5

TRAFFIC TYPE OBFUSCATION

If the only tool you have is a hammer, you tend to see every problem as a nail.
—*Abraham Maslow*

Traffic type obfuscation (TTO) is hiding the type of network traffic, that is, the underlying network protocol, exchanged between two (or multiple) endpoints. Suppose that two Internet endpoints Alice and Bob exchange Internet traffic using an Internet protocol P_1. Alice and Bob may choose to use traffic type obfuscation so that a third party, Eve, is not able to identify P_1 as the protocol used by Alice and Bob for their communication.

In order to be effective, a typical traffic type obfuscation scheme may need to modify one or several of the following traffic features:

- content of traffic, such as IP packet content;
- patterns of traffic, such as IP packet timings and sizes;
- protocol behavior, such as reaction to network perturbations.

Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures, First Edition. Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski.

5.1 PRELIMINARIES

5.1.1 Applications

Traffic type obfuscation techniques are utilized for two main applications.

- *Blocking resistance:* Traffic type obfuscation is used as a countermeasure in application scenarios where the use of particular types of network protocols is prohibited. For instance, some Internet ISPs disallow the access to P2P file sharing protocols such as BitTorrent,[1] and some enterprise networks block the use of Skype.[2] To enforce such blocking, ISPs use different networking mechanisms [1–3] to identify (and then block) the network traffic running the disallowed protocols. As another example, repressive regimes prohibit the use of systems that circumvent Internet censorship, for example, Tor [4]. Similar networking mechanisms are used by censorship technologies to detect the traffic generated by prohibited circumvention software [5–8].
- *Privacy protection:* Traffic type obfuscation is alternatively used to augment Internet users' privacy, for example, by manipulating traffic patterns to conceal the intent of communication. For instance, previous research [9–11] shows that an adversary can learn sensitive information about the Internet browsing activities of users (e.g., the websites they visit) by statistically analyzing their encrypted web traffic.

5.1.2 Comparison with Network Steganography

As noted in Chapter 1, traffic type obfuscation shares similarities with network steganography, the other category of information hiding in communication networks introduced in Chapter 3. Traffic type obfuscation resembles network steganography in that both aim to hide the *existence of a covert communication*. On the other hand, while the main objective of network steganography is to communicate the highest amount of covert information without being detected, the main objective of traffic type obfuscation is to conceal the *type* of network traffic. As a result, features such as bandwidth, robustness, and cost (which are used to evaluate network steganography) are out of scope for type obfuscation. Instead, a type obfuscation scheme should be *unobservable*: third-parties should not be able to (1) detect that the traffic has been obfuscated, and, in some applications, (2) disclose the obfuscated traffic type.

[1] www.bittorrent.com/.

[2] www.skype.com/.

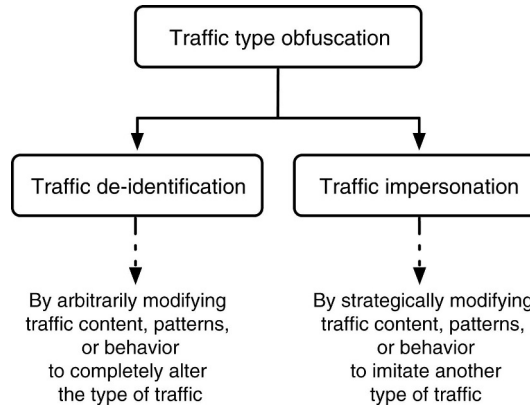

Figure 5.1. Classes of traffic type obfuscation based on the objective.

5.2 CLASSIFICATION BASED ON THE OBJECTIVE

The existing work on traffic type obfuscation can be broadly classified into two categories, as shown in Figure 5.1: *traffic de-identification* and *traffic impersonation*.

5.2.1 Traffic De-identification

Traffic de-identification manipulates network traffic so that the underlying network protocol is concealed. For instance, network packet headers are encrypted to conceal protocol identifier contents, thus obfuscating the underlying network protocol. In the following, some example approaches toward traffic de-identification are presented.

5.2.1.1 Padding Encrypted Packets. While encryption hides packet contents, traffic patterns, such as packet timings and packet sizes, can potentially reveal critical information about the content being communicated. For instance, previous work has shown that analyzing the patterns (e.g., packet sizes) of encrypted web traffic can give away the identities of the websites visited by users [11–14].

To counter this, several proposals have suggested to pad encrypted packets to conceal the actual lengths of plaintext packets. In this approach, already deployed in some implementations of the transport layer protocol (TLS), for example, GnuTLS,[3] random numbers of extra bytes are added at the end of packets before they are encrypted (see Figure 5.2). The SSH, TLS, and IPSec protocols (which are ''prevalent'' protocols for encrypting network traffic) allow up to 255 bytes of padding in their design. Previous work [9] discusses various approaches to pad the encrypted traffic of these protocols:

[3] http://www.gnu.org/software/gnutls/.

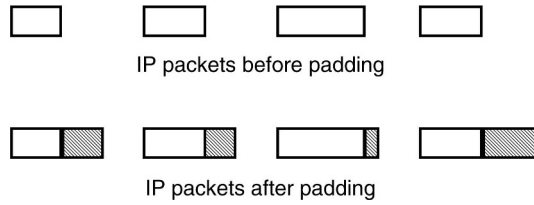

Figure 5.2. Padding network packets to de-identify packet sizes.

1. *Session random padding:* In this approach, a padding size, p, is randomly chosen from the set $\{0, 8, 16, ..., M - \ell\}$, where ℓ is the packet size before padding and M is the maximum transmission unit (MTU), that is, the size of the largest protocol data unit defined in the protocol's standard. For instance, the MTU is 1500 bytes for IEEE 802.3. Every packet in the target session is padded with a dummy plaintext of size p.
2. *Packet random padding:* In contrast to the session padding approach, every single packet is padded with a dummy content of a randomly chosen size, independently chosen across all of the packets in a session.
3. *Linear padding:* In this approach, packets are padded so that all packet sizes are increased to the nearest multiple of 128, or the MTU, whichever is smaller.
4. *Exponential padding:* All packets are padded to the nearest power of 2, or the MTU, whichever is smaller.
5. *Mice-Elephants padding:* The packets smaller than 128 bytes are padded to 128 bytes; all the other packets are padded to the MTU.
6. *Pad to MTU:* All packets are padded to the MTU size.

Note that some of these padding approaches are already deployed by some implementations of secure network protocols, such as GnuTLS.

5.2.1.2 HTTPOS. As noted earlier, traffic analysis can infer sensitive information from the communication patterns of encrypted traffic. In the context of web traffic, previous research [10–14] showed that traffic analysis can infer sensitive information about the browsing activities of a user by analyzing his/her encrypted web traffic, for example, HTTPS traffic (in this context, de-identification aims at masking the identities of HTTP destinations, not the HTTP protocol itself). Previous work has taken several directions to foil this, including the server-side techniques [15,16] that require modification of web entities, such as servers, browsers, and web objects, to deploy protocol de-identification. HTTPOS [17] is an alternative approach for safeguarding encrypted HTTP traffic against identification. HTTPOS deploys a browser-side approach; that is, obfuscation mechanisms are implemented at the client machine without the need to modify any web entity.

HTTPOS obfuscates traffic by modifying four fundamental features of TCP and HTTP protocols: packet sizes, timing of packets, sizes of web objects, and flow sizes.

Note that since HTTPOS works at the client side, it cannot directly modify such traffic features on the traffic from web servers. To be able to do so, HTTPOS exploits a number of basic features in the HTTP protocol, for example, HTTP pipelining and HTTP range, and TCP protocol, for example, maximum segment size negotiation and advertised window.

5.2.2 Traffic Impersonation

As introduced earlier, the aim of traffic impersonation is to not only hide the underlying network protocol, but also pretend to be using another type of protocol, that is, a target protocol. For instance, a BitTorrent client may modify her traffic to look like it is carrying HTTP traffic.

5.2.2.1 SkypeMorph. Tor [4] is a popular system used by Internet users living under repressive regimes to bypass Internet censorship. Unfortunately, today's modern networking technology allows state-level censors, that is, totalitarian governments, to identify and block the use of Tor's network protocol. This is done in different ways, such as statistical analysis of Tor traffic patterns [18] and looking for Tor protocol-specific identifiers [19, Slide 38]. To be usable, it is of paramount importance for Tor to deploy mechanisms that conceal the use of the Tor protocol at the network layer. *Pluggable transports* [20] are Tor's recent initiative toward protecting the identity of Tor traffic (and Tor parties) from the censors.

SkypeMorph [21] is a pluggable transport for Tor. It intends to conceal Tor's traffic by making the traffic between a Tor client and a Tor server look like a Skype video call. That is, a Tor user running SkypeMorph software *mimics* Skype's network protocol.

Figure 5.3 illustrates SkypeMorph's architecture. The following summarizes how SkypeMorph works to mimic Skype:

1. The SkypeMorph server creates a legitimate Skype ID, and shares the ID with the SkypeMorph client. The server, then, logs in to Skype, and listens for incoming Skype calls.
2. The SkypeMorph client similarly creates a legitimate Skype ID.

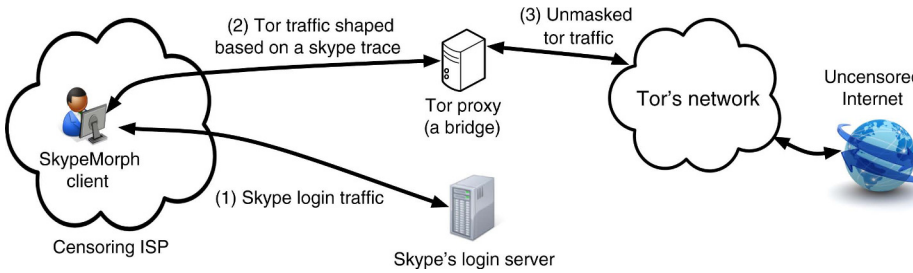

Figure 5.3. The main architecture of SkypeMorph [21].

3. The client checks to see whether the server's Skype ID is online. If so, the client uses Skype's text messaging platform to share some cryptographic keys with the server. The server and client send a few other Skype text messages to finish this initial handshake, which results in sharing cryptographic keys and other secret information needed for the operation of SkypeMorph.
4. The SkypeMorph client starts a Skype video call with the server's Skype ID. To do so, it uses Skype's API to send a ''ringing'' message to the server for a random amount of time.
5. Both the client and the server exit Skype's API. They establish a Tor connection; however, using the cryptographic keys derived in previous steps, they encrypt Tor packets to hide Tor's protocol. Additionally, they *shape* the Tor traffic; that is, they use network traces of Skype traffic to mimic the timings and packet sizes of a previously established Skype conversation.

Analysis of SkypeMorph [21] demonstrates that the traffic patterns (e.g., packet size distribution and interpacket delay distribution) of a Tor connection over SkpeMorph resemble those of Skype traffic, not Tor.

5.2.2.2 FreeWave. FreeWave [22] is a blocking-resistant system that aims at hiding the use of a disallowed network protocol. For instance, it can be deployed by Tor entry points as a Tor pluggable transport [20] in order to disguise Tor traffic, or can be used by BitTorrent clients to obfuscate their file sharing traffic. Similar to SkypeMorph, introduced above, FreeWave mimics the popular Voice-over-IP (VoIP) network protocol of Skype. On the other hand, contrary to SkypeMorph that shapes Tor traffic to look like Skype, FreeWave runs the actual Skype protocol for the whole duration of a connection, and encodes the source traffic (e.g., Tor traffic) into the acoustic signals carried over the Skype protocol. Figure 5.4 illustrates the architecture of FreeWave.

The following are the main components of FreeWave client and server software:

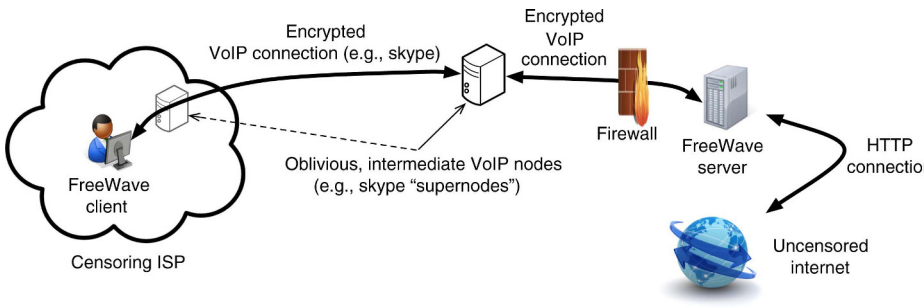

Figure 5.4. The main components of FreeWave [22].

1. *VoIP client:* VoIP client is a Voice-over-IP client software, such as Skype software, that allows VoIP users to connect to one (or more) specific VoIP service(s).
2. *Virtual sound card (VSC):* A virtual sound card (VSC) is a software application that uses a physical sound card installed on a machine to generate one (or more) isolated, virtual sound card interfaces on that machine. A virtual sound card interface can be used by any application running on the host machine exactly the same way a physical sound card is utilized. Also, the audio captured or played by a virtual sound card does not interfere with that of other physical/virtual sound interfaces installed on the same machine. FreeWave uses VSCs to isolate the audio signals generated by its software from the audio belonging to other applications.
3. *MoDem:* FreeWave client and server software use a modulator/demodulator (MoDem) application that translates network traffic into acoustic signals and vice versa. This allows FreeWave to tunnel the network traffic of its clients over VoIP connections by modulating them into acoustic signals. Houmansadr et al. [22] use communication codes to design a reliable MoDem that withstands noise from the network and Skype codec.
4. *Proxy:* The FreeWave server uses an ordinary network proxy application that proxies the network traffic of FreeWave clients, received over VoIP connections, to their final Internet destinations. Two popular choices for FreeWave's proxy are the HTTP proxy [23] and the SOCKS proxy [24].

FREEWAVE'S CLIENT DESIGN. The FreeWave client software consists of three main components described above: a VoIP client application, a virtual sound card, and the MoDem software. Figure 5.5 shows the block diagram of the FreeWave client design. The MoDem transforms the data of the network connections sent by the web browser into acoustic signals and sends them over to the VSC component. The FreeWave MoDem also listens on the VSC sound card to receive specially formatted acoustic signals that carry modulated Internet traffic; MoDem extracts the modulated Internet traffic from such acoustic signals and sends them to the web browser. In a sense, the client web browser uses the MoDem component as a network proxy; that is, the listening port of MoDem is entered in the HTTP/SOCKS proxy settings of the browser.

The VSC sound card acts as a bridge between MoDem and the VoIP client component; that is, it transfers audio signals between them. More specifically, the VoIP client is set up to use the VSC sound card as its ''speaker'' and ''microphone'' devices (VoIP

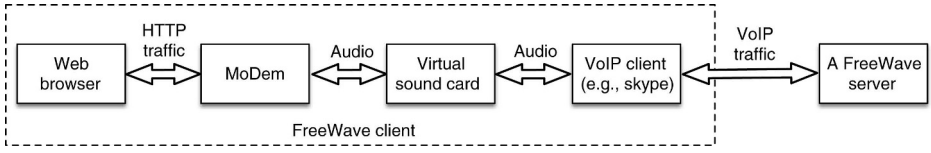

Figure 5.5. The main components of FreeWave [22] client.

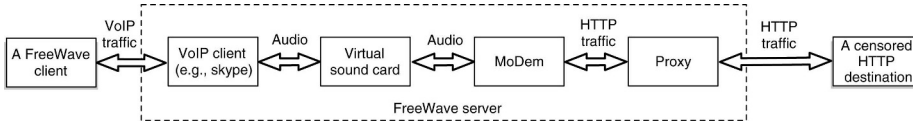

Figure 5.6. The main components of FreeWave [22] server.

applications allow a user to select physical/virtual sound cards). This allows MoDem and the VoIP client to exchange audio signals that contain the modulated network traffic, isolated from the audio generated/recorded by other applications on the client machine.

For the FreeWave client to connect to a particular FreeWave server, it requires to know the VoIP ID belonging to that server. Every time the user starts up the FreeWave client application on his/her machine, the VoIP application of FreeWave client initiates an audio/video VoIP call to the known VoIP ID of the FreeWave server.

FREEWAVE'S SERVER DESIGN. Figure 5.6 shows the design of FreeWave server, which consists of four main elements. FreeWave server uses a VoIP client application to communicate with its clients through VoIP connections. A FreeWave server chooses one or more VoIP IDs, which are provided to its clients, for example, through public advertisement.

The VoIP client of the FreeWave server uses one (or more) virtual sound card(s) as its ''speaker'' and ''microphone'' devices. The VSCs are also used by the MoDem component, which transforms network traffic into acoustic signals and vice versa. More specifically, MoDem extracts the Internet traffic modulated by FreeWave clients into audio signals from the incoming VoIP connections and forwards them to the last element of the FreeWave server, FreeWave proxy. MoDem also modulates the Internet traffic received from the proxy component into acoustic signals and sends them to the VoIP client software through the VSC interface.

5.3 CLASSIFICATION BASED ON THE IMPLEMENTATION DOMAIN

Traffic type obfuscation techniques are classified into several categories based on the domain they use to deploy obfuscation mechanisms, as shown in Figure 5.7.

5.3.1 Content-Based Schemes

Modern state-of-the-art network firewalls are equipped with deep-packet inspection (DPI) technologies, which allows them to deeply scan network traffic. Many open-source DPI systems (e.g., Bro [25], Yaf [26], and nProbe [27]) use regular expressions (regexes) to classify network protocols. DPIs look for content signatures that identify a particular network protocol; hence, they can be used to identify and block specific network protocols, for example, Skype and Tor protocols.

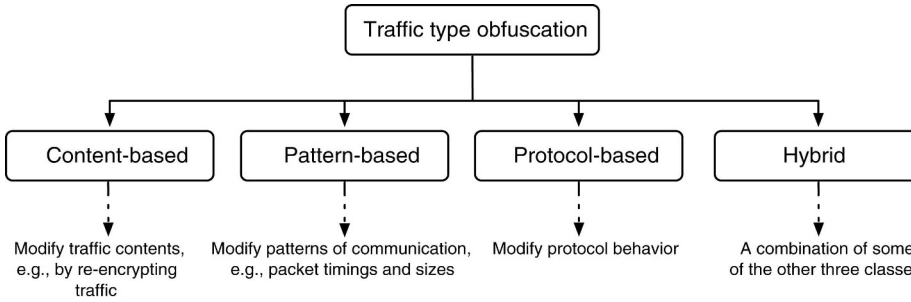

Figure 5.7. Classes of traffic type obfuscation based on the implementation domain.

Content-based traffic type obfuscation schemes aim at defeating DPI technologies by manipulating packet contents (e.g., using encryption) in order to remove content signatures that reveal the use of a particular network protocol. In the following, we introduce several content-based protocol obfuscators.

5.3.1.1 Tor Content Obfuscation. In order to improve its undetectability against competent state-level censors, the Tor project has initiated the pluggable transports project. Pluggable transports aim at hiding the very existence of the Tor protocol by deploying various protocol obfuscation mechanisms, including content-based, pattern-based, and protocol-based techniques.

Obfsproxy [28] is the first Tor pluggable transport, which is a content-based obfuscation mechanism. Figure 5.8 shows the main block diagram of Obfsproxy. Obfsproxy works by re-encrypting Tor packets in order to conceal Tor-specific content signatures in Tor packets.

Obfsproxy is the basis of many subsequent pluggable transport designs, including SkypeMorph (Section 5.2.2.1), StegoTorus (Section 5.3.4.1), and FTE (Section 5.3.1.2). As another content-based example for pluggable transport, Dust [29] aims at defeating protocol fingerprinting through deep-packet inspection by transforming the contents of

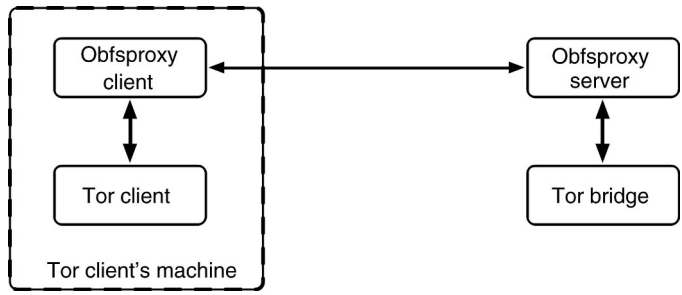

Figure 5.8. The main architecture of Obfsproxy.

Tor traffic into encrypted or random single-use bytes that are indistinguishable from each other and random packets.

5.3.1.2 Format-Transforming Encryption. Dyer et al. [30] investigate mechanisms that allow an attacker to enforce content-based protocol misidentification against DPI mechanisms that rely on regular expressions. They propose the use of a new cryptographic primitive, named format-transforming encryption (FTE), which allows a network entity to produce encrypted content guaranteeing that they match a specific regular expression. In other words, a network entity (e.g., a censored Internet user) can use FTE to transform the regexes of packet contents pertaining to a particular network protocol (the source protocol) into the regexes of another protocol (the target protocol).

To be more specific, FTE is an encryption scheme that takes a regular expression as an input, and encrypts an arbitrary plaintext so that the generated ciphertext matches the input regular expression. Dyer et al. use FTE as a component within the TLS record layer to handle streams of messages from arbitrary source protocols.

FTE, which is inspired by the format-preserving encryption (FPE) [31], works in the following steps:

1. The plaintext message M is encrypted using a standard authenticated encryption scheme, resulting in an intermediate ciphertext Y.
2. Y is treated as an integer in $\mathbb{Z}_S$, which is the set of integers from 0 to the size of the language minus one. S is the set of expressions in the input language.
3. Y is encoded using function $unrank : \mathbb{Z}_S \rightarrow S$, generating the final ciphertext C. To decrypt, there needs to exist a reverse function for $unrank$, that is, $rank : S \rightarrow \mathbb{Z}_S$, that is efficiently computable.

Dyer et al. demonstrate that FTE can make several real-world DPI systems misidentify network protocols at the cost of small bandwidth overhead.

5.3.2 Pattern-Based Schemes

Previous work [9,16,32–34] demonstrates the feasibility of protocol classification by analyzing traffic patterns, that is, traffic analysis. Pattern-based traffic obfuscation schemes aim at concealing the type of traffic by perturbing patterns of network traffic, such as packet timings and packet sizes.

In Section 5.2.1.1, we described per-packet padding mechanisms that aim at modifying packet sizes in order to de-identify the underlying network protocol. Wright et al. [16] propose two approaches that augment such per-packet padding mechanisms by not only concealing the type of the underlying network protocol, but also mimicking that of another network protocol, that is, a target protocol. Note that, instead of mimicking patterns of another protocol, such mechanisms can be used in the same manner to mimic another instance of the same protocol. For example, a pattern-based obfuscation scheme can be used to disguise HTTP traffic with website A as HTTP traffic with another website, B.

5.3.2.1 Direct Target Sampling. In this approach, network packets are padded with arbitrary content (e.g., random bits) in order to change the sizes of network packets whose pattern may potentially reveal the underlying protocol. The following is how the DTS protocol works. Consider a source protocol A and a target protocol B (or, alternatively, a source and target instance of the same protocol). Also, assume D_A and D_B as the respective probability distributions of their packet sizes. The direct target sampling (DTS) scheme works as follows:

1. For any packet of size i from the source protocol, the DTS scheme derives a packet size, i', from D_B at random.
2. If $i' \geq i$, DTS pads the input packet from A to size i', and sends the packet.
3. If $i' < i$, DTS sends the first i' bytes of the input packet as a single network packet. DTS, then, goes to step 1; that is, it continues sampling from D_B until all bytes of the original input packet are sent.

5.3.2.2 Traffic Morphing. Traffic morphing works in a similar manner to direct target sampling, except it uses a different sampling mechanism. Contrary to direct target sampling, which directly samples from the target protocol (D_B), traffic morphing uses convex optimization methods to sample. More specifically, traffic morphing uses convex optimization to generate a *morphing matrix* that makes source traffic look like the target protocol while minimizing the overhead, for example, the number of packets sent.

Consider the source packet sizes as a vector of size n, $S_A = [x_1, x_2, ..., x_n]^T$ (T is the transpose operation). The output packet sizes are derived as

$$S_B = A \cdot S_A, \tag{5.1}$$

where $S_B = [y_1, y_2, ..., y_n]^T$. A is the morphing matrix of size $n \times n$, that is, $A = [a_{ij}]$ $(1 \leq i, j \leq n)$. To morph A to B, each column of the morphing matrix A should be a valid probability mass function over the n packet sizes. For any input packet with size s_j, the morphing algorithm looks at the j^{th} column of A and samples a target size s_i using the probability distribution of that column. Similar to the case of DTS, if the derived target size is larger than the input size, the input packet is padded and sent. Otherwise, the input packet is split into multiple packets recursively running the morphing algorithm.

The morphing matrix is derived using convex optimization so that it minimizes a cost function, $f_0(A)$, subject to multiple constraints. The constraints in this case are

$$\sum_{j=1}^{n} a_{ij} x_j = y_i, \qquad \forall i \in [1, n], \tag{5.2}$$

$$\sum_{i=1}^{n} a_{ij} = 1, \qquad \forall j \in [1, n], \tag{5.3}$$

$$a_{ij} \geq 0, \qquad \forall i, j \in [1, n]. \tag{5.4}$$

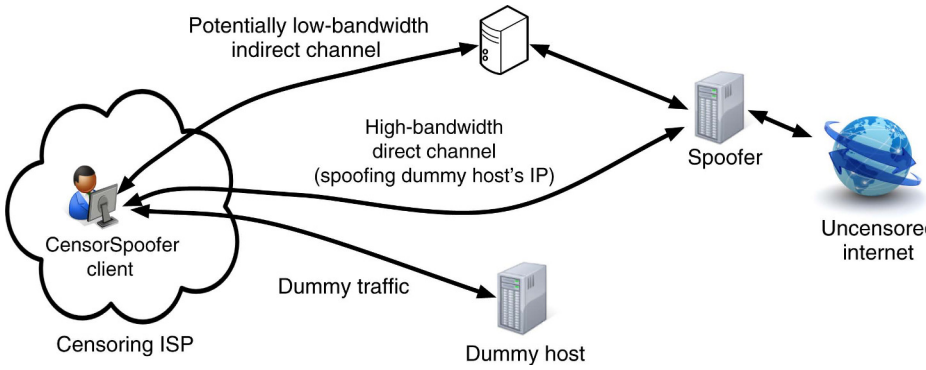

Figure 5.9. The main architecture of CensorSpoofer [36].

5.3.3 Protocol-Based Schemes

The third class of protocol type obfuscation includes protocol-based schemes. These techniques [21,35,36] modify the protocol behavior such as the handshaking mechanism and subprotocol dependencies in order to obfuscate protocol types. In the following, two protocol-based obfuscation schemes are described.

5.3.3.1 CensorSpoofer. CensorSpoofer is a system for blocking-resistant web browsing. It obfuscates the web protocol as a P2P protocol such as VoIP. As CensorSpoofer's source protocol is HTTP, the upstream flow (requested URLs) requires much less bandwidth than the downstream flow (potentially large HTTP responses). Consequently, CensorSpoofer decouples the upstream and downstream flows.

Figure 5.9 shows CensorSpoofer's architecture. A CensorSpoofer client (e.g., a blocked web user) utilizes a low-capacity channel such as email or instant messaging to send HTTP requests to the server (e.g., an HTTP proxy). On the other hand, the server mimics P2P traffic from an oblivious dummy host in order to obfuscate HTTP responses. The server chooses the dummy hosts through random port scanning of Internet IPs.

CensorSpoofer's prototype uses a SIP-based VoIP protocol as the target protocol for obfuscation. A CensorSpoofer client initiates a SIP connection with the CensorSpoofer server by sending a SIP INVITE to the appropriate SIP ID. The CensorSpoofer spoofer replies with a SIP OK message spoofed to look as if its origin is the dummy host. Once the client receives this message, it starts sending encrypted RTP/RTCP packets with random content to the dummy host. At the same time, the spoofer starts sending spoofed, encrypted RTP/RTCP packets to the client ostensibly from the dummy host's address.

To browse a URL, the client sends it through the upstream channel. The spoofer fetches the contents and embeds them in the spoofed RTP packets to the client. To terminate, the client sends a termination signal upstream. The spoofer replies with a spoofed SIP BYE message, and the client sends a SIP OK message and closes the call.

5.3.4 Hybrid Schemes

Hybrid schemes combine two or all three main mechanisms of content-based, pattern-based, and protocol-based obfuscation in order to provide a more resilient protocol type obfuscation. Many of the protocol obfuscation proposals, including some of the example schemes introduced before, are hybrid schemes, while others perform only one obfuscation mechanism. For instance, SkypeMorph [21] (Section 5.2.2.1) performs all content-based, pattern-based, and protocol-based mechanisms, while Obfsproxy [28] (Section 5.3.1.1) deploys only content-based obfuscation. In the following, we introduce StegoTorus [35], which is a hybrid protocol obfuscation scheme.

5.3.4.1 StegoTorus. Similar to SkypeMorph introduced in Section 5.2.2.1, StegoTorus [35] is a pluggable transport [20] for Tor, which is derived from Obfsproxy [28]. That is, StegoTorus aims at making Tor connections, established between Tor clients and Tor relays, look like other network protocols.

StegoTorus is composed of two main techniques: *chopping* and *steganography*. The chopper performs content-based obfuscation (as introduced in Section 5.3.1); that is, its goal is to foil statistical traffic analysis by modifying packet sizes and timings. The chopper carries Tor traffic over *links* comprised of multiple connections, where each connection is a sequence of blocks, padded and delivered out of order. The second main component of StegoTorus is the steganography module, whose goal is to mimic popular network protocols such as HTTP, Skype, and Ventrilo. That is, the steganography module is responsible for protocol-based obfuscation.

The StegoTorus project [35] implements two variations of the steganography module.

- *Embed steganography:* StegoTorus-Embed aims to mimic a P2P protocol such as Skype or Ventrilo VoIP. It takes as input a database of genuine, previously collected Skype or Ventrilo traffic, and then uses that for shaping the Tor traffic of Tor users. More specifically, embed steganography modifies the timing and sizes of Tor packets based on the timing and size information of the collected network traffic. The Embed module additionally emulates the headers of the target protocols (i.e., content-based obfuscation as introduced in Section 5.3.1), for example, adds required packet headings.
- *HTTP steganography:* The StegoTorus-HTTP module mimics unencrypted HTTP traffic by simulating an HTTP *request generator* and an HTTP *response generator*. These simulators operate by using a pre-recorded trace of HTTP requests and responses collected over actual HTTP sessions. The request generator picks a random HTTP GET request from the trace and hides the payload produced by the chopper in the `<uri>` and `<cookie>` fields of the request by encoding the payload into a modified `base64` alphabet and inserting special characters at random positions to make it look like a legitimate URI or cookie header. The response generator picks a random HTTP response consistent with the request and hides the data in the chapters carried by this response. The StegoTorus implementation uses PDF, SWF, and JavaScript chapters for this purpose.

5.4 COUNTERMEASURES

To foil traffic type obfuscation schemes, various countermeasures [9,37,38] have been proposed that seek one of the following two objectives:

1. Detect the utilization of traffic type obfuscation mechanisms. For instance, state-level censors are interested in detecting the use of censorship resistance systems by their Internet users in order to block users' Internet access.
2. In addition to detecting the use of type obfuscation mechanisms (previous goal), identify the types of obfuscated network protocols. For instance, ISPs are interested in identifying and warning Internet subscribers who use BitTorrent file sharing protocol [3].

Earlier in this chapter (Section 5.3), we classified traffic obfuscation techniques into the three main classes of content-based, pattern-based, and protocol-based schemes. Accordingly, we classify countermeasures to traffic obfuscation techniques into the three categories of *content-based countermeasures*, *pattern-based countermeasures*, and *protocol-based countermeasures* (see Figure 5.10).

5.4.1 Content-Based Countermeasures

Content-based countermeasures aim at detecting traffic obfuscation (and possibly the type of the obfuscated protocol) by looking for content discrepancies in the target traffic. Two examples of content-based countermeasures are provided below.

5.4.1.1 Missing Skype Headers in Skype Imitators. Previously, we introduced SkypeMorph (Section 5.2.2.1) and StegoTorus (Section 5.3.4.1) as two protocol type obfuscation mechanisms that aim at mimicking Skype; that is, they make their network traffic look like Skype. Houmansadr et al. [37] show that these schemes can be identified by using content-based countermeasures. More specifically, Skype uses

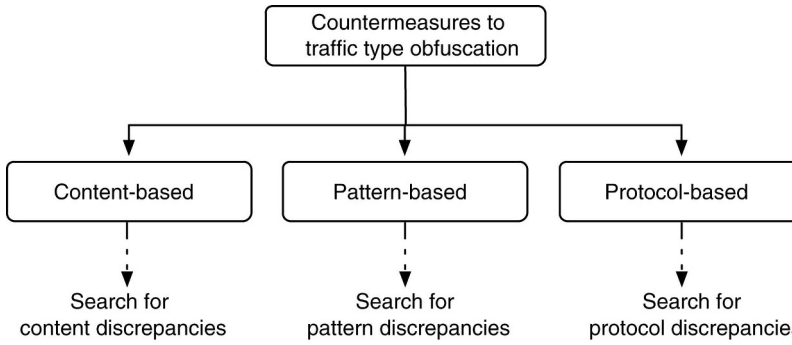

Figure 5.10. Countermeasures to traffic type obfuscation.

special headers, called the start of message (SoM), for its UDP packets [2]. The SoM headers are present in all UDP packets generated by the Skype protocol, and they appear in plaintext, that is, are not encrypted.

Despite Skype being a proprietary protocol, previous work [2,39] has taken steps toward reverse engineering it. In particular, the SoM headers contain two easily recognizable fields of *ID* and *Fun* [2]. *ID*, which uses the first two bytes of the SoM header, uniquely identifies a Skype message. This value is randomly generated by the sender and copied in the receiver's reply. *Fun* is a 5-bit field obfuscated into the third byte of SoM and revealed by applying the `0x8f` bitmask. Previous research [2,39] investigated the values of Fun for different messages. For instance, `0x02`, `0x03`, `0x07`, and `0x0f` indicate signaling messages during the login process and connection management, while `0x0d` indicates a data message, which may contain encoded voice or video blocks, chat messages, or data transfer chunks.

Houmansadr et al. [37] show that both SkypeMorph and StegoTorus obfuscation systems ignore the SoM header in their design. That is, the Skype-like UDP packets generated by SkypeMorph and StegoTorus entirely miss an SoM header, enabling countermeasures that can easily distinguish between imitated and genuine Skype traffic.

5.4.1.2 Countermeasures Using File-Format Semantics. As described in Section 5.3.4.1, StegoTorus's ''HTTP steganography'' module embeds the contents of covert traffic into cover chapters of types PDF, SWF, or JavaScript. StegoTorus does so by using real chapters and replacing specific fields with hidden content. This preserves the file's syntactic validity, but not its semantics. Houmansadr et al. [37] show that these cover chapters can be identified using low-cost content classifiers that use file-format semantics.

In particular, Houmansadr et al. show how to analyze the PDF chapters carrying hidden StegoTorus traffic. StegoTorus uses a fake trace-generator to produce templates for PDF chapters; the generator misses an essential object called *xref table*. In a genuine PDF file, this table follows the `xref` keyword and declares the number and position of various objects. The absence of this table in StegoTorus's imitations is detectable via simple deep-packet inspection at line speed, without any need to reconstruct or parse the file.

Houmansadr et al. additionally demonstrate that adding a fake `xref` table to the PDF file does not fix the detectability issue. This is because there are various software scripts available that are able to check the validity (or invalidity) of a PDF file's `xref` table at low computation cost, with no need of rendering the PDF file. Even if a non-fake `xref` table is generated and used by StegoTorus, there are simple, fast scripts (e.g., the Unix `pdftotext` command) that are able to extract the encoded text from PDF files, which then can be validated through natural language processing techniques.

5.4.2 Pattern-Based Countermeasures

Recent studies [9,32–34] show that many existing pattern-based obfuscation techniques have high overhead, are easily defeatable, or both. In particular, Dyer et al. [9] and Cai et al. [32] evaluate numerous website fingerprinting defenses (including HTTPOS

introduced in Section 5.2.1.2 and traffic morphing described in Section 5.3.2.2) and demonstrate the feasibility of countermeasures against them. In the following, we present several example countermeasures.

5.4.2.1 Damerau–Levenshtein Distance. The Damerau–Levenshtein distance is an information theory tool to measure the distance between two strings, that is, finite sequences of symbols. The Damerau–Levenshtein distance counts the number of operations needed to transform one string to the other, where each operation performs one of the four main tasks of insertion, deletion, substitution of a single character, or transposition of two adjacent characters.

Cai et al. [32] use the Damerau–Levenshtein distance in order to counter several pattern-based obfuscation mechanisms including HTTPOS (Section 5.2.1.2), traffic morphing (Section 5.3.2.2), and padding-based mechanisms (Section 5.2.1.1) such as the randomized pipelining deployed by Tor [40]. They argue that the studied pattern-based obfuscation defenses against traffic analysis are based on ad-hoc heuristics and are likely to fail. Cai et al. additionally use hidden Markov models to extend their webpage classifier into a website classifier.

5.4.2.2 Naïve Bayes Classifier. Naïve Bayes (NB) is a probabilistic machine learning classifier that applies Bayes' theorem with (naively) strong independence assumptions between the features. Specifically, for a given *feature vector* $Y = (Y_1, Y_2, ..., Y_n)$ of size n, the Bayes theorem returns the conditional probability of $\Pr(\ell_i|Y)$ to be

$$\Pr(\ell_i|Y) = \frac{\Pr(Y|\ell_i)\Pr(\ell_i)}{\Pr(Y)}, \tag{5.5}$$

where ℓ_i is the ith hypothesis out of a set of possible outcomes with size k. The Bayes rule computes $\Pr(\ell_i|Y)$ for each of the $i = \{1, 2, ..., k\}$ potential outcomes and projects the hypothesis with the largest value of $\Pr(\ell_i|Y)$ as the estimated outcome. The naïve form of Bayes rule (NB) assumes that any of the two features Y_s and Y_p of the feature vector Y are independent (for $1 \leq s, p \leq n$ and $s \neq p$), simplifying the Bayes rule to

$$\Pr(\ell_i|Y) = \frac{\Pr(\ell_i)}{\Pr(Y)} \prod_{t=1}^{n} \Pr(Y_t|\ell_i). \tag{5.6}$$

Liberatore and Levine [11] use a naïve Bayes classifier to identify the web destinations browsed over encrypted HTTPS connections. They use the counts of the lengths of the packets sent in each direction of an encrypted HTTPS connection as the feature vector Y. Given that there are 1449 possible packet lengths for an HTTPS packet in each direction, the size of this feature vector is $2 \times 1449 = 2898$. Liberatore and Levine assume equal probability for HTTPS destination, that is, $\Pr(\ell_i) = 1/k$ (k is the number of all possible HTTPS destinations), and use kernel density estimation over the example feature vector to estimate $\Pr(Y|\ell_i)$. The devised algorithm is able to identify the source of encrypted HTTP connections between 60% and 90% of the time, for a bounded set of sources.

Similarly, Herrmann et al. [10] use a *multinominal* naïve Bayes classifier to identify web destinations browsed over encrypted connections. The multinominal naïve Bayes classifier works similar to the naïve Bayes classifier described above, except that it uses the aggregated frequency of the features across all training vectors. That is, the features are evaluated using a normalized counting of the packet sizes as opposed to a raw counting.

5.4.2.3 Support Vector Machine Classifier. A support vector machine (SVM) [41] is a supervised machine learning algorithm that can be used as a classifier for observed data. Given a set of training data points each tagged as belonging to one of the two possible categories, SVM represents each of the examples as points in a high-dimensional space and derives a hyperplane that maximally separates the points belonging to the two categories.

Panchenko et al. [42] suggest the use of SVM classifiers to identify websites browsed over encrypted anonymity systems such as Tor [4]. Their proposed classifier utilizes a wide range of features of network flows pertaining to the volume, time, and direction of the traffic. This includes fine-grained features such as counts of packet lengths, and coarse-grained features such as the total number of transmitted packets. All of the used features are rounded and all TCP acknowledgment packets are removed to minimize noise during the training. The proposed scheme achieves a surprisingly high true detection rate of around 73% for a very small false negative rate.

5.4.2.4 Standard Deviation. FreeWave, introduced in Section 5.2.2.2, obfuscates the underlying network protocol by encoding cover messages into audio signals and sending them over encrypted VoIP connections. Geddes et al. [38] show that FreeWave connections build over Skype can be distinguished from genuine Skype calls by performing statistical traffic analysis. Such countermeasures stem from the fact that Skype's audio codec, SILK [43], is a variable-bit length codec, which makes the sizes and timings of Skype packets depend on the spoken conversation. Geddes et al. particularly show that the standard deviation of packet lengths can distinguish a genuine Skype connection from one carrying FreeWave's data-modulated audio.

Note that such attacks were acknowledged in the original FreeWave paper [22], where they proposed three countermeasures.

1. To deploy FreeWave on a VoIP protocol that, unlike Skype, uses a fixed-bit rate codec (e.g., the widely used G.7xx[4] codec series). This will make traffic patterns independent of the underlying conversation.
2. To superimpose the modulated audio of FreeWave with pre-recorded human conversations.
3. To tunnel cover traffic in the video content of a VoIP conversation as a video has less dependence on the underlying content compared with audio.

[4] http://www.voip-info.org/wiki/view/Codecs.

TABLE 5.1. Passive protocol-based countermeasures to detect imitators of the Skype protocol.

Countermeasure	SkypeMorph	StegoTorus
Skype HTTP update traffic	Not defeated	Defeated
Skype login traffic	Not defeated	Defeated
Periodic message exchanges	Defeated	Defeated
Typical Skype client behavior	Defeated	Defeated
TCP control channel	Defeated	Defeated

Reproduced from [24] with permission of IEEE.

5.4.3 Protocol-Based Countermeasures

Protocol-based countermeasures [37,38] look for abnormalities in the traffic imitated by a protocol-based obfuscation scheme. They include *passive* countermeasures, which passively monitor network traffic for discrepancies, and *active* countermeasures, which perturb suspected traffic looking for revealing behavior.

5.4.3.1 Passive Countermeasure Examples. SkypeMorph (Section 5.2.2.1) and StegoTorus (Section 5.3.4.1) obfuscate their traffic by imitating Skype's protocol. Despite the Skype protocol being proprietary, it has been reverse engineered in previous work [2,39,44], resulting in various Skype identification tests [2,39] that are used by ISPs and enterprise networks to detect (and sometimes block) the use of Skype. Houmansadr et al. [37] show that such tests can be used as countermeasures to identify protocol-based obfuscations schemes that—incompletely—imitate Skype. Table 5.1 shows some of the tests that can be used to identify SkypeMorph and StegoTorus.

Houmansadr et al. [37] argue that one can combine the tests listed in Table 5.1 into a hierarchical detection tool to build more advanced countermeasures. In fact, similar tools have been proposed for real-time detection of Skype traffic [45,46], including line-rate detectors by Pláček [39], who used these tests in an NfSen[5] plug-in, and by Adami et al. [8].

5.4.3.2 Active Countermeasure Examples. In the following, we describe several active countermeasures proposed by Houmansadr et al. [37] that target protocol-based obfuscation systems that mimic Skype.

MANIPULATING SKYPE CALLS. This countermeasure tampers with a Skype connection by dropping, reordering, and delaying packets or modifying their contents, and then observes the endpoints' reaction. These changes are fairly mild and can occur naturally; thus, they do not drastically affect genuine Skype connections.

When UDP packets are dropped in a genuine Skype call, there is an immediate, very noticeable increase in the activity on the TCP control channel that accompanies the

[5] http://nfsen.sourceforge.net/.

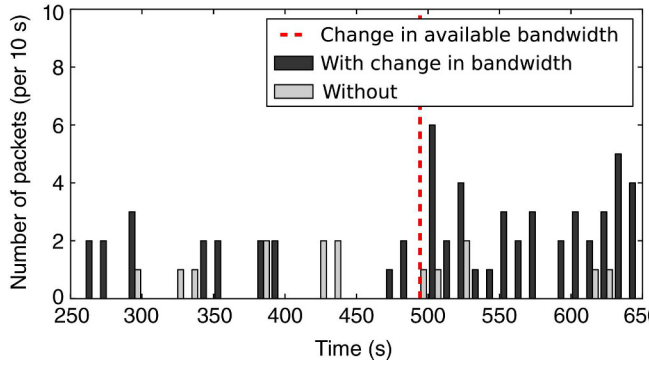

Figure 5.11. Skype TCP activity with and without changes in bandwidth. (Reproduced from [37] with permission of IEEE.)

main UDP connection (see Figure 5.11). Houmansadr et al. conjecture that this is caused by Skype endpoints renegotiating connection parameters due to perceived changes in network conditions.

Houmansadr et al. argue that it is *extremely* difficult for an imitator of Skype to convincingly imitate dynamic dependencies between network conditions and Skype's control traffic; hence, this can be used as a countermeasure to detect such protocol-based obfuscation mechanisms. This active countermeasure does not adversely affect normal Skype users. Dropping a few packets does not disconnect the call, but only degrades its quality for a short period of time. Geddes et al. [38] propose similar countermeasures that aim at disrupting (as opposed to identifying) Skype's protocol-based obfuscation systems by strategically dropping Skype packets.

MANIPULATING THE TCP CONTROL CHANNEL OF SKYPE. The countermeasures introduced above assert that perturbing Skype's main UDP connection causes observable changes in the TCP control channel. Alternatively, Houmansadr et al. [37] show that perturbing the TCP channel of Skype can cause observable changes in the UDP connection, which could be used to identify imitations of Skype.

1. *Closing the TCP connection:* Closing the TCP channel (e.g., by sending an RST packet) causes genuine Skype nodes to immediately end the call. A typical imitator of Skype is likely not to mimic this behavior because their fake TCP channel may have no relationship to the actual call.
2. *Withholding or dropping selected TCP packets:* The TCP connection of Skype sends a packet every 30–60 s, or when network conditions change. Tampering with these packets causes observable changes in the genuine UDP channel, but not on an imitated one.
3. *Triggering a supernode probe:* A Skype client keeps a TCP connection with its supernode [2]. If this connection is closed, a genuine client immediately launches a UDP probe [44] to search for new supernodes.

4. *Blocking a supernode port:* After a successful UDP probe, a genuine client establishes a TCP connection with the same port of its supernode. If this port is not available, the client tries connecting to port 80 or 443 [44].

5.5 SUMMARY

Traffic type obfuscation is an increasingly important type of information hiding in communication networks. In contrast to network steganography, traffic type obfuscation hides the *type* of the network traffic exchanged between network entities, that is, the underlying network protocols. Traffic type obfuscation techniques do this by modifying contents of traffic, patterns of traffic, or the behavior of network protocols. They are utilized mainly for blocking resistance and privacy protection.

REFERENCES

1. Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, and P.-C. Lin. Application classification using packet size distribution and port association. *Journal of Network and Computer Applications*, 32(5):1023–1030, 2009.
2. D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing Skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review*, 37(4):37–48, 2007.
3. K. Bauer, D. McCoy, D. Grunwald, and D. Sicker. BitStalker: accurately and efficiently monitoring BitTorrent traffic. In *1st IEEE International Workshop on Information Forensics and Security (WIFS 2009)*, pp. 181–185. IEEE, 2009.
4. R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *USENIX Security*, 2004.
5. P. Winter and S. Lindskog. How the Great Firewall of China is blocking Tor. In *FOCI*, 2012.
6. T. Wilde. Knock knock knockin' on bridges' doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors, 2012.
7. Ten ways to discover Tor bridges. https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges.
8. D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe. Skype-Hunter: a real-time system for the detection and classification of Skype traffic. *International Journal of Communication Systems*, 25(3):386–403, 2012.
9. K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, I still see you: why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy (SP)*, pp. 332–346. IEEE, 2012.
10. D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-Bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, pp. 31–42. ACM, 2009.
11. M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 255–263. ACM, 2006.

12. G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted HTTP streams. In *Privacy Enhancing Technologies*, pp. 1–11. Springer, 2006.
13. A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pp. 171–178. Springer, 2003.
14. Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 19–30. IEEE, 2002.
15. S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy (SP)*, pp. 191–206. IEEE, 2010.
16. C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: an efficient defense against statistical traffic analysis. In *Network and Distributed System Security Symposium (NDSS)*, 2009.
17. X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. HTTPOS: sealing information leaks with browser-side obfuscation of encrypted flows. In *Network and Distributed System Security Symposium (NDSS)*, 2011.
18. M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli. Tunnel Hunter: detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 53(1):81–97, 2009.
19. How governments have tried to block Tor. https://svn.torproject.org/svn/projects/presentations/slides-28c3.pdf.
20. Tor: pluggable transports. https://www.torproject.org/docs/pluggable-transports.html.en.
21. H. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.
22. A. Houmansadr, T. Riedl, N. Borisov, and A. Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *Network and Distributed System Security Symposium (NDSS)*, 2013.
23. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol—HTTP/1.1. RFC 2616, June 1999.
24. M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928, April 1996.
25. V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23):2435–2463, 1999.
26. C. Inacio and B. Trammell. Yaf: yet another flowmeter. In *Proceedings of the Large Installation System Administration Conference (LISA)*, 2010.
27. L. Deri. nprobe: an open source netflow probe for gigabit networks. In *Proceedings of TERENA Networking Conference*, 2003.
28. A simple obfuscating proxy. https://www.torproject.org/projects/obfsproxy.html.en.
29. B. Wiley. Dust: a blocking-resistant internet transport protocol. Technical Report, School of Information, University of Texas at Austin, 2011.
30. K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2013.
31. M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In *Selected Areas in Cryptography*, pp. 295–312. Springer, 2009.

32. X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pp. 605–616. ACM, 2012.
33. X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2014.
34. M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS 2014)*, 2014.
35. Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.
36. Q. Wang, X. Gong, G. Nguyen, A. Houmansadr, and N. Borisov. CensorSpoofer: asymmetric communication using IP spoofing for censorship-resistant web browsing. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.
37. A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: observing unobservable network communications. In 34*th IEEE Symposium on Security and Privacy*, Oakland, CA, 2013.
38. J. Geddes, M. Schuchard, and N. Hopper. Cover your ACKs: pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 361–372. ACM, 2013.
39. L. Ptáček. Analysis and detection of Skype network traffic. Master's thesis, Masaryk University, 2011.
40. M. Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/blog/experimental-defense- website-traffic-fingerprinting, September 2011.
41. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
42. A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, pp. 103–114. ACM, 2011.
43. S. Jensen, K. Vos, and K. Soerensen. SILK speech codec. Technical Report draft-vos-silk-02.txt, IETF Secretariat, Fremont, CA, September 2010.
44. S. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer Internet telephony protocol. In *INFOCOM*, 2006.
45. D. Bonfiglio and M. Mellia. Tracking down Skype traffic. In *INFOCOM*, 2008.
46. E. Freire, A. Ziviani, and R. Salles. On metrics to distinguish Skype flows from HTTP traffic. *Journal of Network and Systems Management*, 17(1–2):53–72, 2009.