Adversarially Enhanced Traffic Obfuscation

By

Steven Sheffey

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

May 2020

Thesis Committee:

Dr. Ferrol Aderholdt

Dr. Yi Gu

Dr. Joshua Phillips

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Aderholdt, for keeping me on track throughout the process. Without the support of faculty, peers, and the research community, none of this would be possible.

ABSTRACT

As the Internet becomes increasingly crucial to distributing information, Internet censorship has become more pervasive and advanced. A common way to circumvent Internet censorship is Tor, a network that provides anonymity by routing traffic through various servers around the world before it reaches its destination. However, adversaries are capable of identifying and censoring access to Tor due to identifying features in its traffic. Meek, a traffic obfuscation method, protects Tor users from censorship by hiding Tor traffic inside an HTTPS connection to a permitted host. This approach provides a defense against censors using basic deep packet inspection (DPI), but machine learning attacks using side-channel information against Meek pose a significant threat to its ability to obfuscate traffic. In this thesis, we develop a method to 1. efficiently gather reproducible packet captures from both normal HTTPS and Meek traffic, 2. aggregate statistical signatures from these packet captures, and 3. train a generative adversarial network (GAN) to minimally modify statistical signatures in a way that hinders classification. Our GAN successfully decreases the efficacy of trained classifiers, increasing their mean false positive rate (FPR) from 0.183 to 0.834 and decreasing their mean area under the precision-recall curve (PR-AUC) from 0.990 to 0.414.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
Introduction

Internet censorship has become a global issue. Freedom on the Net 2018 [17] claims that Internet freedom around the world has been on the decline for eight consecutive years. This global rise in censorship creates a need for anti-censorship technology capable of neutralizing this threat. Tor is commonly used to circumvent censorship, but is subject to censorship itself. Traffic obfuscation is capable of mitigating censorship against Tor, but recent attacks have exposed weaknesses in existing traffic obfuscation methods. Traffic obfuscation can be modeled as a conflict between an adversary attempting to detect unwanted traffic and a user attempting to modify their traffic in a way that circumvents this [11]. Adversarial neural networks represent this model well, and present an opportunity to improve traffic obfuscation methods.

Though it was designed as an anonymity network, Tor [10] is frequently used to circumvent censorship. Consequently, access to Tor is frequently targeted or blocked by adversaries performing censorship such as China [37] and Iran [2]. In response, the Tor Project supports numerous "pluggable transports" that employ traffic obfuscation to circumvent blocking [33] including Meek [14], a pluggable transport that uses domain fronting to circumvent censorship.

Domain fronting works by hiding traffic to a forbidden host (such as a censored website or Tor bridge) inside the encrypted payload of traffic to an allowed host (such as `google.com`). In Meek, the forbidden host is a Tor bridge, and the permitted host is usually some major service hosted on the same platform, such as `ajax.aspnetcdn.com` when using `meek-azure`. Because the payload and destination of Meek traffic are encrypted, the use of DPI for censorship is mitigated as the censor lacks the ability to observe a packet's metadata fields. Such an approach requires the censor to block both the forbidden and allowed domains. This exploits censors' unwillingness to block a high profile website such as `google.com` or `amazon.com`, an action that could disrupt business or cause civil unrest [36].

Despite its immunity to simple metadata-based filtering, Meek is not entirely undetectable. Wang et al. [39] were able to detect Meek traffic from regular HTTPS traffic with a FPR as low as 0.00006 using machine learning to train a classification model over side-channel features such as packet sizes and inter-arrival times. As computational power increases, the use of ML to perform censorship becomes increasingly feasible. This presents a significant threat to the efficacy of traffic obfuscation methods, and Internet censorship circumvention in general. In order to address this threat, traffic obfuscation methods must operate with awareness of identifiable statistical features present in their behavior, and work to correct these features.

Generative Adversarial Networks (GANs) have a very similar model to that of traffic obfuscation. GANs are typically composed of two components: a generator and a discriminator [15]. The goal of the generator is to generate realistic looking data, while the goal of the discriminator is to determine whether this synthetic data is real or fake. By training the generator and discriminator in unison, each can learn from the other until an equilibrium is reached. In this work, the generator plays the role of the obfuscator, while the discriminator plays the role of the censor. In this work, we aim to evaluate the efficacy of GANs by modifying statistical signatures of Meek traffic in a way that makes them seem more similar to regular HTTPS traffic.

1.1 **Contributions**

In this thesis, we

1. Develop a framework to efficiently and reproducibly capture web browsing traffic in order to generate large datasets.

2. Aggregate statistical signatures of side-channel features in network traffic captures.

3. Demonstrate a feature-space attack that modifies statistical signatures of Meek traffic

in a way that makes them similar to regular HTTPS signatures.

4. Evaluate effectiveness of this feature-space attack against machine learning attacks.

This thesis is organized as follows. In Chapter II, we discuss Tor, traffic obfuscation, and adversarial machine learning. In Chapter III, we outline related work. In Chapter IV, we describe our data collection framework used to generate the dataset for our experiments. In Chapter V, we describe the process used to extract aggregated statistical signatures from captured packets. In Chapter VI, we describe our adversarial approach to modifying Meek statistical signatures to look more like those from regular HTTPS. In Chapter VII, we describe our evaluation process, and present the results of our adversarial approach. In Chapter VIII, we discuss our results. This thesis ends with conclusions in Chapter IX.

CHAPTER II

Background

In this chapter, we discuss Tor, an anonymous communication network. We then discuss censorship against Tor and methods to evade censorship. Next, we introduce various methods of traffic obfuscation. Finally, we discuss adversarial machine learning.

2.1 **Tor**

Tor is a “circuit-based low-latency anonymous communication service” that operates using a system called “onion routing” [10]. The Tor network is composed of nodes (called onion routers) that forward traffic through the network. To communicate over Tor, clients first choose a path through the network i.e., a circuit. This circuit is chosen using the consensus, which is the list of all public onion routers in the Tor network. Tor circuits are typically composed of 3 onion routers: the entry guard, the middle relay, and the exit node. The role of entry guard is typically assigned to powerful onion routers with high bandwidth. While relays and entry guards require very low risk to run, exit nodes are particularly dangerous to operate, as they transmit traffic out of the Tor network, and may be liable for the traffic they transmit. Once the circuit is constructed, fixed-size cells are then transmitted to and from the client through this path, with layers of symmetric keys used to ensure that each relay is only aware of the cell’s previous source, and the onion router to forward traffic to. Tor provides anonymity to both clients and servers, if the server is configured to act as a Tor hidden service. When using Tor, traffic is not sent directly to its destination, but through a series of relays. This allows users to hide their traffic’s true destination from traffic inspection, making it a useful tool against censorship.

2.2 **Censorship**

Tor’s anonymity and privacy applications have made it a target for blocking. Because the IP addresses of most onion routers are listed publically in the consensus, some censors simply block all or most IPs listed in the consensus [40]. The Tor Project aims to solve this problem using bridges, onion routers that are not listed in the consensus. However, the Chinese government is known to probe and block hosts suspected of running Tor bridges [12], making them similarly weak to IP blocking. Tor traffic may also be blocked via DPI, as the Transport Layer Security (TLS) headers used by Tor have a unique fingerprint [40]. Winter et al. discovered that the Great Firewall of China (GFC) "identifies Tor connections by searching for the cipher list sent by Tor clients" [40], an unencrypted metadata field sent during the TLS client hello which uniquely identifies Tor [40]. The Tor Project continues to address identifiable signatures in Tor traffic [29], as well as obfuscation methods to prevent blocking of Tor traffic.

2.3 **Traffic Obfuscation**

The goal of traffic obfuscation is to make it difficult or impossible for any entity capable of monitoring network traffic (monitors) to fingerprint or extract useful information from the traffic. In the context of Tor, obfuscation is frequently used to prevent monitors from being able to determine that Tor is being used. There are many methods of obfuscating network traffic. The details of an obfuscation method rely on the specific threat model used, or assumptions about the adversary. Dixon et al. classify traffic obfuscation methods as encryption, randomization, mimicry, and tunneling [11].

2.3.1 Payload Encryption

Payload encryption methods such as TLS are effective methods to prevent monitors from gaining any information about traffic's payload. However, payload encryption methods are not sufficient to prevent censorship, because the traffic will still have metadata fields such as IP addresses, domain names, or TLS fingerprints that identify the traffic's true destination or protocol [11]. TLS 1.3 can mitigate this problem somewhat by encrypting the Subject Name Indicator (SNI) [5] in the TLS headers (ESNI), but a client must still make a DNS request for the host before using ESNI. Encrypted DNS solutions such as DNS over TLS and DNS over HTTPS exist [42], but censors could potentially disallow these protocols. In the end, traffic must have a destination IP address, which must be unencrypted in order for networking to function. This introduces at least one point of blocking.

2.3.2 Randomization

One method of obfuscating network traffic is to randomize traffic metadata in a way that the censor's automated DPI engine fails to classify it, treats it as an unknown protocol, and ignores it [11]. This type of obfuscation operates under the assumption that the adversary is only blacklisting forbidden traffic, because if the adversary is whitelisting specific traffic classes, randomized traffic will be immediately discarded [11]. However, whitelisting of traffic protocols is not common, as enumerating all types of permitted traffic can be very difficult.

Obfsproxy is a randomizing obfuscator based on ScrambleSuit [41]. Obfsproxy traffic is encrypted with AES and then shaped with a packet morpher and delayer. As a result, it is difficult to identify obfsproxy traffic as any specific protocol.

2.3.3 Protocol mimicry

Protocol mimicry is a form of obfuscation that masks traffic as a permitted protocol [11]. Mimicry-based forms of obfuscation usually do not attempt to produce legitimate forms of the target protocol, just traffic that looks similar. Examples of this include prepending HTTP headers to Tor traffic [9], disguising traffic as Skype traffic [26], or general traffic transformations such as Format-Transforming Encryption (FTE) [22]. However, protocol mimicry may "deviate, often significantly, from that of messages conforming to the cover protocol" [11] which makes them easier to identify by an adversary using DPI.

2.3.4 Tunneling

Tunneling is an obfuscation method that transmits traffic inside another, more permissible protocol. This includes any protocols such as HTTPS proxies and VPNs [11]. Meek, the tunneling protocol we examine in this work, tunnels traffic inside an HTTPS connection with modified host headers, in order to spoof the traffic's destination. Tunneling obfuscation

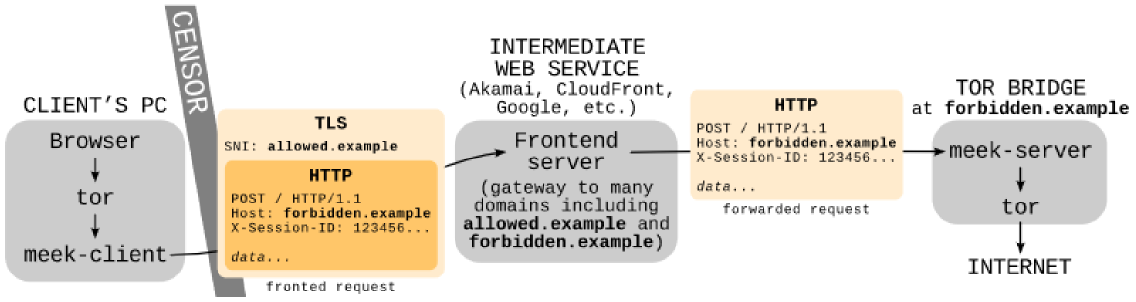

Figure 1: Meek Architecture [14]

methods differ from protocol mimicry obfuscators, because they follow the cover protocol. However, tunneled network traffic may have traffic characteristics that differ from typical use of the cover protocol. For example, Meek traffic (transmitted over HTTPS) exhibits different packet sizes and inter-arrival times than typical HTTPS traffic [14].

2.4 **Adversarial Machine Learning**

Machine learning algorithms are known to be vulnerable to adversarial examples [7]. Adversarial examples are inputs that have been created or perturbed in a way that causes a model to misclassify them; for example, an image of a cat may be modified with small perturbations such that a human still recognizes it as a cat, but a neural network recognizes it as something entirely different, such as a dog.

There exists a form of adversarial neural network called an Adversarial Transformation Network (ATN), in which the generator applies minimal transformations to an input in order to cause the discriminator to misclassify it [3]. These neural networks are effective at producing adversarial data from regular data, and effectively changing the class of a piece of a data from the point of view of the classifier. For this reason, we believe ATNs are very well-suited to traffic obfuscation problems.

CHAPTER III
Related Work

3.1 Network Traffic Obfuscation

In “Blocking-resistant communication through domain fronting”, Fifield et al. compare packet lengths and connection duration in regular HTTPS traffic with those in Meek traffic [14]. They compare HTTPS connections to Google servers from the Lawrence Berkeley National Laboratory dataset with Meek traffic transmitted using Google App Engine as the front domain. Fifield et al. found that while normal traffic and Meek traffic have a similar number of zero-length packets (mostly ACKs), Meek packet sizes exhibit “small peaks at a few specific lengths, and a lack of short payloads of around 50 bytes”. Fifield at al. speculate these differences can be attributed to the fixed cell size of the underlying Tor traffic. Additionally, Meek traffic showed longer connections, due to the aggressive use of HTTP keepalive. However, despite the fact that Meek exhibits longer connection durations, a censor may not be willing to interrupt these connections, because they may represent downloads or other high-value connections. Additionally, Meek is resilient to broken connections, because it uses a session ID header to identify its connection. While Fifield et al. acknowledge that these features represent a way to potentially differentiate Meek traffic from HTTPS traffic, they note that “Circumvention traffic need not be perfectly indistinguishable, only indistinguishable enough that that blocking it causes more and costlier false positives than the censor can accept”. Meek’s utility relies on censors’ unwillingness to create false positives in their censorship, due to either collateral damage, or the sheer cost of false positives in an environment with large volumes of traffic.

3.2 Detection of Network Traffic Obfuscation

In “Seeing through Network-Protocol Obfuscation”, Wang et al. investigate various methods to differentiate obfuscated traffic from non-obfuscated traffic [39]. Notably, they use machine learning to identify Meek traffic. Their approach trains various models using the following features, measured over the first 30 packets of a connection:

- Minimum, average, maximum shannon entropy of payloads in each direction
- Histogram of logarithmically binned TCP ACK inter-arrival times in each direction
- Percentage of TCP ACK packets sent in each direction
- 5 most common packet payload lengths in each direction

The entropy-based features are less useful for Meek traffic, since HTTPS and Meek traffic is typically either encrypted (very high entropy) or empty, zero-entropy ACKs. It provides more utility for other obfuscation methods such as obfs4. TCP ACK inter-arrival times and the percentage of TCP ACK packets in each direction are used due to an observed high areP ACK frequency in Meek traffic.

The authors' best model, a decision tree, achieves a false positive rate of 0.00006 when comparing automatically generated Meek traffic (over Google App Engine) with a dataset of 14 million flows. Additionally, the small feature set used by this attack makes it a promising approach as DPI-based censorship becomes increasingly powerful.

The scripts used to generate traffic for the results presented by Wang et al. are open source. We improve on their data collection method by introducing a parallel work queue system, and implementing more robust error handling and failsafes to ensure that if a failure occurs while collecting data, that failure is handled gracefully and repeated if the error can not be recovered from.

In "Meek-Based Tor Traffic Identification with Hidden Markov Model", Yao et al. distinguish Meek traffic from regular HTTPS traffic using a Hidden Markov Model trained over packet sizes and inter-packet timings [43]. Yao et al. do not describe their Meek data collection process. Overall, their approach further demonstrates the potential for machine-learning based attacks on Meek's obfuscation.

Shahbar et al. also analyze Meek traffic and find that it is identifiable from other types of traffic with an FPR of 0 [35]. However, they do not describe their Meek data collection process or classification model in detail.

Nasr et al. performed flow correlation on Tor traffic, including Tor over Meek [28]. They show that Meek does not provide any protection against flow correlation attacks. They also note that "DeepCorr has a significantly lower performance in the presence of obfs4 with IAT=1". In obfs4, IAT=1 enables obfuscation of inter-arrival times via probabilistic packet delays. This gives confidence to the notion that traffic shaping techniques could be effective for evading flow correlation in addition to censorship. Nasr et al. use the Alexa top 50k as datapoints for their experiments. In this work, we only analyze over the Alexa top 10k.

Verma et al. use adversarial machine learning to modify high-level packet characteristics, in order to fool classifiers into misclassifying traffic protocols [38]. They note that many obfuscated protocols can be identified via statistical signatures, but do not implement measures to modify or perturb these statistical signatures. They also claim that obfuscation techniques such as increasing packet sizes artificially, introducing additional packets (known as chaff), or delaying packets can be modeled as transformations of a statistical signature. They train a GAN using the Carlini-Wagner L2 algorithm to apply perturbations to statistical signatures in order to fool a classifier into misclassifying signatures as different protocols (BULK, DATABASE, MAIL, SERVICES, P2P, WWW). Their adversarial modifications decrease the accuracy of trained classifiers between 1% and 48%.

In this work, we use similar techniques to modify the traffic characteristics of Meek traffic. We also base our features and adversarial transformations on statistical signatures, but our technique operates over frequency histograms, while Verma et al.'s operates over even higher-level statistical features such as mean and quartile packet sizes and inter-arrival times, as described in [27]. Our approach allows for potentially more flexible transformations. Additionally, our model operates over Meek and HTTPS traffic, while Verma et al. focus on protocol misclassification. Finally, we evaluate our adversarial models using differences in PR-AUC and FPR, metrics that focus on censorship feasibility, while Verma et al. use accuracy as a primary metric.

3.3 Adversarial Machine Learning

In this work, we use StarGAN as a basis for our adversarial model [6]. StarGAN was originally designed for applying specific features to images over multiple datasets. StarGAN was chosen due to being a state of the art adversarial transformer, since the complex underlying generator and discriminator can be replaced with simple dense neural networks. Our use of StarGAN comes primarily from its training process. StarGAN optimizes for both transformation that fools a classifier and reconstruction, making it well-suited to traffic obfuscation problems. One concern with transforming traffic based on a pattern would be converting it back to its original format at the server. Without including extra metadata in existing traffic, a server could transform the given signature back to its original class, and determine which bytes are important that way.

CHAPTER IV

Data Collection

Efficiently training machine learning models requires large datasets in order to ensure the model is exposed to a diverse set of inputs within the target domain. To meet this demand, we develop a data collection framework capable of capturing regular HTTPS and Meek flows efficiently and in a controlled environment. Previous work analyzing Meek traffic uses sequential scripts [35], [39] or does not appear to describe their Meek data collection process [28], [43].

4.1 Container-based Supporting Tools

We use Docker [18] to allow our data collection process to be performed in parallel, and in a reproducible environment. Docker is a platform based on reproducible environments known as containers [18].

A major benefit of Docker in a research environment is that it can be used to promote reproducible research. A file called `Dockerfile` defines the Linux distribution, dependencies, and environment variables used to control the environment in which our data collection program runs. By controlling for as many variables as possible, our data collection process produces datasets that only vary based on network conditions, and changes in the websites requested such as time-sensitive content or language changes based on the IP address of the exit node.

Our data collection method is composed of two categories of containers: a work queue and workers. The work queue manages a queue of data collection work, and allows the workers to request work over an HTTP interface. Each work item in the queue contains a URL and a proxy type. Workers navigate to URLs using the given proxy type, and produce packet captures for each piece of work. Proxy types include “normal”, which uses no proxy, and “tor”, which uses Tor over Meek. We use docker-compose [19] to start one instance of the work queue, and 5 instances of the workers. This master-worker architecture allows our data collection process to scale linearly, until a bottleneck such as memory or network usage is reached.

4.2 Data Collection Process

During data collection, each worker repeats the process shown in Figure 2. Overall, the data collections process produces a report containing information about all captured traffic (`report.json`) and a PCAP file for each website visit performed. An example of the data stored in `report.json` can be seen in Figure 3. The field `filename` is generated from random bytes, and `start_time` and `end_time` are Unix timestamps in nanoseconds. `success` is set to false if an error if the worker fails to perform a request. This allows the work to be added back into the queue and repeated in another work request.

4.3 Datasets

We collect datasets from a residential desktop (H), a university office desktop (U), and an Amazon Web Service (AWS) server (A). Datasets H and U were collected using Docker installed on NixOS hosts, while dataset A was generated using an AWS `m5.2xlarge` instance provisioned by `docker-machine`. Each dataset contains 20000 samples, created by navigating to the top 10000 websites of the Alexa top 1M dataset [1] using both regular HTTPS and Meek using the `meek-azure` bridge from Tor Browser. Datasets H and U were collected in Middle Tennessee, while dataset A was generated from the AWS `us-east-1` region (North Virginia) [34].

4.4 Bias

Our datasets contain HTTPS traffic generated with and without Meek. However, HTTPS traffic does not encompass the scope of all Meek traffic. We only collect traffic from connections to the homepages of popular websites, but Tor Browser users may navigate to other pages, use hidden services, or communicate using other protocols. Our data collection framework may be extended to include hidden services, but is not suited to non-web traffic.

1. Request a piece of work by sending a `POST /work/get` to the work queue
2. If there is no more work, the work queue will respond with a 204 status code, and the worker will shut down
3. Send a start message to the tcpdump manager, and wait for a response. The tcpdump manager will start tcpdump, and wait for it to print its initial message. This ensures that tcpdump is fully started before any traffic is generated.
4. If the given piece of work specifies to use Meek, start Tor/Meek, and then wait for 10 seconds to ensure Tor has been properly initialized, and to reduce load on the Meek bridge. Using test traffic to test whether Meek/Tor has started would add noise to the data collection process, and potentially cause bias.
5. Start Firefox using Selenium.
6. Command Firefox to navigate to the given URL using Selenium.
7. Wait for either an element with a common tag (`<script>`) to load, or a 60 second timeout. This is done to speed up the data collection process and avoid getting stuck on pages that never finish loading.
8. Thoroughly shut down processes in the reverse order that they were started
   (a) Firefox
   (b) Tor/Meek
   (c) tcpdump
9. Send a report to the work queue containing information about the work done, and the filename of the generated PCAP file.

Figure 2: Worker program algorithm

```
{"success":true,"work_type":"tor","work":{"index":1,"url":"google.com","
    filename":"05
    aa7fa3350ffe82135ad2fa4518b19158b372c6762165ec9f49791ca0824971.pcap"},"
    type_index":1,"start_time":1581563645608065792,"finish_time
    ":1581563750744979712}
{"success":true,"work_type":"tor","work":{"index":2,"url":"youtube.com","
    filename":"27370324
    e5a8eb171f9059f2d63b381d8cea6f6aafa81d9241695b1ebae0dda5.pcap"},"
    type_index":1,"start_time":1581563845072464384,"finish_time
    ":1581563945478068736}
{"success":true,"work_type":"tor","work":{"index":3,"url":"facebook.com","
    filename":"82
    b2be4051755ac716faaf0d40cef8b967836197d3d989a65260f81909839ab9.pcap"},"
    type_index":1,"start_time":1581563845167630848,"finish_time
    ":1581563946218956544}
{"success":true,"work_type":"tor","work":{"index":5,"url":"wikipedia.org","
    filename":"5
    c620e7838f3ceeea7d4aa877cb921c583ad8fb4a58edf3240ee30da0e5859c6.pcap"},"
    type_index":1,"start_time":1581563845221189632,"finish_time
    ":1581563946530061056}
{"success":true,"work_type":"tor","work":{"index":1,"url":"google.com","
    filename":"09
    f30f27f3b4d97c221a665029cfd28e5b139b3e6167bee52a65e70e5642e0ec.pcap"},"
    type_index":1,"start_time":1581563844856229888,"finish_time
    ":1581563947734976512}
```

Figure 3: report.json, as generated by our data collection process

CHAPTER V

Feature Extraction

5.1 Goals

In this work, we analyze the following side-channel traffic features:

- TCP payload sizes
- Per-direction packet inter-arrival times.

We ignore basic packet metadata such as IP addresses or TLS parameters. While Meek traffic may have distinct values for these features compared to the wide variety of HTTPS clients on the Internet, we assume that these basic fields could be trivially modified. The side-channel features we analyze are acknowledged in the original implementation of Meek [14] and used by Wang et al. [39], Nasr et al. [28], and Yao et al. [43] to identify Meek. These features are identifiable weaknesses in Meek, but their statistical distribution may be modified through traffic shaping techniques; for example, Verma et al. [38] propose inserting extra data (chaff) into packets or delaying packet transmission in order to match a distribution generated by an adversarial neural network. HTTPOS [23] applies a similar technique to HTTP traffic.

5.2 Process

We use Zeek, a DPI engine, to aggregate packets from each PCAP into a set of HTTPS connections [30]. We then associate each packet with an HTTPS connection using its source IP, destination IP, source port, destination port, and timestamp. All packets unrelated to HTTPS connections are ignored. As much more information is found in smaller payload lengths and inter-arrival times than larger ones, we aggregate these features into logarithmic bins. For TCP payload lengths, we use bins of size 10 from 0 to 100 bytes, size 100 from 100 to 1000 bytes, size 1000 from 1000 to 10000 bytes, and a single bin for packets larger than 10000 bytes. For packet inter-arrival times, we use bins of size 1 from 0 to 10 ms, size 10 from 10 to 100 ms, size 100 from 100 to 1000 ms, and a single bin for inter-arrival times

Table 1: An example of features generated for a single flow (rounded to 3 decimal places)

Class	URL	First	Packet Length	Inter-arrival time (to client)	Inter-arrival time (from client)
tor	google.com	true	0.427, 0.002, 0.0, 0.175, 0.048, 0.0, 0.034, 0.002, 0.005, 0.002, 0.007, 0.012, 0.01, 0.002, 0.01, 0.007, 0.007, 0.0, 0.0, 0.209, 0.041, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0	0.601, 0.012, 0.0, 0.0, 0.0, 0.0, 0.004, 0.0, 0.0, 0.0, 0.181, 0.012, 0.0, 0.004, 0.008, 0.0, 0.0, 0.0, 0.004, 0.025, 0.0, 0.0, 0.012, 0.021, 0.008, 0.004, 0.033, 0.021, 0.049	0.684, 0.017, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.006, 0.017, 0.0, 0.011, 0.006, 0.006, 0.011, 0.006, 0.0, 0.0, 0.017, 0.011, 0.0, 0.017, 0.023, 0.011, 0.011, 0.034, 0.034, 0.075

above 1000ms. These bin sizes are similar to those used by Wang et al. [39]. An example of these features can be seen in Table 1, which shows features generated from navigating to `google.com` over Meek.

5.3 **Visualization**

Figures 4, 5, and 6 show the average frequencies of TCP payload sizes, inter-arrival times from the client, and inter-arrival times to the client, respectively over the home traffic dataset (H) as defined in Section 4.3. Inter-arrival times represent the time between two packets sent in the same direction. One difference between normal HTTPS and Meek traffic can be seen in Figure 4 where Meek traffic has a much larger proportion of packets with payload size between 60 and 70 bytes. Additionally, The inter-arrival times of Meek traffic in Figure 5 and Figure 6 seem to indicate a higher latency in Meek traffic. Our observed differences in TCP payload length distribution differ from the TCP payload length distribution measured

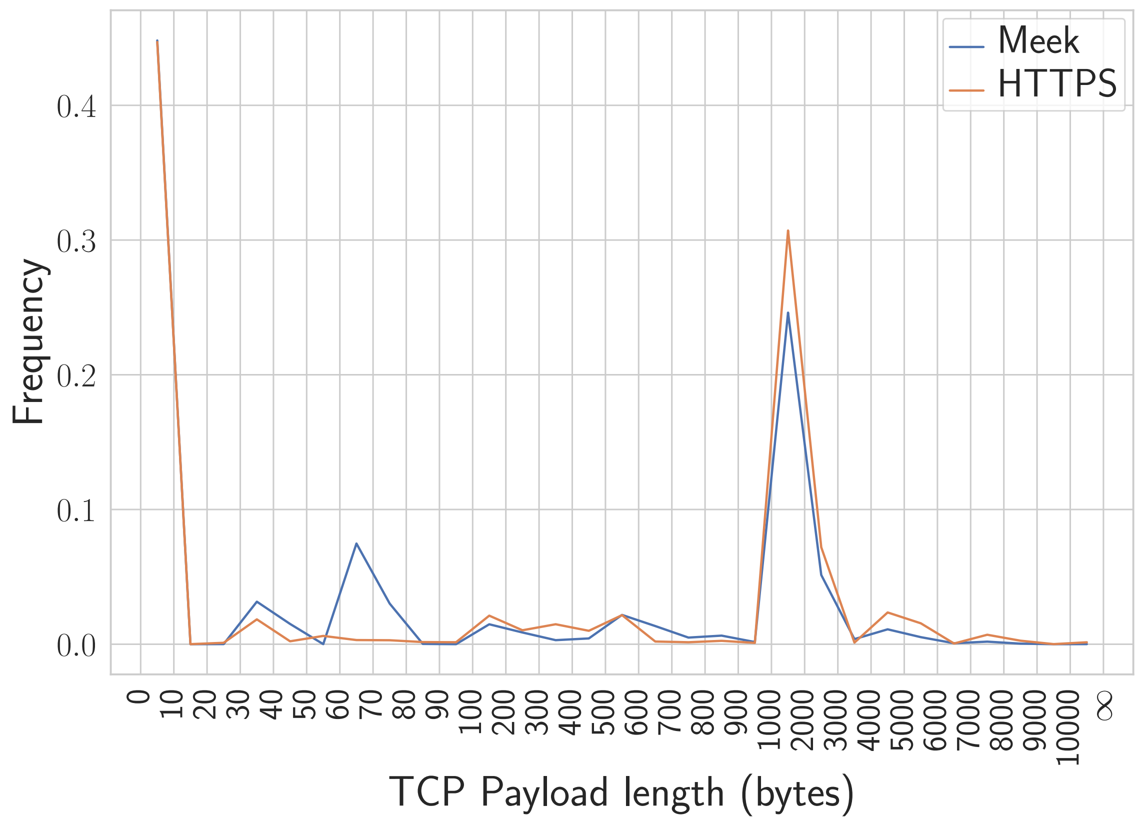

Figure 4: Average TCP payload length frequency

by Fifield et al. [14], where Meek traffic exhibited a much larger number of payloads around 1400 bytes and a lack of payloads around 50 bytes. This may be due to a difference in data sources [14] or modifications to Meek [13]. Fifield et al. [14] compared Google traffic from Lawrence Berkeley National Laboratory to traffic generated by navigating to the Alexa top 500 over Meek. Since Meek's creation, it has introduced many changes such as HTTP/2 support [13], which can result in different traffic characteristics [24].

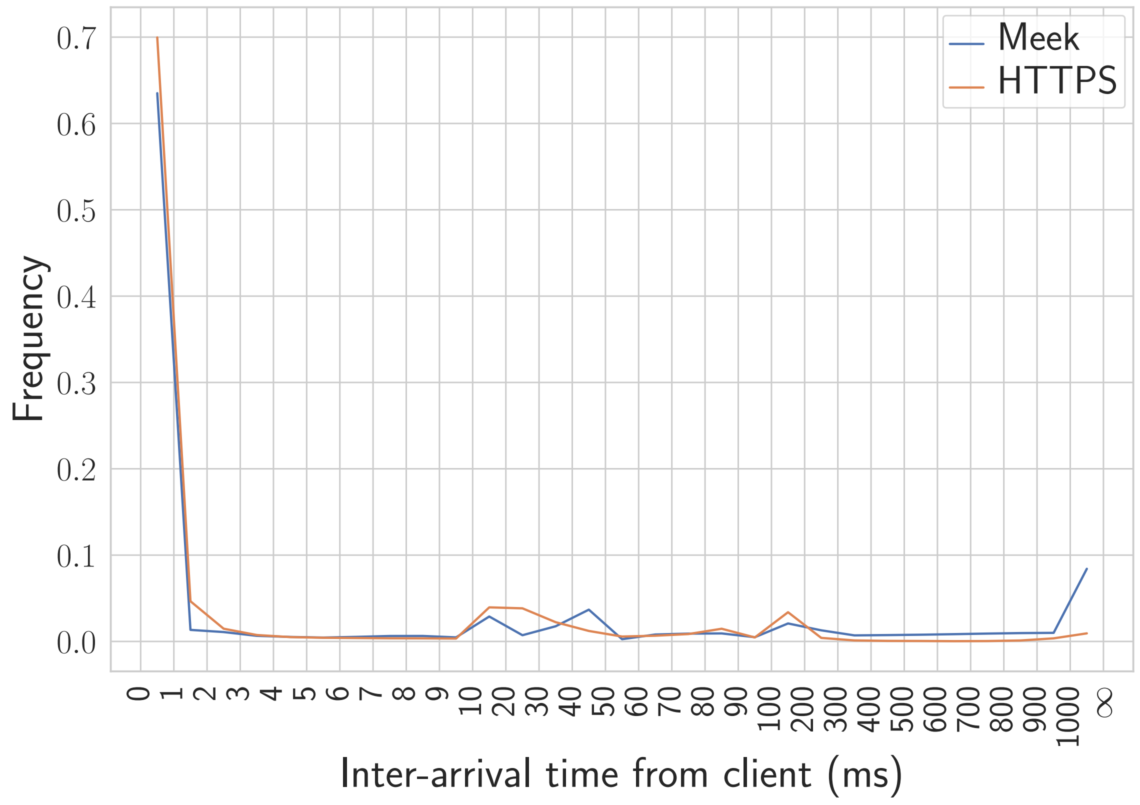

Figure 5: Average inter-arrival time frequency (from client)

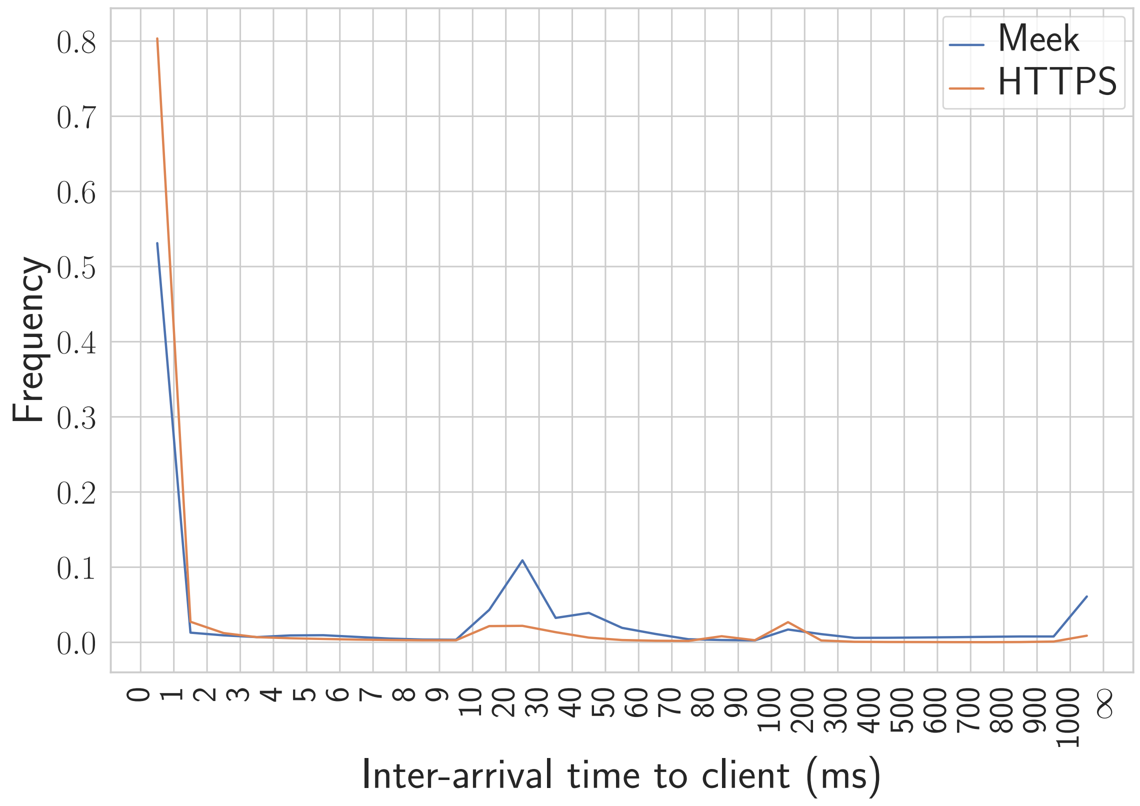

Figure 6: Average inter-arrival time frequency (to client)

CHAPTER VI
Feature Space Attack

While typical GANs contain a generator, which generates adversarial data from noise, our model uses a transformer, which transforms a traffic signature from one class to another. This model is similar to an adversarial transformation network [3]. We use StarGAN [6] as a basis for our model. StarGAN is an adversarial transformation model that transforms images by selectively applying features from various domains and datasets. While StarGAN can support modifying any number of features, we only modify one binary feature: whether the traffic signature represents Meek traffic or normal HTTPS traffic. This allows us to significantly simplify StarGAN while still taking advantage of its powerful adversarial transformation features.

6.1 **Architecture**

In our model, we replace StarGAN's complex, multi-layered convolutional layers with a simple fully-connected hidden layer. While convolutional neural networks are useful for image classification tasks [21], our traffic signatures are simple and can be classified using a small number of parameters. The discriminator accepts a signature, contains a single fully-connected hidden layer of size 16, and outputs a label Y (the probability that the signature represents a Meek flow), and a source S (the probability that a signature was modified using a transformer). The transformer accepts a signature (X) and a target class, contains a single fully-connected hidden layer of size 128, and outputs a modified signature (X'). The transformer contains a larger hidden layer in order to avoid losing information during reconstruction.

6.2 **Training**

We train our model using a modified version of the StarGAN training process. For each training iteration, we train the discriminator using the following steps:

1. Retrieve a batch of 16 signatures X and labels Y from the training set. A label is 0 if

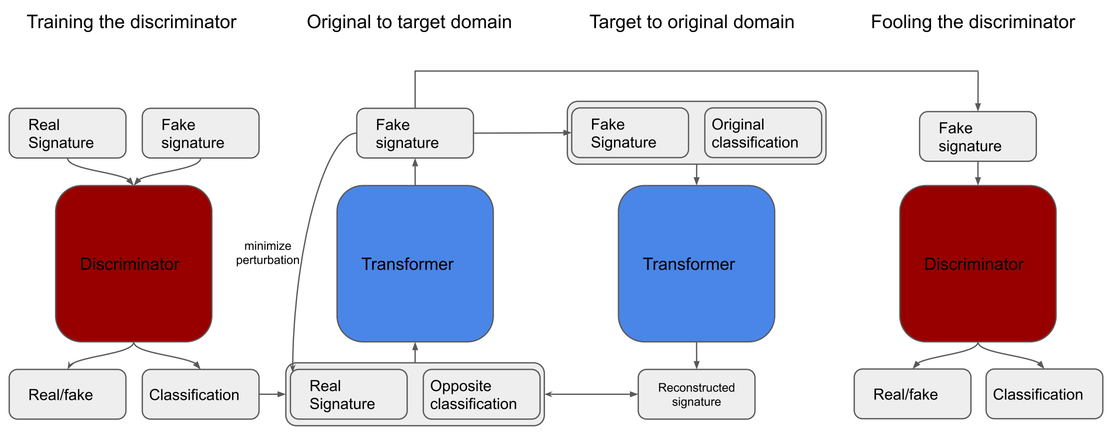

Figure 7: Training process for our GAN, adapted from the original figure in [6].

the flow is a regular flow, and 1 if the flow is Meek.

2. Predict the flow's label and source, and calculate loss for the predictions. BCE is binary cross-entropy.

$$Y_{predicted}, S_{predicted} = D(X)$$

$$DLoss_{cls} = BCE(Y, Y_{predicted})$$

$$Dloss_{src} = -mean(S_{predicted})$$

3. Generate 16 random labels Y_{random}

4. Use T to transform X given Y_{random}

$$X' = T(X, Y_{random})$$

5. Calculate loss for the discriminator's source prediction over the transformed signatures. Class is ignored here, because the class of traffic does not matter if it is determined to

be fake.

$$S'_{predicted} = D(X')$$
$$Dloss'_{src} = mean(S'_{predicted})$$

6. Calculate gradient penalty loss $DLoss_{gp}$, as defined in [6].

7. Calculate the final loss function for the discriminator.

$$Dloss = DLoss_{src} + Dloss_{cls} + DLoss'_{src} + 10DLoss_{gp}$$

8. Perform gradient descent over the discriminator's weights to minimize *DLoss* using the Adam optimizer [20].

Every 5 iterations, we perform a transformer training iteration. This method is used by StarGAN based on [16] to prevent the transformer from overfitting too quickly.

We train the transformer using the following steps:

1. Calculate loss for the discriminator's label and source prediction over the transformed signatures.

$$Y'_{predicted}, S'_{predicted} = D(X')$$
$$TLoss_{cls} = BCE(Y_{random}, Y'_{predicted})$$
$$TLoss_{src} = -mean(S'_{predicted})$$

2. Calculate the perturbation loss. This measures the mean absolute distance between unmodified and transformed traffic.

$$TLoss_{pert} = mean(|X - X'|)$$

3. Transform the transformed signature back to its original class using the transformer, and measure the mean absolute difference between the original and reconstructed signature.

$$X_{rec} = T(X', Y)$$
$$TLoss_{rec} = mean(|X - X_{rec}|)$$

4. Calculate the final loss function for the transformer.

$$TLoss = TLoss_{cls} + TLoss_{src} + 10TLoss_{pert} + 10Tloss_{rec}$$

5. Perform gradient descent over the transformer's weights to minimize $TLoss$ using the Adam optimizer [20].

The signatures generated by our transformer represent a new distribution of packet sizes and timings that, if matched, would make Meek traffic appear similar to regular HTTPS traffic. However, introducing delays or extra data into a traffic stream in order to match this distribution introduces overhead [38]. In order to minimize this, we introduce an additional objective into our transformer's loss function called perturbation loss, defined above in step 2 of the training process. This measures the mean absolute difference between the original signature and the transformed signature. By introducing perturbation loss, we train the transformer to make minimal modifications to the traffic signature while simultaneously fooling the discriminator. By minimizing changes made by the transformer, we reduce the amount of work a traffic shaping method would have to do to modify the Meek traffic stream in order to fool classifiers.

In order to reduce overfitting, a situation in which neural networks generalize poorly due to relying on noise present in the training set, we introduce early stopping measures [4].

Our training process iterates repeatedly over the training set until both the discriminator loss $DLoss$ and transformer loss $TLoss$ have not decreased by 0.0001 over 2000 batches. This is to ensure that D and T cease training when they have reached an equilibrium.

CHAPTER VII

Results

7.1 Evaluation

To avoid biasing our experiments by evaluating models using data that they have been trained on, we split datasets H, U, A (defined in Section 4.3) into three parts:

- 30% GAN training set (G_{train})
- 20% Classifier training set (C_{train})
- 50% Classifier testing set (C_{test})

These splits are chosen to provide a larger amount of data for testing, because both the GAN and classifier reach convergence with very little data.

Our training and evaluation process is composed of 8 steps:

1. Train the Discriminator (D) and Transformer (T) using G_{train}, as described in Section 6.2.

2. Train a neural network classifier and decision tree with C_{train}

3. Split the classifier training set into two equally sized sets

$$C_{train1}, C_{train2} = split(C_{train})$$

4. Transform C_{train2} set into the opposite class using the transformer, while retaining the original (unmodified labels). This is to simulate an adversary who is aware of the

traffic modification scheme, and aims to classify modified traffic as its original class.

$$\begin{aligned} X_{train2}, Y_{train2} &= C_{train2} \\ X'_{train2} &= T(X_{train2}, 1 - Y_{train2}) \\ C'_{train2} &= X'_{train2}.Y_{train2} \\ C'_{train} &= C_{train1} \bigcup C'_{train2} \end{aligned}$$

5. Train a neural network classifier over C'_{train}

6. Evaluate the PR-AUC and FPR of all classifiers over C_{test}

7. Transform C_{test} using T

$$C'_{test} = T(C_{test})$$

8. Evaluate the PR-AUC and FPR of all classifiers over C'_{test}

The neural network classifiers are fully connected neural networks that accept a signature, contain a hidden layer identical to the discriminator, and output the probability that the signature represents Meek traffic. The decision tree is the default decision tree provided by scikit-learn [31], which is identical to the best-performing classifier type in Wang et al. [39].

To avoid overfitting when training C, we separate C_{train} into a smaller C_{train} with 90% of its original size, and C_{val} containing 10% of C_{train}. Each epoch, we evaluate the loss of N using C_{val} to calculate the validation loss. If the validation loss has not decreased by 0.001 in 5 epochs (full iterations over C_{train}), we stop training the classifier.

We evaluate classifiers using PR-AUC and FPR. PR-AUC is a particularly useful metric when dealing with a domain in which the number of negative samples vastly outweighs the number of positive samples [8]. This suits traffic obfuscation well, as most Internet users do not use Meek. Additionally, PR-AUC takes prediction confidence into account, and graphs precision vs recall based on a classifier's ability to confidently provide predictions [8]. FPR is commonly used when evaluating obfuscation methods, as falsely blocking a connection can cause degraded network performance [39]. Additionally, existing work [39] uses PR-AUC and FPR to measure obfuscator classification performance, allowing our work to be more readily comparable.

Finally, to increase confidence in our results, we use a method similar to K-fold validation. We shuffle each dataset, then repeat the training and evaluation process using all 6 orderings of G_{train}, C_{train}, and C_{test}. Our final results are the average of each evaluation metric over all orderings.

7.2 **Results**

The effects of our transformer on classifier PR-AUC and FPR are shown in Tables 2 and 3 respectively. “Naive NN” is the neural network classifier trained only on unmodified signatures, while “Informed NN” is the neural network classifier trained on both unmodified and modified signatures. Our transformer successfully hinders all tested classifiers on all datasets.

Figure 8 shows the average changes in classifier PR-AUC across all datasets. On average, the naïve neural network and decision tree achieve near perfect baseline, while the informed neural network is unable to achieve this. Figures 9, 10, and 11 show changes in classifier PR-AUC for the home, university, and AWS datasets, respectively. While the results for the naïve neural network and decision tree are mostly consistent across all locations, the home dataset performs worse pre-transformation and better post-transformation. Of the three datasets, dataset H has the slowest internet connection, which may affect this.

Figure 12 shows the average changes in FPR across all datasets. The naïve neural network and decision tree have a near-perfect baseline, while the informed neural network has difficulty learning the dataset. The results of transforming data on the naïve neural network and decision tree demonstrate that identifying Meek traffic with the desired modifications made would be completely infeasible from a censor’s point of view, due to the massive number of false positives. Figures 13, 14, and 15 show differences in false positive rates for the home, university, and AWS datasets, respectively. The baseline and transformed results for the naïve neural network remain entirely consistent, while the results for the informed neural network and decision tree vary. Again, the informed neural network performs poorly on dataset H in Figure 13, likely due to differences in internet speed.

Table 2: Effect of transformer on PR-AUC

Data set	Classifier	Baseline PR-AUC	Modified PR-AUC
H	Naive NN	0.999	0.309
	Informed NN	0.915	0.583
	Decision Tree	0.998	0.476
U	Naive NN	1.000	0.309
	Informed NN	0.999	0.428
	Decision Tree	1.000	0.503
A	Naive NN	1.000	0.309
	Informed NN	0.999	0.309
	Decision Tree	0.999	0.503
Avg	Naive NN	1.000	0.309
	Informed NN	0.971	0.440
	Decision Tree	0.999	0.494

In Figures 16, 17, and 18, we show an example of our features over dataset H. These graphs visualize normal signatures from dataset H_{normal} and transformed Meek signatures ($T(H_{meek})$), with a transformer T that has been trained on the entirety of dataset H. Compared to Figures 4, 5, and 6, the differences between modified Meek and Normal traffic are much less pronounced. One notable traffic signature modification can be seen in Figure 16, where the difference in payload lengths between 60 bytes and 70 bytes has been reduced. Modified Meek inter-arrival times, shown in Figures 17 and 18 are much closer to those of normal traffic, and the frequency of inter-arrival times above 1000 ms has been reduced.

Table 3: Effect of transformer on FPR

Data	Classifier	Baseline FPR	Modified FPR
H	Naive NN	0.005	1.000
	Informed NN	0.654	0.667
	Decision Tree	0.001	0.999
U	Naive NN	0.000	1.000
	Informed NN	0.351	0.501
	Decision Tree	0.000	1.000
A	Naive NN	0.002	1.000
	Informed NN	0.630	0.833
	Decision Tree	0.001	0.510
Avg	Naive NN	0.002	1.000
	Informed NN	0.545	0.667
	Decision Tree	0.001	0.836

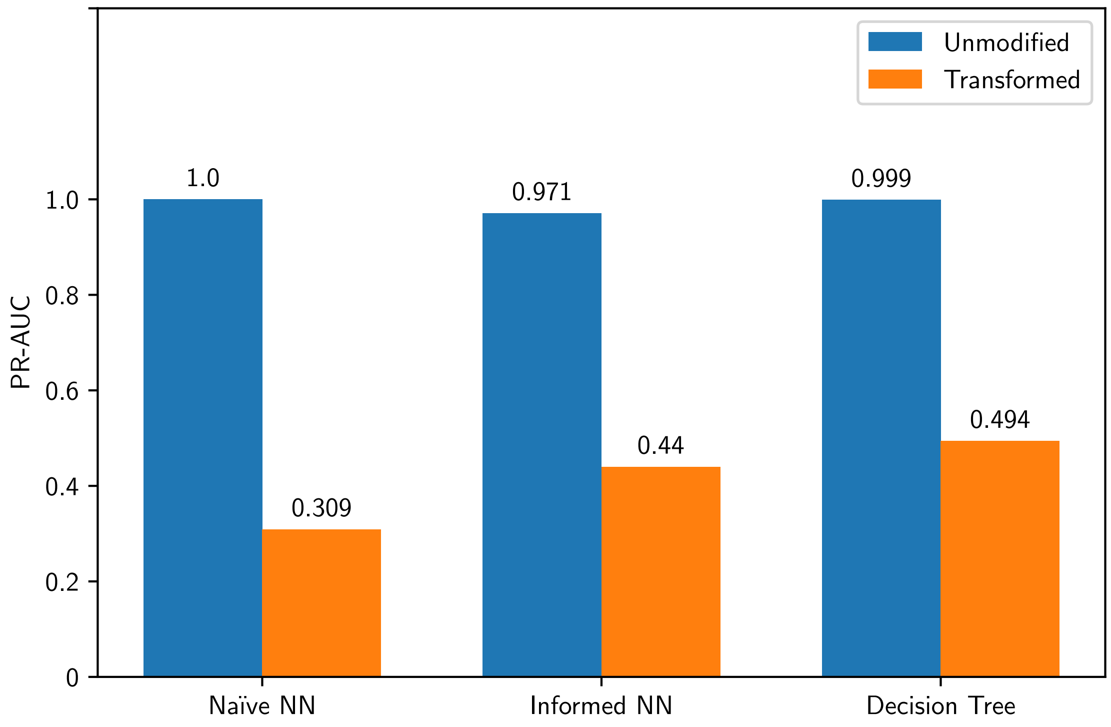

Figure 8: Average PR-AUC differences over all datasets

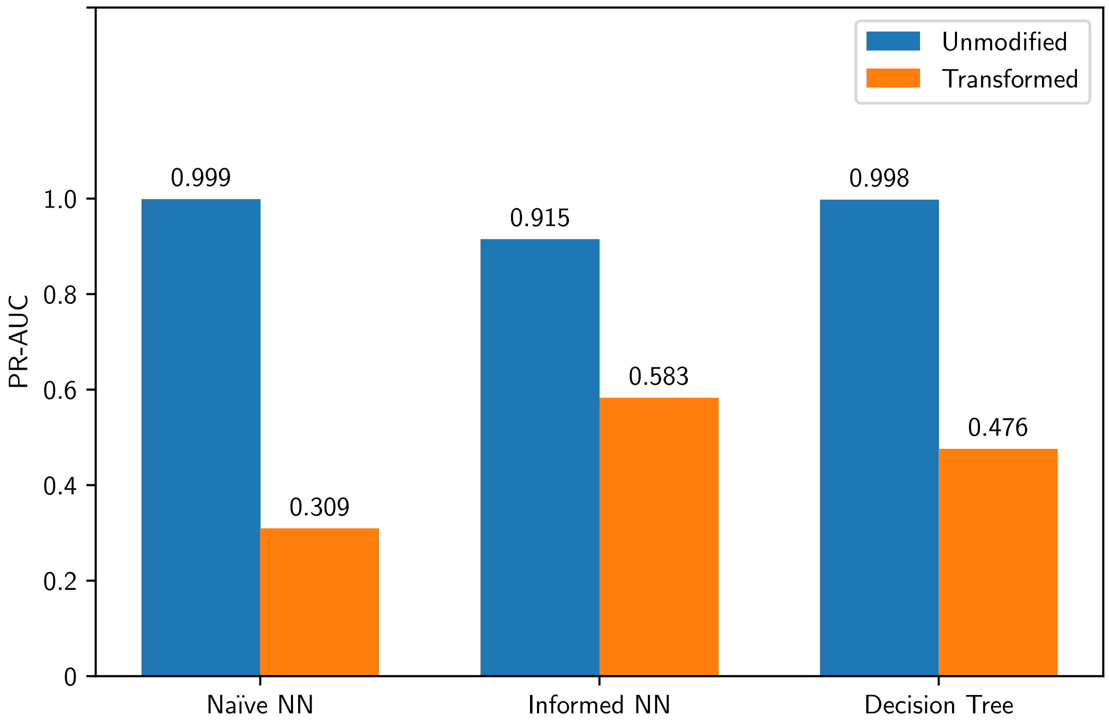

Figure 9: Average PR-AUC changes over dataset *H*

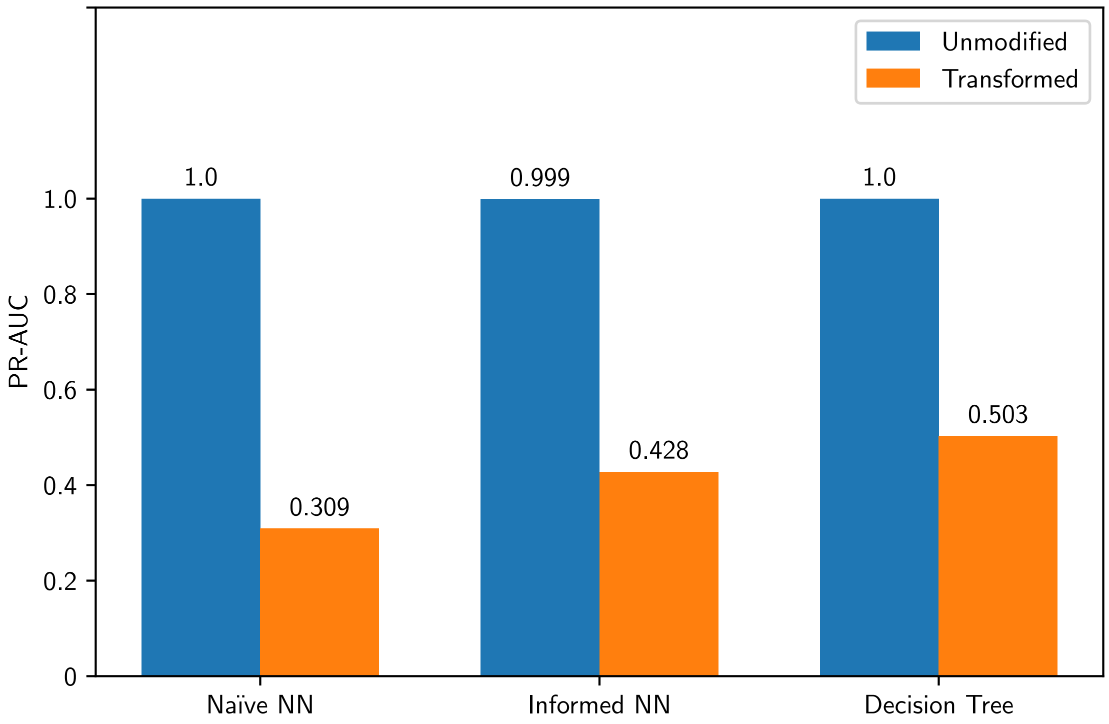

Figure 10: Average PR-AUC changes over dataset U

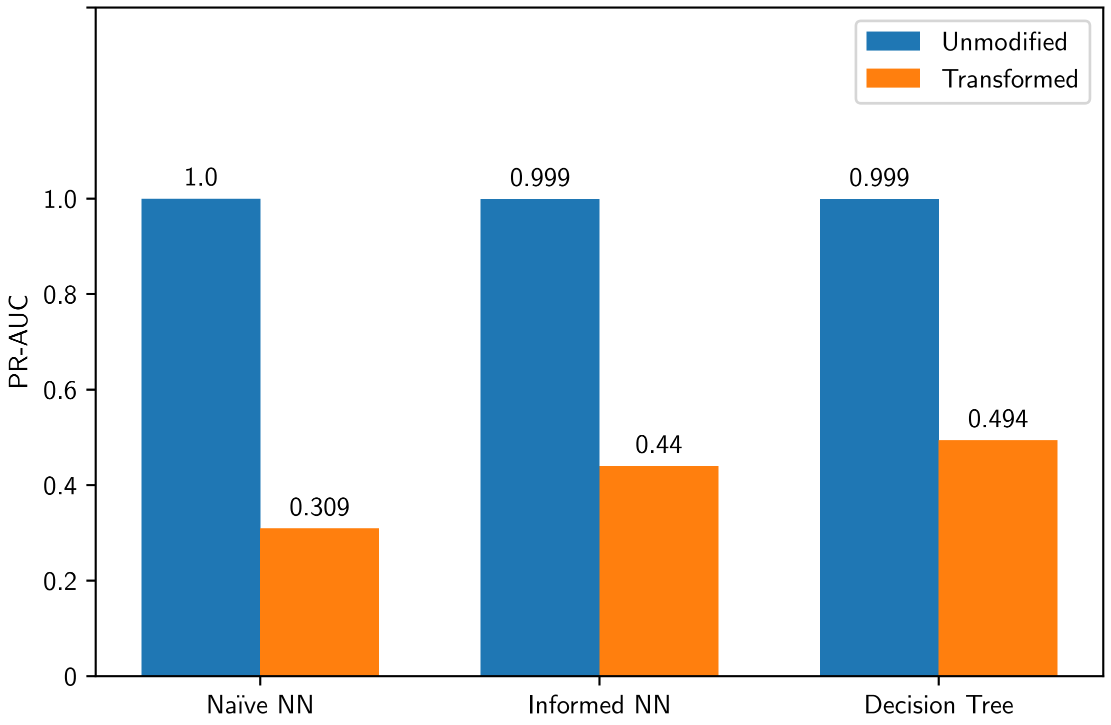

Figure 11: Average PR-AUC changes over dataset *A*

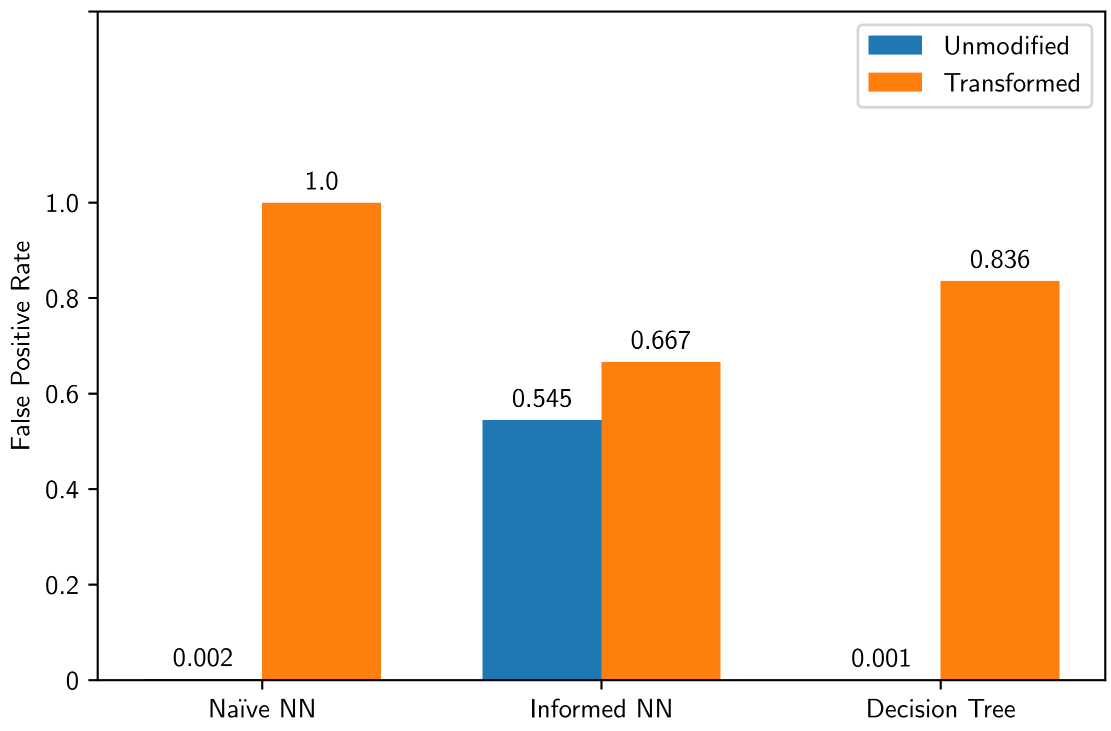

Figure 12: Average FPR differences over all datasets

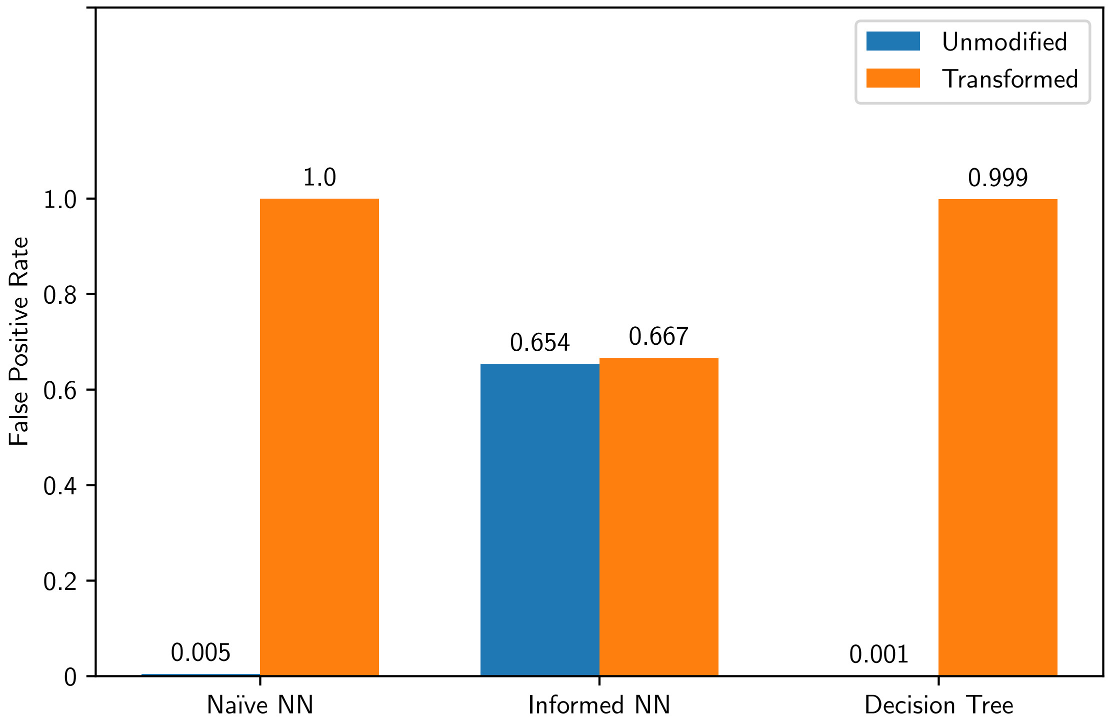

Figure 13: Average FPR differences over dataset *H*

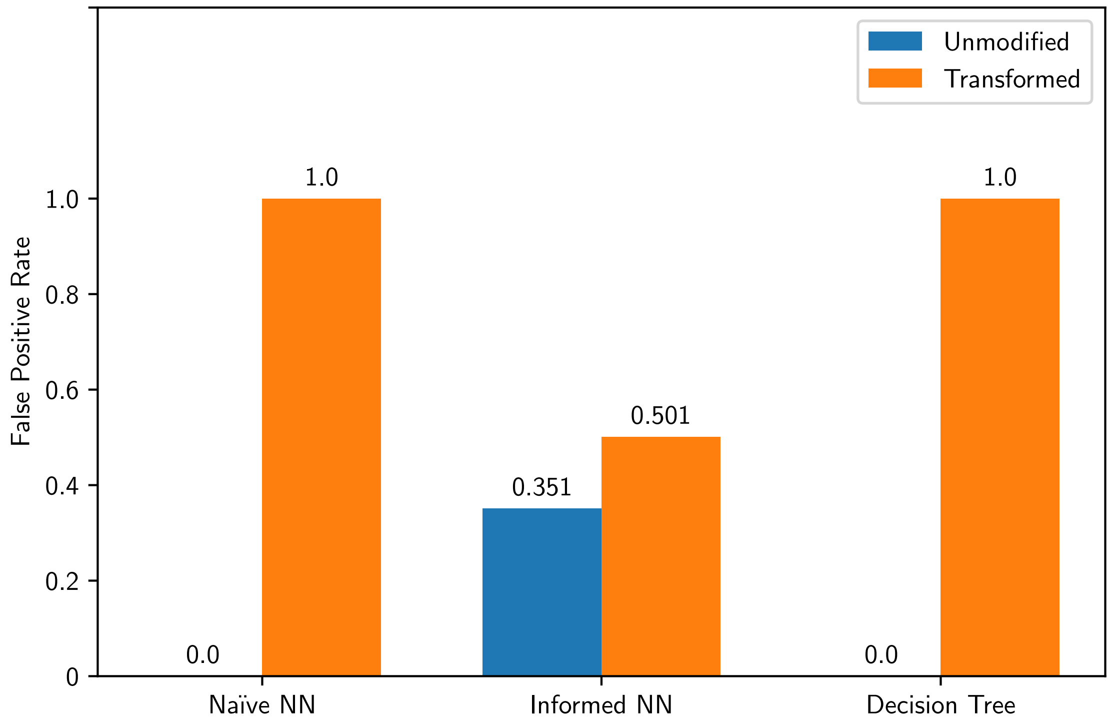

Figure 14: Average FPR differences over the dataset *U*

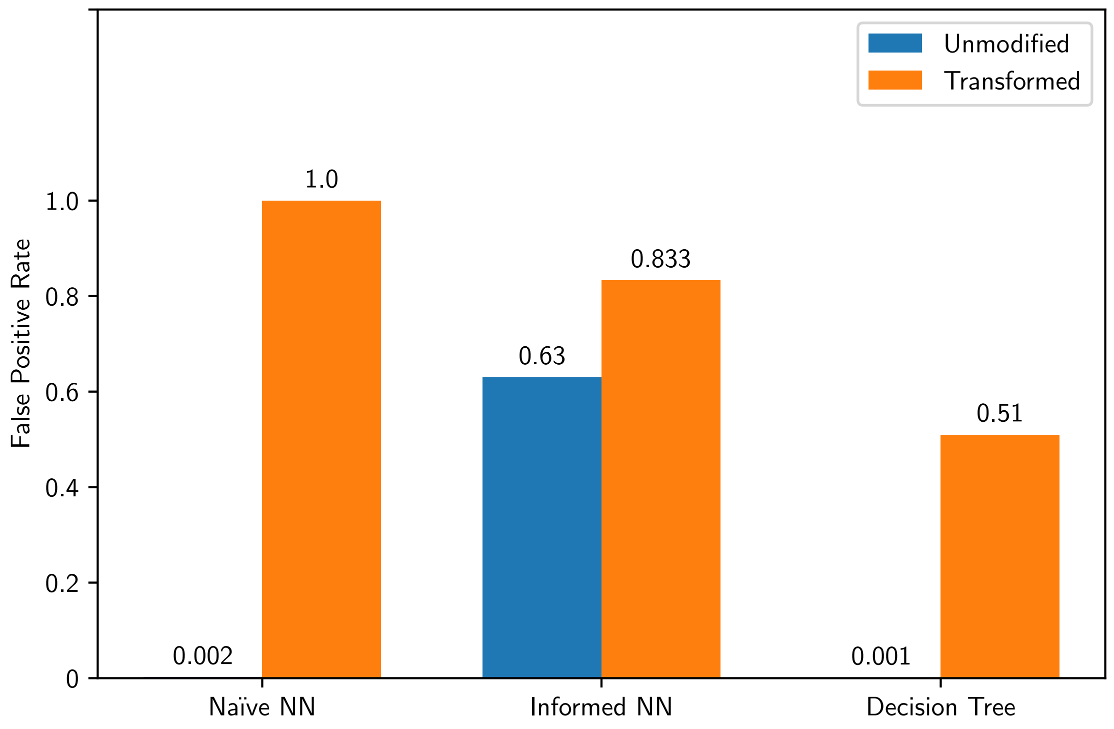

Figure 15: Average FPR differences over dataset *A*

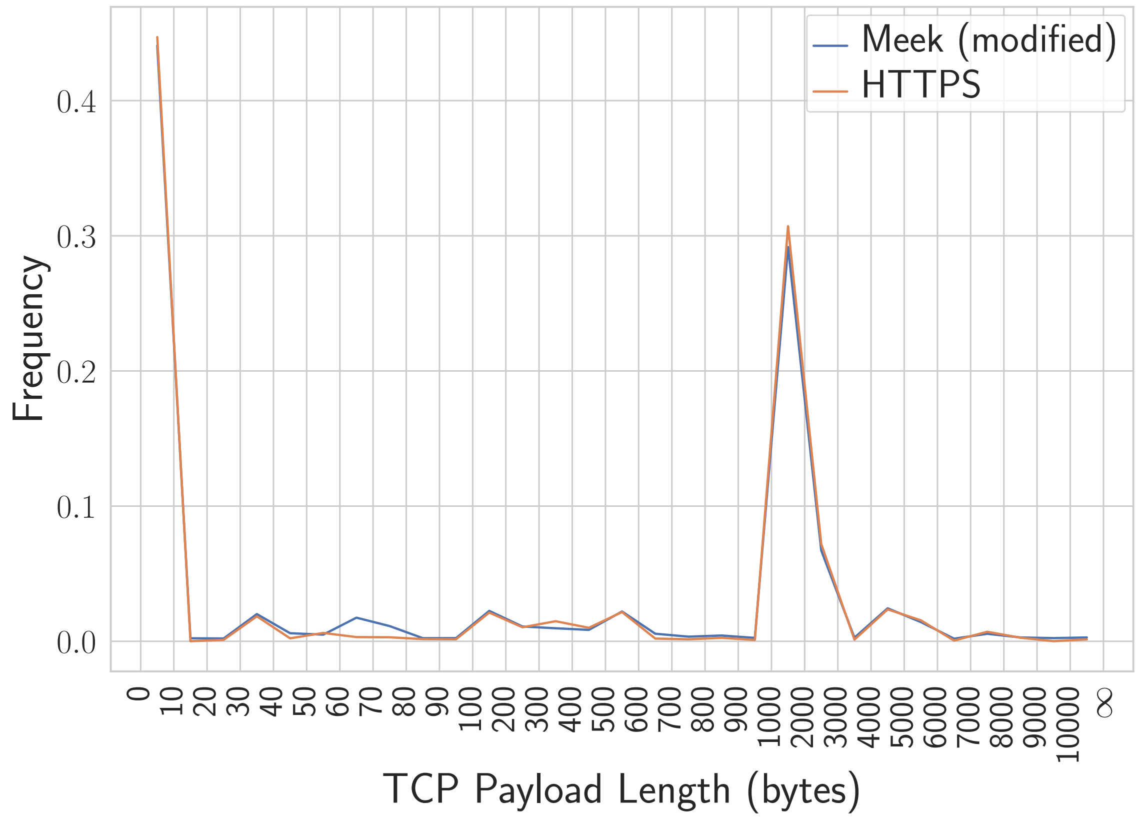

Figure 16: Average TCP payload length frequency over dataset H

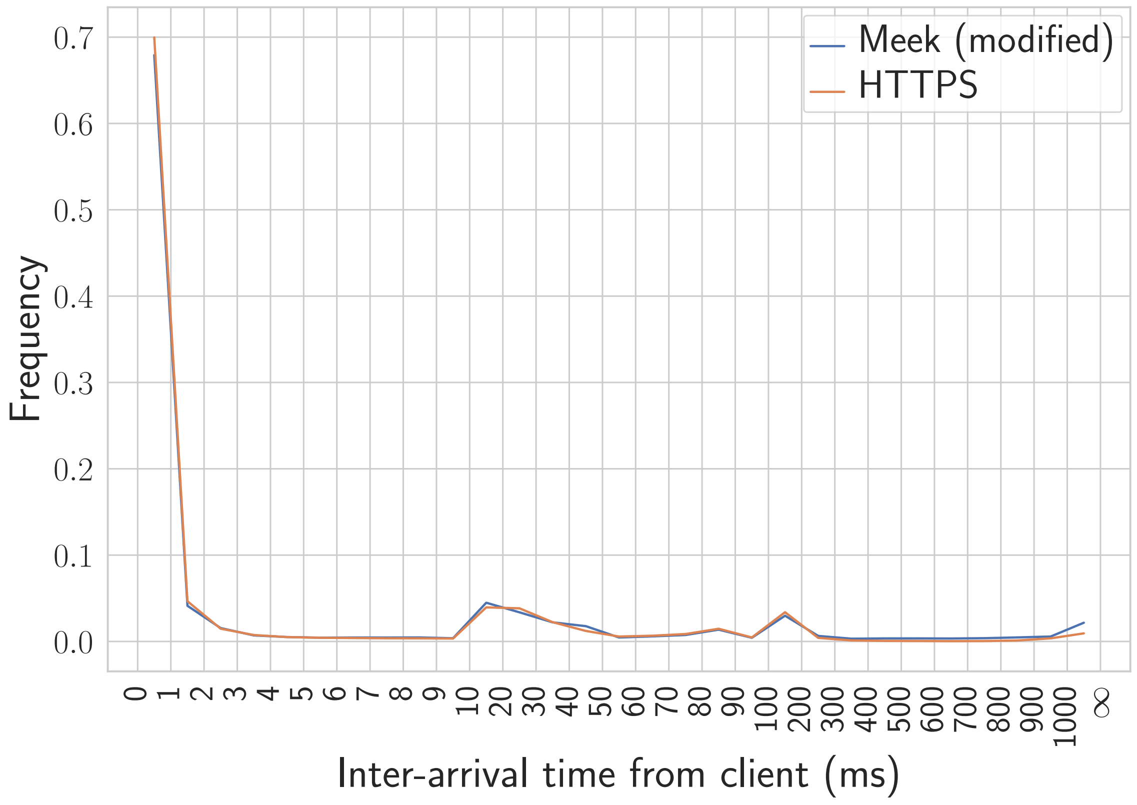

Figure 17: Average inter-arrival time frequency (from client) over dataset H

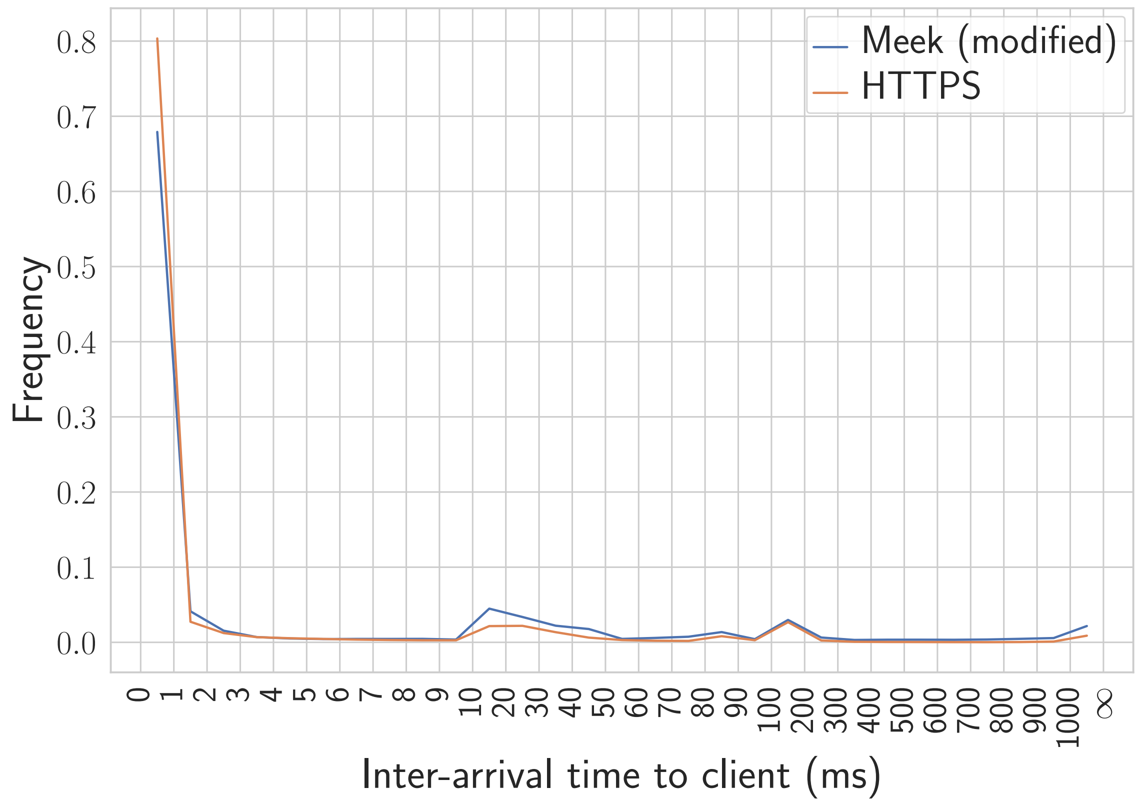

Figure 18: Average inter-arrival time frequency (to client) over dataset H

CHAPTER VIII

Discussion

The baseline classification results in Figures 2 and 3 show that Meek is easily identifiable using machine learning attacks. In every case, our classifiers trained on unmodified data achieved PR-AUC above 0.998 and FPR below 0.005. Wang et al. [39], Yao et al. [43], and Nasr et al. [28] also achieve impressive classification results. However, this strength can also be a weakness. Machine learning models are prone to overfitting [4], making them sensitive to perturbation. For example, over all datasets, the naive neural network achieves a PR-AUC of 1.00 on unmodified data while the informed neural network achieves a PR-AUC of 0.971. However, when classifying modified data, the neural network trained with unmodified data achieves a PR-AUC of 0.309, while the neural network trained using both unmodified and modified data achieves a PR-AUC of 0.440. However, the informed neural network tends to perform poorly in terms of false positive rate compared to the naïve neural network. This may be due to irreconcilable noise caused by conflicting information between the unmodified and modified training set.

Because we ignore hostnames, we lose some identifiable features. During training, we compare Meek traffic to all regular HTTPS traffic, rather than with HTTPS traffic to the Meek fronting host. For example, the `meek-azure` bridge uses `ajax.aspnetcdn.com` as the fronting host [32]. This host typically serves "popular third party JavaScript libraries such as jQuery" [25]. Traffic that mimics average HTTPS traffic to all domains may appear unusual to an adversary compared to typical traffic through this host. In this work, we assume that an increase in false positive rate is sufficient to make classification of Meek traffic less feasible, but future work may target hosts on the Content Delivery Network (CDN) used for domain fronting during data collection.

Additionally, our dataset lacks geographical diversity. All traffic was generated from the Eastern US, and models generated from this data may not be useful to Meek users in other countries. While the Alexa top 1M dataset [1] contains websites from around the world, the data collection workers are set to use an English locale, which may result in latency differences compared to requesting the webpages in other languages.

CHAPTER IX

Conclusions

In this work, we develop a data collection framework capable of efficiently producing reproducible packet captures of Meek and normal HTTPS traffic. We evaluate multiple classification methods over this captured traffic and train classifiers capable of identifying Meek. We then show that our adversarial modification scheme is capable of modifying traffic signatures in a way that reduces average classifier PR-AUC from 0.990 to 0.414 and increases average classifier FPR from 0.183 to 0.834.

While we focus on Meek and normal HTTPS traffic in this work, our adversarial modification scheme and data collection framework can potentially be applied to any Tor pluggable transport in order to identify and correct for weaknesses. In the future, adversarial models could be applied to shape traffic in real-time in order to improve any obfuscation method that relies on protocol mimicry or tunneling.

As adversaries performing censorship become more advanced, researchers developing obfuscation methods must become aware of their capabilities. Performing classification and transformation simultaneously using adversarial machine learning can allow researchers to model theoretical capabilities of both the censor and the obfuscator.

9.1 **Future Work**

Adversarial techniques show great promise as a method of evading censorship, fingerprinting, and flow correlation. We intend to pursue this field of research further, and eventually build tools that allow users to take advantage of adversarially trained traffic shaping to improve their privacy.

9.1.1 Adversarial Traffic Shaper

One possible approach to this would be an extension to Meek that modifies either the transmitted traffic or the traffic transmission strategy to produce HTTPS flows with traffic characteristics similar to those of typical HTTPS traffic. If successful, this could provide an additional layer of protection to Meek users. However, there are numerous problems that must be solved before this becomes feasible. These issues include practical implementation, bias, and efficiency.

Our results show that the effectiveness of classifiers can be hindered when adversarial techniques are used to modify high-level features of Meek traffic. However, applying our methods to actual traffic would require mapping high-level modifications of traffic features to low-level traffic scheduling and shaping algorithms. This presents an opportunity for future work.

Another potential concern of a potential adversarial traffic-shaper for Meek is that all machine learning algorithms have bias. Regular HTTPS traffic likely has very different characteristics in Tennessee than it does in China, for example, so if a Meek user in mainland china used an American model, the traffic still might be anomalous. This could be due to differences in network infrastructure, websites visited, popular software, and a variety of other factors. Tailoring a machine learning model for traffic shaping in a specific situation would require generating new data and training new models for different areas. This opens up new areas of research about the granularity required to have an effective model against censorship, and the transferability of models. Generating datasets may also be difficult if Meek is being censored already. A possible solution may involve decentralized networks to share examples of regular traffic characteristics to mimic, but this comes with problems such as bad actors, and also privacy concerns.

Traffic shaping that performs actions such as injecting additional packets and payloads, or delaying packets introduces some level of overhead to an obfuscation method. Given that Tor and Meek introduce significant overhead already, any additional may further decrease usability. We attempt to address this potential concern by minimizing modification to high level features, though an implementation of this transformation may need to operate under different constraints.

CHAPTER X

Availability

All code used to produce the results in this work including the traffic generation framework, feature extractor, and machine learning code is open source, and can be accessed at `https://github.com/starfys/packet_captor_sakura`

The project is split into three directories:

- `data_collection`
- `data_generator`
- `analysis`

`data_collection` contains code used to generate new Tor/Meek datasets. Users can modify `docker-compose.yml` to point to a folder on their machine, then run `make scale` to run the data collection process with 5 workers. The number of workers used can be modified by changing `CAPTURE_SCALE` in `Makefile`. The dependencies required to run programs in this directory are Docker and Make.

`data_generator` contains code to extract features from data generated by `data_collection`. The command to run `data_generator` is `cargo run --release -- DATA_DIR OUTPUT_DIR` where `DATA_DIR` is the directory containing `report.json` and PCAP files, and `OUTPUT_DIR` is the directory the program should output features to. `data_generator` will output 2 files containing binary, packed floating point values, and a JSON file describing the data's shape. The dependencies required to run `data_generator` are a Rust toolchain and Zeek.

`analysis` contains the neural network code, and the code used to generate this paper's figures. The only modification required to make this code work on an arbitrary dataset (generated by `data_generator`) are changing the hardcoded dataset paths. `analysis` requires Python 3, and has a list of Python dependencies in `requirements.txt`. In `requirements.txt`, `pytorchWithoutCuda` can be replaced with `pytorch` to utilize CUDA acceleration.

BIBLIOGRAPHY

[1] Alexa Internet, Inc.,. Keyword research, competitive analysis, & website ranking. `https://www.alexa.com`. Accessed: 2019-05-01.

[2] Aryan, S., Aryan, H., and Halderman, J. A. Internet censorship in iran: A first look. In *Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet*, Washington, D.C., 2013. USENIX.

[3] Baluja, S. and Fischer, I. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[4] Caruana, R., Lawrence, S., and Giles, C. L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[5] Chai, Z., Ghafari, A., and Houmansadr, A. On the importance of encrypted-sni (ESNI) to censorship circumvention. In *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, Santa Clara, CA, August 2019. USENIX Association.

[6] Choi, Y., Choi, M., Kim, M., Ha, J.-W., Kim, S., and Choo, J. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[7] Dalvi, N., Domingos, P., Sanghai, S., and Verma, D. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108, 2004.

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

[9] Dingledine, R. #2759 (proof of concept transport plugin: http headers) tor bug tracker & wiki. `https://trac.torproject.org/projects/tor/ticket/2759`, 2011.

[10] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[11] Dixon, L., Ristenpart, T., and Shrimpton, T. Network traffic obfuscation and automated internet censorship. *IEEE Security Privacy*, 14(6):43–53, Nov 2016.

[12] Ensafi, R., Fifield, D., Winter, P., Feamster, N., Weaver, N., and Paxson, V. Examining how the great firewall discovers hidden circumvention servers. In *Proceedings of the 2015 Internet Measurement Conference*, pages 445–458, 2015.

[13] Fifield, D. pluggable-transports/meek - https transport. `https://gitweb.torproject.org/pluggable-transports/meek.git/commit/?id=cea86c937dc278ba6b2100c238b1d5206bbae2f0`. Accessed: 2019-05-10.

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

[15] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[16] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[17] House, F. The rise of digital authoritarianism. freedom on the net 2018. https://freedomhouse.org/report/freedom-net/freedom-net-2018/rise-digital-authoritarianism, 2018.

[18] Inc., D. Docker — what is a container? https://www.docker.com/resources/what-container. Accessed: 2019-04-21.

[19] Inc., D. Docker compose — docker documentation. https://docs.docker.com/compose/. Accessed: 2019-07-05.

[20] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Luchaup, D., Dyer, K. P., Jha, S., Ristenpart, T., and Shrimpton, T. Libfte: A toolkit for constructing practical, format-abiding encryption schemes. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 877–891, San Diego, CA, August 2014. USENIX Association.

[23] Luo, X., Zhou, P., Chan, E. W., Lee, W., Chang, R. K., and Perdisci, R. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, volume 11. Citeseer, 2011.

[24] Marques, D. B. C. Learning from http/2 encrypted traffic: a machine learning-based analysis tool. 2018.

[25] Microsoft,. Microsoft ajax content delivery network — microsoft docs. https://docs.microsoft.com/en-us/aspnet/ajax/cdn/overview. Accessed: 2019-07-02.

[26] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108, 2012.

[27] Moore, A., Zuev, D., and Crogan, M. Discriminators for use in flow-based classification. Technical report, 2013.

[28] Nasr, M., Bahramali, A., and Houmansadr, A. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976. ACM, 2018.

[29] nickm,. #4744 (gfw probes based on tor's ssl cipher list) tor bug tracker & wiki. `https://trac.torproject.org/projects/tor/ticket/4744`, 2012.

[30] Paxson, V. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

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

[32] Project, T. doc/meek tor bug tracker & wiki. `https://trac.torproject.org/projects/tor/wiki/doc/meek`. Accessed: 2019-07-01.

[33] Project, T. Tor project: Pluggable transports. `https://2019.www.torproject.org/docs/pluggable-transports.html.en`. Accessed: 2019-04-20.

[34] Services, A. W. Aws regions and endpoints - amazon web services. `https://docs.aws.amazon.com/general/latest/gr/rande.html`. Accessed: 2019-07-05.

[35] Shahbar, K. and Zincir-Heywood, A. N. Traffic flow analysis of tor pluggable transports. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 178–181. IEEE, 2015.

[36] Sheffey, S. and Aderholdt, F. Improving meek with adversarial techniques. In *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*, Santa Clara, CA, August 2019. USENIX Association.

[37] Tschantz, M. C., Afroz, S., Paxson, V., and others,. Sok: Towards grounding censorship circumvention in empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 914–933. IEEE, 2016.

[38] Verma, G., Ciftcioglu, E., Sheatsley, R., Chan, K., and Scott, L. Network traffic obfuscation: An adversarial machine learning approach. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pages 1–6, Oct 2018.

[39] Wang, L., Dyer, K. P., Akella, A., Ristenpart, T., and Shrimpton, T. Seeing through network-protocol obfuscation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 57–69, New York, NY, USA, 2015. ACM.

[40] Winter, P. and Lindskog, S. How china is blocking tor. *arXiv preprint arXiv:1204.0447*, 2012.

[41] Winter, P., Pulls, T., and Fuss, J. 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, 2013.

[42] Wu, P. Dns encryption explained. `https://blog.cloudflare.com/dns-encryption-explained/`, 2019.

[43] Yao, Z., Ge, J., Wu, Y., Zhang, X., Li, Q., Zhang, L., and Zou, Z. Meek-based tor traffic identification with hidden markov model. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 335–340. IEEE, 2018.