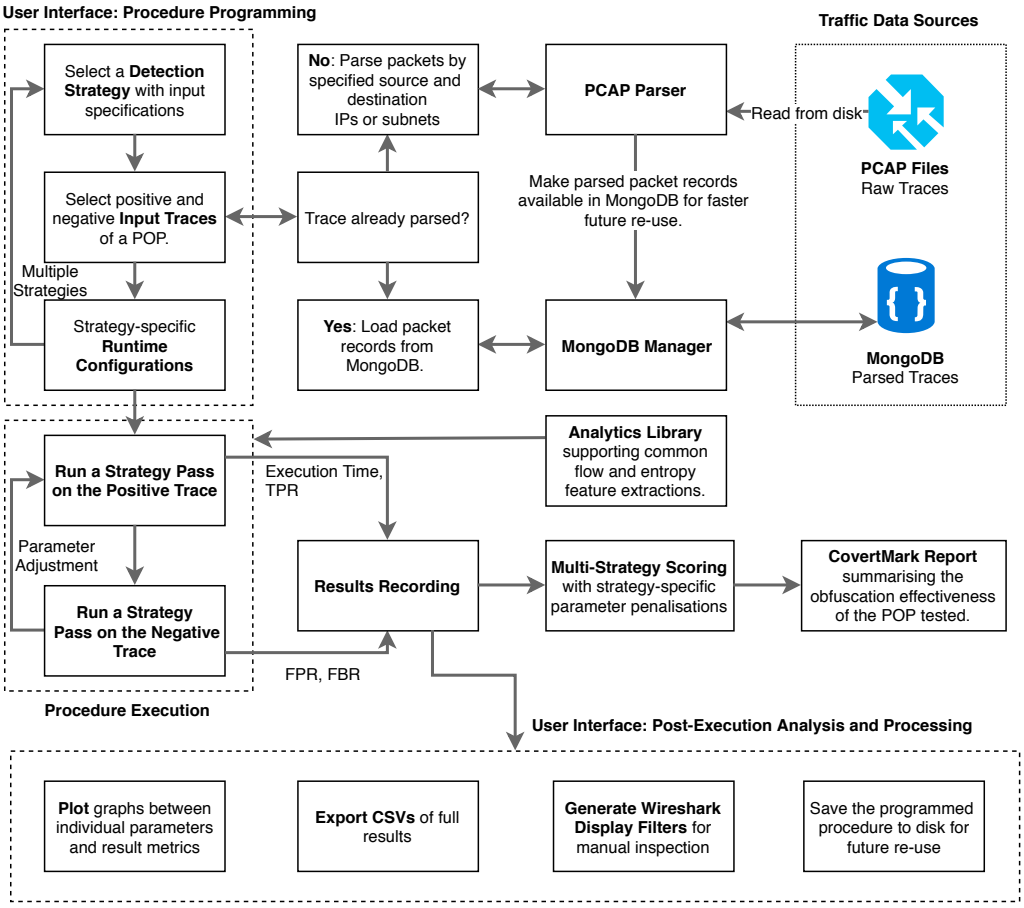

Figure 7.1: The system design of CovertMark.

strategies. Most packet attributes of interest to DPI firewalls below Layer 7 are included. On the application layer of TCP packets, specialised attribute parsing has been implemented for two protocols: TLS and HTTP (requests only). Attribute parsing for other protocols require their parsers' availability within the `dpkt` library, but is otherwise trivial to implement in CovertMark. A full list of packet attributes stored can be found in Appendix A.

A new DPI detection technique can be integrated into CovertMark as a strategy class, implementing the abstract detection strategy we supply. The strategy primarily plays the role of a scheduler to manage the application of shared analytic methods (e.g. flow information extraction, entropy dis-

tribution testing) on positive and negative packet records. As adjustments of analytic parameters will influence the performance of most strategies, the strategy can repeat the analysis on positive and negative traces multiple times with different configurations of such parameters to locate the most optimal configuration. For each of them, the positive run's execution will be automatically timed, along with the resulting TPR, FPR, and FBR, to give the overall score sheet.

7.2 Benchmark scoring

As discussed in section 2.3, a high TPR is not necessarily required for a detection technique to be effective. As directed by the programmed procedure, strategies can also be applied on different traces during the same run, or on the same positive and negative traces with different input filters. It is therefore difficult to implement horizontal comparison on the same metric between results from different strategies. Hard minimum standards were instead set for a strategy to be deemed realistically effective: $\geq$33% for TPR and $\leq$5% for FPR. If no configuration set results in detection performance metrics meeting both standards, the strategy is rejected. Otherwise, an exponential penalty is calculated for each metric based on its deviation from the metric's target. The target for FPR is set at 0.25%; this is based on FPR values considered reasonably low by past studies and results from evaluating our detection techniques. Correspondingly, the target for FBR is set at 0%. Targets for other metrics are dynamically set at the best performance observed among the configurations.

We chose exponential deviation penalties to ensure that configurations with most or all metrics close to their targets will be preferred over configurations that yield a mix of good and bad metrics, as we suspect that a balanced configuration will likely serve a censor better. Due to deviation percentages of metrics being frequently close to zero, the natural logarithm plus one function (`log1p`) [61] is used to calculate penalties, as shown in Figure 7.2,

$$P_{TPR} = \text{log1p}\,(Best\_TPR - TPR)$$

$$P_{FPR} = \text{log1p}\,(\max(FPR - 0.25, 0))$$

$$P_{FBR} = \text{log1p}\,(FBR - 0)$$

$$P_{ExecutionTime} = \text{log1p}\,[(\frac{ExecutionTime}{WorstTime} - \frac{BestTime}{WorstTime}) \cdot 100]$$

$$P_{Overall} = P_{TPR} \cdot 10\% + P_{FPR} \cdot 40\% + P_{FBR} \cdot 25\% + P_{ExecutionTime} \cdot 25\%$$

$$ConfigScore = \frac{\text{log1p}\,(100) - P_{Overall}}{\text{log1p}\,(100)} \cdot 100$$

Figure 7.2: Formulae for calculating deviation penalties of shared metrics as well as the weighted overall configuration score.

where P_M represents the target deviation penalty applied to a metric M. To de-emphasise TPR in our model and place greater emphasis on reducing false positives, the framework calculates the overall configuration score with weights 10%, 40%, 25%, 25% for the configuration's TPR, FPR, FBR, and execution time respectively.

As some configurations will be more difficult to apply in practice than others (such as smaller block sizes causing entropy-based attacks to run slower as discussed in chapter 6), a strategy-specific penalisation factor can be added to the performance score of each configuration parameter. For the payload length attack, larger payload length clusters from higher mean shift clustering bandwidths require the DPI system to filter more packets, so we apply a 2.5% penalty for each additional payload length in a cluster beyond the size of the smallest. For the ML model, no additional penalisation is currently required. An occurrence threshold could possibly reduce false positives by tuning down the sensitivity of the classifier, but will trigger penalisation due to additional states required. Finally, the positive execution time in the shard metrics already represents the cost of entropy-based attacks fairly well, and

the only additional penalisation is based on the number of statistical tests required by the best fusion criteria for each protocol – with a 10% penalty for using Majority with two statistical tests on the same payload, and a 20% penalty for using Conservative with three. The same is not required for the alternative fast entropy estimation test.

We consider a strategy to be reasonably effective on a protocol if its best configuration score is between 75% and 90% after strategy-specific penalisation. At this level we estimate the censor can achieve reasonable detection performance, albeit with some implementation difficulties. If the best configuration score exceeds 90%, we consider the strategy to be very effective and definitely worth implementing by a censor. We believe that this rating scheme reflects the practical usefulness of each detection strategy against the relevant obfuscation protocols, as shown in Table 7.1.

Protocol	Payload Length Clustering	SVM ML Model w/ SGD	Entropy Distribution	Entropy Estimation
obfs4	Ineffective (0)	Effective (94.17)	Effective (90.83)	Reasonable (88.30)
Shadowsocks	Ineffective (0)	Effective (93.41)	Effective (90.74)	Effective (90.88)
meek	Effective (90.66)	Effective (98.21)	Reasonable (80.39)	Reasonable (76.66)

Table 7.1: CovertMark benchmark ratings and scores (in brackets, not to be confused with TPR) of each implemented detection strategy on protocols tested.

Chapter 8

Related Work

Systematic Internet censorship by nation states took off in the 1990s [62], as Internet access was expanded by governments considered to be authoritarian or hybrid regimes [13]. Since then, there have been extensive studies of the mechanisms and effects of these censorship systems. For China's GFW, Xu et al. [63] established through probing that the GFW is a distributed system deployed on the edge equipment of ISPs peering with the global network. Crandall et al. [64] developed another automated probing mechanism to track the keywords triggering blockage by the GFW. Mechanisms used by the GFW to block connections to blacklisted hosts, such as keyword-based filtering through TCP resets were summarised by Anderson [18], while their Iranian equivalents were documented by Aryan et al. [19]. Active attacks by the GFW to confirm and block suspected Tor bridge nodes have been studied extensively by Fifield et al. [8, 9].

With the emergence of DPI capabilities for both active [65] and passive [66] detection of censorship-circumvention tools, extensive efforts have been made to develop effective protocol obfuscation proxies (POPs) with different DPI-evasion mechanisms, with those designed as Tor PTs summarised by Dixon et al. [6]. There are generally three categories of mitigation techniques deployed by POPs: pseudo-randomisation of traffic to erase any identifiable signature, as represented by the deployed obfs4 PT [25] (derived from ScrambleSuit

[7]); transforming traffic to mimic the signatures of a protocol considered as "legitimate" by the censor, as represented by PTs FTE (deployed) [23] and SkypeMorph [36]; and tunnelling traffic through a semantically valid implementation of a "legitimate" protocol, as represented by the deployed meek PT with TLS domain fronting [24]. The non-anonymous DPI-resistant proxy Shadowsocks [41] also implements pseudo-randomisation.

In developing our covertness analysis and testing framework, we sought to include and extend all the relevant recent research, and to give the work a practical focus, we based the development of our detection techniques on three representative protocols. We first studied meek [24] as a starting point for the process of developing obfuscation detection techniques applicable to several fronting and mimicry POPs, such as the resulting payload length clustering technique. We then studied both obfs4 [25] and Shadowsocks [41] as two POPs implementing pseudo-randomising encryption of entire payloads with or without an anonymity requirement – the former being a Tor PT while the latter provides censorship-circumvention without strong anonymity. All three POPs were used to evaluate the generalised ML model and the entropy-based methods we developed or enhanced.

There have been only a few other studies of the detection and classification of POPs, with varying degrees of success. One of the earliest was by Houmansadr et al. [26] in 2013, focusing on semantic verification of flows from mimicry PT protocols such as SkypeMorph [36]. They determined that almost no mimicry-based PTs could produce semantically-perfect imitations of the mimicked protocol. However, the false-positive aspects were then studied further by Wang et al. [12], who established that the noisy and non-compliant nature of web traffic would significantly hinder the use of such techniques in practice through a high false positive rate. We opted not to integrate semantic-based detectors into CovertMark; their protocol-specific nature is also in tension with our design goal of a general framework.

Wang et al. also developed two classes of detection techniques that have been influential, namely entropy-distribution and byte-uniformity tests for detecting encrypted payloads, with obfs4 as subject; and a CART-based ML model

for detecting deployed PTs including meek through flow reconstruction. Both classes of techniques achieved good results during their evaluations: entropy-based tests were shown to be effective against pseudo-randomisation of obfs4 (as the lack of identifiable feature in every fully-encrypted payload itself constituted a pattern); and the ML model was shown to able to detect deployed PT handshakes accurately with low false positives, albeit with degraded performance when ported between network conditions. While their detection techniques were based primarily on the initial packets of a TCP connection, we expanded the scope to arbitrary segments of TCP sessions, due to suggestions for censorship systems to expand their scope to counter active attack-resistant (meek) or stateless (Shadowsocks) POPs [10, 11]. We also generalised the ML model, and improved entropy-based attacks to reduce their computational cost.

Following Wang et al., there have been several new ML models for detecting POPs [11, 27, 28]. Soleimani et al. [27] reproduced the effectiveness of ML models on initial packets of ScrambleSuit and obfs4, but did not explore whether relying on initial packets will work for stateless DPI-resistant proxies like Shadowsocks. This was earlier explored by Deng et al. [11] with a Random Forest model, but they did not evaluate false positives to establish its effectiveness in practice. Our SVM model offers an expanded scope of passive analysis and is effective on all of meek, obfs4, and Shadowsocks. Additionally, Nasr et al. [67] developed a feature-compression technique to improve passive analysis performance, including that of attacks developed by Wang et al. [12]. Our improvements to the entropy-based attacks share some similarities with their compression model.

Due to the scale at which state censors operate, we (and other researchers) assume the primary goal of censors is to detect and block POP servers, rather than spending computational resources on fingerprinting individual websites (a related field of research with a substantial literature, such as [40, 68, 69]). While it is beneficial for a state censor to gain this information even with a certain degree of uncertainty, the high computational cost involved normally only allows them to observe selected high-value users.

Another related field of study to the detection of POPs is the use of DPI techniques for detecting malware traffic and other abnormalities within an intrusion detection system (IDS). In particular, malware's prevalent use of conventional traffic encryption protocols to evade rudimentary detection through keyword and URL matching has inspired the use of handshake fingerprinting by modern IDS implementations. Similar to the GFW's detection of OpenVPN connections via their TLS handshakes [21], identifiable features such as cipher-suite composition and TLS extension set have been designed to detect multiple families of malware with some success [70]. When combined with other features such as the clear-text DNS request or the lack thereof, the identification accuracy and false positive rates could be further improved [71]. While the aforementioned research deployed and selected from a vast range of feature sets, a smaller, generalised feature set applied via machine-learning models can also be effective. An example of this is the largely-unsupervised model by Tegeler et al. [72], which utilises timing-based and quantitive features of TCP flows. This shares some similarities with our SVM model, but their feature set was generalised for IDS-oriented purposes.

Other malware detection methods such as infection sequence models [73] have the potential to be adapted for modelling long-term behaviour deviations between the traffic of a tunnelling or mimicry POP and the cover protocol's traffic. They are however likely to be highly stateful and thus impractical for adoption by censors. With a trend in malware designs towards deploying obfuscation protocols for covert propagation [74], we also expect some of our detection techniques for POPs to become useful in malware propagation detection within local networks of medium and large sizes in the future.

Chapter 9

Conclusions

Estimating the capabilities of state censors has always been a difficult task for academic researchers. Our models have evolved based on the observed effects of censorship systems on different DPI-resistant proxies and Tor PTs, which we unified as protocol obfuscation proxies (POPs) in this study. Proposals from government and military-affiliated researchers [10, 11] have led us to investigate the predictable next generation of stateful detection mechanisms and make them available in our testbed.

Our CovertMark framework basically breaks all the POPs currently in use. The payload-length clustering attack leverages meek's TLS compliance to pinpoint its polling packets with mostly negligible false positives; our generalised SVM-based classifier achieves comparable performance to protocol-specific models from past studies while offering reasonable portability; and we have investigated how to slash the computational cost of entropy-based attacks. We discovered that given sufficient computational power, a censor whose political climate allows them to tolerate false positives can eventually detect and block servers of all currently deployed Tor PTs and the popular Shadowsocks proxy. Our SVM classifier is also a very good general-purpose screening tool for detecting weak proxies. Someone wishing to engineer significantly more covert proxies would absolutely have to work out how to defeat classifiers of this type, rather than just defeating entropy-based tests.

During our study, we also noticed that some features included in the Tor protocol for anonymity reasons [22] make it harder for POPs to circumvent censorship. For example, Tor uses a fixed 512-byte cell size over TCP to mitigate website fingerprinting attacks, but this allows empty-polling and cell-carrying client-to-server TLS packets to be spotted easily in meek's upstream traffic, as shown in Figure 4.1. This was only resolved in obfs4 through generous randomised padding with increased bandwidth overhead. Non-anonymous proxies such as Shadowsocks do not pad or segment encrypted payloads, which we actually found to fit reasonably well into regular traffic.

This brings us to a general question: in developing secure networking systems, are anonymity and censorship-circumvention capabilities necessarily in tension? Featured prominently in Tor's threat model, website fingerprinting techniques rely heavily on the dynamic nature of the modern web. Tor was designed to protect web browsing of the 2000s internet, by providing anonymity against passive adversaries with less than global scope. The world is now different. Is there a fundamental limit to the level of covertness that modern POPs carrying Tor traffic can offer? We hope this question can help stimulate the development of new PTs and further research on traffic analysis strategies. In making the positive and negative traces produced during this study available to other researchers, we hope they will assist future work in these areas.

Appendix A

MongoDB packet record format

The format of packet records stored as documents within MongoDB by CovertMark is listed as followed, from the perspective of a Python-based detection strategy.

Attribute Key	**Attribute Description**
type	Type of IP packet: `v4` or `v6`.
dst	Destination IP address, can be IPv4 or IPv6.
src	Source IP address, can be IPv4 or IPv6.
len	Full length of the packet at IP layer.
proto	Protocol of transport layer, usually `TCP` or `UDP`.
time	The UNIX timestamp marking the packet's capture, with at least 6 decimal places of accuracy.
ttl	The time-to-live of IPv4 packets in milliseconds, or the remaining hop limit of IPv6 packets.
tcp_info	A dictionary containing additional information for TCP packets on the transport layer, as detailed blow. Value is None if the packet is not a TCP packet.
tcp_info.sport	Integer value of source port.
tcp_info.dport	Integer value of destination port.
tcp_info.flags	A dictionary of TCP values and their set/unset (0/1) values, including `FIN`, `PSH`, `SYN`, `ACK`, `URG`, `ECE`, and `CWR` as keys.
tcp_info.opts	A list of (option number, option value) tuples storing the packet's TCP options.
tcp_info.seq	The absolute SEQ number of the TCP packet.
tcp_info.ack	The absolute ACK number of the TCP packet.

tcp_info.payload	The TCP payload carried, which will be Base64-encoded when stored, but is always in raw bytes when available to the detection strategy.
tls_info	A dictionary containing additional information for TLS packets on the application layer, as detailed blow. Value is None if the TCP packet is not a TLS packet.
tls_info.type	The type of TLS message transmitted, one of `CHANGE_CIPHER_SPEC`, `ALERT`, `HANDSHAKE`, or `APPLICATION_DATA`.
tls_info.ver	The version of TLS protocol used, one of `1.0`, `1.1`, `1.2`, or `1.3`.
tls_info.len	The total byte length of all TLS records carried. Each complete TLS packet may carry several TLS records, but usually at most 2.
tls_info.records	The total number of TLS records carried by the complete TLS packet.
tls_info.data	A list of TLS payloads across all TLS records, each is Base64-encoded when stored, but always in raw bytes when available to a detection strategy.
tls_info.data_length	A list of payload lengths matching the corresponding payloads in **tls_info.data**.
http_info	A dictionary containing additional information for HTTP request packets on the application layer, as detailed blow. Value is None if the TCP packet is not a HTTP request packet.
http_info.headers	A dictionary storing HTTP request headers such as `user-agent`, and their values.
http_info.uri	A string storing the HTTP request's uniform resource identifier (URI).
http_info.version	The version of HTTP protocol used, one of `0.9`, `1.0`, or `1.1`.

Bibliography

[1] The Tor Project. Bridge users by country and transport. https://metrics.torproject.org/userstats-bridge-combined.html?start=2018-04-16&end=2018-04-23&country=ru, 2018. [Online; accessed April 24, 2018].

[2] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Independent comparison of popular DPI tools for traffic classification. *Computer Networks*, 76:75–89, 2015.

[3] Sheharbano Khattak, Laurent Simon, and Steven J Murdoch. Systemization of pluggable transports for censorship resistance. *arXiv preprint arXiv:1412.7448*, 2014.

[4] The Tor Project. Tor: Pluggable Transports. https://www.torproject.org/docs/pluggable-transports.html.en, 2018. [Online; accessed April 23, 2018].

[5] Zhen Lu, Zhenhua Li, Jian Yang, Tianyin Xu, Ennan Zhai, Yao Liu, and Christo Wilson. Accessing google scholar under extreme internet censorship: a legal avenue. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, pages 8–14. ACM, 2017.

[6] Lucas Dixon, Thomas Ristenpart, and Thomas Shrimpton. Network traffic obfuscation and automated internet censorship. *IEEE Security & Privacy*, 14(6):43–53, 2016.

[7] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224. ACM, 2013.

[8] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining how the great firewall discov-

ers hidden circumvention servers. In *Proceedings of the 2015 Internet Measurement Conference*, pages 445–458. ACM, 2015.

[9] David Fifield and Lynn Tsai. Censors' delay in blocking circumvention proxies. In *FOCI*, 2016.

[10] Zhou Lin, Li Tong, Mao Zhijie, and Li Zhen. Research on cyber crime threats and countermeasures about tor anonymous network based on meek confusion plug-in. In *Robots & Intelligent System (ICRIS), 2017 International Conference on*, pages 246–249. IEEE, 2017.

[11] Ziye Deng, Zihan Liu, Zhouguo Chen, and Yubin Guo. The random forest based detection of shadowsock's traffic. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2017 9th International Conference on*, volume 2, pages 75–78. IEEE, 2017.

[12] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 57–69. ACM, 2015.

[13] Steven Levitsky and Lucan A Way. *Competitive authoritarianism: Hybrid regimes after the Cold War*. Cambridge University Press, 2010.

[14] Larry Diamond. Facing up to the democratic recession. *Journal of Democracy*, 26(1):141–155, 2015.

[15] Espen Geelmuyden Rød and Nils B Weidmann. Empowering activists or autocrats? the internet in authoritarian regimes. *Journal of Peace Research*, 52(3):338–351, 2015.

[16] King-wa Fu, Chung-hong Chan, and Michael Chau. Assessing censorship on microblogs in china: Discriminatory keyword analysis and the real-name registration policy. *IEEE Internet Computing*, 17(3):42–50, 2013.

[17] Tariq Elahi, Colleen M Swanson, and Ian Goldberg. Slipping past the cordon: A systematization of internet censorship resistance. *Centre for Applied Cryptographic Research (CACR), University of Waterloo, Tech. Rep*, 10, 2015.

[18] Daniel Anderson. Splinternet behind the great firewall of China. *Queue*, 10(11):40, 2012.

[19] Simurgh Aryan, Homa Aryan, and J Alex Halderman. Internet censorship in iran: A first look. In *FOCI*, 2013.

[20] Kejing Yang. The door is closed, but not locked: China's VPN policy. Master's thesis, Georgetown University, Washington, D.C., 2017.

[21] Yi Pang, Shuyuan Jin, Shicong Li, Jilei Li, and Hao Ren. OpenVPN traffic identification using traffic fingerprints and statistical characteristics. In *International Conference on Trustworthy Computing and Services*, pages 443–449. Springer, 2012.

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[23] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 61–72. ACM, 2013.

[24] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[25] Yawning. obfs4. `https://github.com/Yawning/obfs4`, 2014. [Online; accessed April 23, 2018].

[26] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 65–79. IEEE, 2013.

[27] Mohammad Hassan Mojtahed Soleimani, Muharram Mansoorizadeh, and Mohammad Nassiri. Real-time identification of three tor pluggable transports using machine learning techniques. *The Journal of Supercomputing*, pages 1–18, 2018.

[28] Kevin R Dougherty. *Identification of low-latency obfuscated traffic using multi-attribute analysis*. PhD thesis, Monterey, California: Naval Postgraduate School, 2017.

[29] BBC News. Russia Telegram ban hits Google and Amazon services. `http://www.bbc.co.uk/news/technology-43865538`, 2018. [Online; accessed April 24, 2018].

[30] Statisa Inc. Number of internet users in Russia from 2013 to 2019 (in millions). `https://www.statista.com/statistics/251818/number-of-internet-users-in-russia/`, 2018. [Online; accessed April 25, 2018].

[31] Liang Wang. obfs-detection. https://github.com/liangw89/obfs-detection, 2015. [Online; accessed April 25, 2018].

[32] Fei Shen. Great firewall of China. *Encyclopedia of social media and politics*, 22:599–602, 2014.

[33] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[34] Giuseppe Aceto and Antonio Pescapé. Internet censorship detection: A survey. *Computer Networks*, 83:381–421, 2015.

[35] Charles Arthur. China tightens 'great firewall' internet control with new technology. https://www.theguardian.com/technology/2012/dec/14/china-tightens-great-firewall-internet-control, 2018. [Online; accessed April 26, 2018].

[36] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012.

[37] Center for Applied Internet Data Analysis, UCSD. CAIDA Data. http://www.caida.org/data/, 2018. [Online; accessed April 26, 2018].

[38] Lawrence Berkeley National Laboratory. Traces available in the Internet Traffic Archive . http://ita.ee.lbl.gov/html/traces.html, 2008. [Online; accessed April 26, 2018].

[39] Jinliang Fan, Jun Xu, Mostafa H Ammar, and Sue B Moon. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *Computer Networks*, 46(2):253–272, 2004.

[40] Tao Wang and Ian Goldberg. On realistically attacking Tor with website fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016(4):21–36, 2016.

[41] The Shadowsocks Project. Shadowsocks - a secure socks5 proxy. https://shadowsocks.org, 2018. [Online; accessed April 26, 2018].

[42] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[43] David Fifield. meek-client.go
meek-client - pluggable-transports/meek - HTTPS transport. `https://gitweb.torproject.org/pluggable-transports/meek.git/tree/meek-client/meek-client.go`, 2018. [Online; accessed April 26, 2018].

[44] Isabela Bagueros. meek transport evaluation. `https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/MeekEvaluation`, 2018. [Online; accessed May 24, 2018].

[45] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015.

[46] Yichao Wu and Yufeng Liu. Robust truncated hinge loss support vector machines. *Journal of the American Statistical Association*, 102(479):974–983, 2007.

[47] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

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

[49] Adi Shamir and Nicko Van Someren. Playing 'hide and seek'with stored keys. In *International conference on financial cryptography*, pages 118–124. Springer, 1999.

[50] Andrew M White, Srinivas Krishnan, Michael Bailey, Fabian Monrose, and Phillip A Porras. Clear and present data: Opaque traffic and its security implications for the future. In *NDSS*, 2013.

[51] Michael Bedford Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 16. IEEE Press, 2013.

[52] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[53] Agner Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. *Copenhagen University College of Engineering*, 93:110, 2011.

[54] Teofilo Gonzalez, Sartaj Sahni, and William R. Franta. An efficient algorithm for the Kolmogorov-Smirnov and Lilliefors tests. *ACM Transactions on Mathematical Software (TOMS)*, 3(1):60–64, 1977.

[55] Rachel S McCrea and Byron JT Morgan. *Analysis of capture-recapture data*. CRC Press, 2014.

[56] Fabien AP Petitcolas, Ross J Anderson, and Markus G Kuhn. Attacks on copyright marking systems. In *International workshop on information hiding*, pages 218–238. Springer, 1998.

[57] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[58] Kiran Bandla. dpkt: fast, simple packet creation / parsing, with definitions for the basic TCP/IP protocols. `https://github.com/kbandla/dpkt`, 2018. [Online; accessed May 1, 2018].

[59] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001–. [Online; accessed May 23, 2018].

[60] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[61] Nelson HF Beebe. Computation of expm1 (x) = exp (x) - 1. Center for Scientific Computing, University of Utah, July 2002.

[62] A Lin Neumann. The great firewall. *CPJ Briefings*, 2001.

[63] Xueyang Xu, Z Morley Mao, and J Alex Halderman. Internet censorship in china: Where does the filtering occur? In *International Conference on Passive and Active Network Measurement*, pages 133–142. Springer, 2011.

[64] Jedidiah R Crandall, Daniel Zinn, Michael Byrd, Earl T Barr, and Rich East. ConceptDoppler: a weather tracker for internet censorship. In *ACM Conference on Computer and Communications Security*, pages 352–365, 2007.

[65] Philipp Winter and Jedidiah R Crandall. The great firewall of china: How it blocks tor and why it is hard to pinpoint. *Login: The Usenix Magazine*, 37(6):42–50, 2012.

[66] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Format-transforming encryption: More than meets the DPI. *IACR Cryptology ePrint Archive*, 2012:494, 2012.

[67] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2053–2069. ACM, 2017.

[68] Giovanni Cherubin. Bayes, not naïve: Security bounds on website fingerprinting defenses. *Proceedings on Privacy Enhancing Technologies*, 2017(4):215–231, 2017.

[69] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

[70] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, pages 1–17, 2016.

[71] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 35–46. ACM, 2016.

[72] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 349–360. ACM, 2012.

[73] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *USENIX Security Symposium*, volume 7, pages 1–16, 2007.

[74] Christian Rossow, Christian Dietrich, and Herbert Bos. Large-scale analysis of malware downloaders. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 42–61. Springer, 2012.