Computers & Security 92 (2020) 101777

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Protocol Proxy: An FTE-based covert channel

Jonathan Oakley*, Lu Yu, Xingsi Zhong, Ganesh Kumar Venayagamoorthy, Richard Brooks

Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA

ARTICLE INFO

Article history:
Received 25 September 2019
Revised 24 December 2019
Accepted 22 February 2020
Available online 24 February 2020

Keywords:
Covert channel
Format Transforming Encryption (FTE)
Steganography
Traffic analysis
Deep Packet Inspection (DPI)
Pluggable transport (PT)
Deterministic Hidden Markov Model (HMM)
Synchrophasor

ABSTRACT

In a hostile network environment, users must communicate without being detected. This involves blending in with the existing traffic. In some cases, a higher degree of secrecy is required. We present a proof-of-concept format transforming encryption (FTE)-based covert channel for tunneling TCP traffic through *protected static* protocols. Protected static protocols are UDP-based protocols with variable fields that cannot be blocked without collateral damage, such as power grid failures. We (1) convert TCP traffic to UDP traffic, (2) introduce observation-based FTE, and (3) model interpacket timing with a deterministic Hidden Markov Model (HMM). The resulting Protocol Proxy has a very low probability of detection and is an alternative to current covert channels. We tunnel a TCP session through a UDP protocol and guarantee delivery. Observation-based FTE ensures traffic cannot be detected by traditional rule-based analysis or DPI. A deterministic HMM ensures the Protocol Proxy accurately models interpacket timing to avoid detection by side-channel analysis. Finally, the choice of a *protected static* protocol foils stateful protocol analysis and causes collateral damage with false positives.

1. Introduction

Traffic analysis classifies network traffic using observable information. Network engineers use traffic analysis to ensure quality of service and identify threats. As a result, the development of hardware and software tools that quickly and effectively classify traffic has been encouraged. Commercial traffic analysis tools are used by governments to block access to websites that counter their current narrative (Heydari et al., 2017). Tools for countering traffic analysis have been developed for both criminal use and covert channels.

Tor is a popular overlay network that routes traffic through three randomly chosen nodes on the Internet. Tor uses nested encryption to ensure messages cannot be intercepted. The client encrypts packets. Each relay node decrypts the outermost layer, revealing another encrypted layer for the next hop to decrypt. By wrapping encryption (like the layers of an onion), it is possible to encrypt traffic so each node only knows its neighbors. This is not a silver bullet. In contested network environments, it is easy to detect and block Tor (Dingledine, 2011). To prevent blocking, pluggable transports (PTs) were developed to obfuscate Tor's traffic patterns.

PT developers must ensure their tools are able to penetrate nation state firewalls while authoritarian governments must determine the optimal defense (Garnaev et al., 2016). Some popular PTs simply wrap encrypted traffic with a new header to allow TLS traffic to pass through firewalls (Wiley, 2011; Yawning, 2019). At first, this may seem like an elegant solution, but it is simple to add another firewall rule to block this traffic. This is security through obscurity.

Format Transformation Encryption (FTE) is a form of steganography that translates network traffic into a host protocol.[1] Previous FTE implementations used regular expressions (Dyer et al., 2013) and context free grammars (Dyer et al., 2015). Padding and rerouting has obfuscated traffic and removed side-channels (Guan et al., 2001). In previous work, we used FTE and hidden Markov models (HMMs) to translate traffic flows into DNS requests and responses (Fu et al., 2016; 2017) and smart grid sensor traffic (Zhong et al., 2015b). Fridrich determined an upper bound on the amount of information that could be steganographically encoded in JPEG images before distortions were visually detected (Fridrich, 2006). HMMs have some notable advantages:

1. data windowing of HMMs (Schwier et al., 2011) makes them effective for both protocol detection and mimicry,
2. tools for differentiating HMMs are well defined (Schwier et al., 2011), and
3. a normalized metric space can directly measure the quality of protocol mimicry (Lu et al., 2013).

---

* Corresponding author.
E-mail address: joakley@g.clemson.edu (J. Oakley).

[1] The host protocol refers to the protocol being mimicked.

https://doi.org/10.1016/j.cose.2020.101777

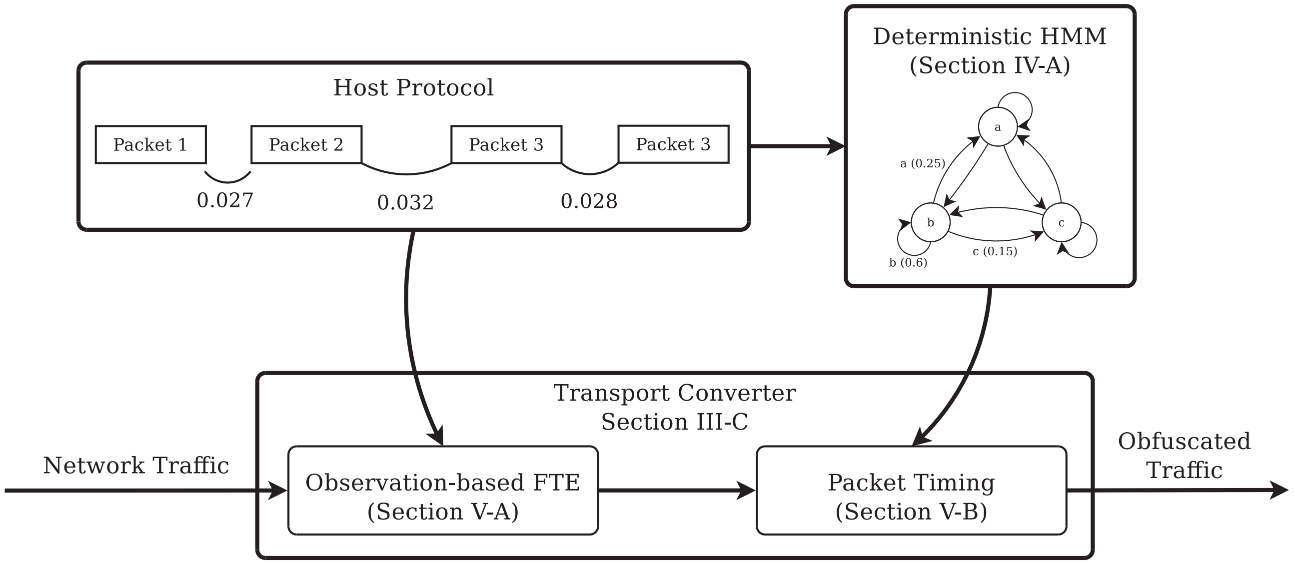


Fig. 1. Protocol Proxy architecture with relevant sections indicated.

We propose security through collateral damage. Certain protocols are more expensive to block than others. Usually, blocking the wrong TLS stream has little collateral damage other than disgruntled users.

We chose Synchrophasor traffic for our FTE implementation, but another example is Network Time Protocol (NTP) traffic. We use Synchrophasor traffic because we have access to Clemson's Real-Time Power and Intelligent System (RTPIS, 2019) Laboratory and real Synchrophasor traffic.[2] To accomplish this transformation, we made the following novel contributions:

1. An architecture to tunnel TCP traffic through UDP traffic.
2. Real-time observation-based format transforming encryption (FTE) (Zhong et al., 2015b).
3. A theoretical upper bound on the channel capacity of observation-based FTE.
4. Emulating the packet timing of a host protocol.
5. A proxy capable of tunneling SSH (TCP) through power-grid Synchrophasor traffic (UDP) in a statistically indistinguishable manner.

In Section 2, we introduce related work. In Section 3, we justify observation-based FTE as a undetectable communication channel. In Section 4, we provide the mathematical background behind deterministic HMMs. In Section 5, we provide the system architecture and justify our design decisions. Figure 1 shows our high-level system architecture: Section 5.1 describes observation-based format transforming encryption using our novel method. Section 5.2 describes massaging packet timing with a deterministic HMM inferred using the method described in Sections 4.1, and 5.3 describes how everything fits the overall Protocol Proxy architecture. Section 6 provides the experimental setup, and Section 7 details our results. Finally, we provide our closing thoughts and future work in Section 8.

2. Related work

Creating covert online communication tools has been the focus of many privacy advocacy groups. Since the data in covert channels is encrypted, the goal is to balance probability of detection with throughput based on the desired application (Smith and Knight, 2010). Timing side-channels are used when low probability of detection is prioritized over throughput (Kiyavash et al., 2013; Yao et al., 2009).

Named Data Networking (NDN) posits an alternative internet architecture based on content delivery (Zhang et al., 2014). Consumers express *interest* in a topic to NDN routers. These NDN routers check their *Content Stores* to see if an interest is cached. If the information is not available, the router adds the interest to a *Pending Interest Table* and forwards the interest upstream according to the *Forwarding Information Base* and *Forwarding Strategy*. Tsudik et. al proposed an anonymous communication network using NDN (Tsudik et al., 2016). Cui et al. proposed a model for preventing censorship using *smart* NDN routers (Cui et al., 2016). Neither of these solutions addresses the issue of covert communications in a contested environment.

Ambrosin et. al proposed a method for delay-based covert communication using cache techniques (Ambrosin et al., 2014). Given a sender and receiver share an NDN router at some point in the network, the sender and receiver can communicate using the round trip time (RTT) of the receiver's interest requests. The sender and receiver agree on C_0 and C_1 out-of-band. The sender requests a certain interest, C_b, and the receiver receives the message by requesting both C_0 and C_1. By comparing the RTT of both C_0 and C_1, it is possible to determine which interest the sender requested–C_b will have a shorter RTT since it already exists in the router's cache. While this covert channel is interesting, it would be easy to detect in Iran or China since CCNx (the NDN implementation) is not widely used. The traffic would be anomalous in that environment and could be used to identify users before being blocked (Mosko, 2014).

Tor (Tor, 2019) anonymity network wraps network traffic in layers of encryption. Each layer can only be decrypted by the next hop in the *onion* network. While it provides anonymous access to the Internet, the Tor protocol is easy to detect and block (Dingledine, 2011; Winter and Lindskog, 2012). Undetectable communication was not one of Tor's goals, but it spawned the Pluggable Transport project to address this challenge and encourage the development of other covert communication tools (Internews, 2017).

---

[2] Synchrophasor traffic is a UDP-based protocol generated by Phasor Measurement Units (PMUs) that contains alternating current phase measurements. This protocol is used to balance the load at different points in the power grid.

Pluggable Transports (Pluggable Transports, 2020) address this concern. PTs offer a generic way to obfuscate traffic. *Shape-shifting* PTs transform traffic into a different protocol. SkypeMorph (Mohajeri Moghaddam et al., 2012) makes network traffic resemble a Skype session. StegoTorus (Weinberg et al., 2012) demultiplexes connections to avoid traffic analysis and uses steganography to hide information in different protocols (including Skype). In *The parrot is dead: Observing unobservable network communications*, Houmansadr et al. (2013a) found both approaches fell short of true protocol mimicry. In both cases, handshake packets were incorrect. Other flaws were noted with StegoTorus's implementation of HTTP steganography (Houmansadr et al., 2013a). Censorspoofer mimics the Ekiga VoIP software, but it also falls short of mimicking protocol intricacies (Houmansadr et al., 2013a).

A number of PTs *scramble* traffic to remove fingerprints. Obfs2 (obfsproxy, 2015), Obfs3 (obfsproxy, 2015), Obfs4 (Yawning, 2019), and ScrambleSuite (Winter et al., 2013) each attempt to remove a network fingerprint by *scrambling* the data. Dust2 and its previous version (Dust) change statistical properties of traffic to bypass firewalls (Wiley, 2011). With technologies like software defined networking (SDN), these statistical PTs will likely be blocked by adaptive firewalls.

Recent PTs use *domain fronting*. Traffic is sent to a benign destination (Google, Amazon, Azure, etc.) and allowed through the firewall because blocking such a large domain would cause unintended collateral damage. FlashProxy (Moshchuk et al., 2008), SnowFlake (SnowFlake, 2016), and meek (Fifield et al., 2015) all use variations of *domain fronting*. This approach has been successful but is not condoned by companies whose domains are being used since it exposes them to potential backlash.

FTE PTs are a subset of *shape-shifting* PTs that steganographically encode traffic using values typical of the host protocol (Dyer et al., 2013). It is best to use a widely adopted protocol, such as DHCP (Rios et al., 2013) or VoIP (Schmidt et al., 2018). Marionette (Dyer et al., 2015) is a *shape-shifting* PT that uses a probabilistic context-free grammar (PCFG) and production rules to mimic the host protocol. The PCFG ensures traffic is syntactically and semantically correct and production rules occur at the expected frequency. Determining the appropriate PCFG to model a protocol is an open research question (Dyer et al., 2013; 2015). Marionette ensures interpacket timing, packet size, and session count mimic the host protocol.

Refraction Networking (TapDance) (Wustrow et al., 2014) spoofs the destination IP address. *If* the packet is routed through a decoy router, the true destination IP address is substituted for the spoofed address. Recent work has shown it may be inexpensive to censor decoy routers (Schuchard et al., 2012). Alternatively, TARN (Yu et al., 2017) provides an approach that mixes traffic from different autonomous systems at the software defined exchange (SDX) level. This provides a high level of anonymity and is resistant to a malicious ISP or BGP injection, but it is not realistic for a covert channel. Network-based moving target defense solutions have also been proposed (Heydari et al., 2017) for covert channels.

GNUnet (GNUnet, 2002), I2P (I2P, 2003), and (Freenet, 2001) all seek to provide anonymous access to the Internet. GNUnet is a toolbox for developing secure decentralized applications, but widespread censorship is possible (Kügler, 2003). I2P uses garlic routing (an onion-based routing protocol) to route traffic securely, but I2P is meant to be a self-contained network. I2P can be blocked if an adversary controls a small number of routers in the network and uses traditional IP-based filtering (Hoang et al., 2018). I2P routers can be identified because hiding I2P traffic was not a design goal (I2P's Threat Model, 2010). Freenet focuses on using prior knowledge to form connections, and it is possible to passively (or actively) scan the network for nodes (Roos et al., 2014). It arguably provides more anonymity, but it is resource intensive. Many governments block access to these tools, which makes the first hop important.

Traditional Virtual Private Networks (VPNs) are not usually effective in a contested environment because encrypted data can indicate malicious activity (Brandom, 2018). As a result, Psiphon (Psiphon, 2006), Lantern (Lantern, 2013), and Ultrasurf (Ultrasurf, 2002) have started using PTs. With Lantern, traffic is only forwarded through the PT if it is likely to be blocked.

Image steganography is also an effective means of covert communication. Fridrich investigated the relationship between distortion and information capacity (Fridrich, 2006). Unfortunately, the model derived in Fridrich (2006) does not directly apply to FTE-based covert channels.

2.1. Previous work

This article extends the work by Zhong et al. (2015b) where network traffic was manipulated offline as a proof-of-concept. The Protocol Proxy architecture presented in this work is a novel contribution designed to address the issues that result from real-time traffic manipulation. The observation-based FTE algorithm was enhanced from Zhong et al. (2015b) to increase throughput. In this work, we present a theoretical bound on the information that can be encoded using observation-based FTE. Zhong et al. (2015b) manipulated packet timing in an offline proof-of-concept by setting the packet timestamp in the PCAP file. In this work, we manipulate packet timing in real-time, which creates additional challenges that are addressed by the novel Protocol Proxy architecture. Zhong et al. (2015b) did not consider guaranteed delivery for a UDP protocol, which must be addressed when manipulating TCP traffic in realtime. Finally, while Zhong et al. (2015b) used HMMs to manipulate packet timing, they did not consider the need for formal model verification.

3. Undetectability

Detecting protocol mimicry can be done in several ways: rule-based analysis, deep packet inspection (DPI), stateful protocol analysis, side-channel analysis, and statistical analysis. If a packet matches a set of predetermined rules, then it is accepted (or rejected). Rules typically look only at the packet headers. They concentrate on IP source, IP destination, source port, destination port, protocol type (TCP or UDP), and several other fields. These fields are available in the packet header without the need for DPI, so this is the first line of defense for high throughput use cases. If packets are destined for IP addresses that belong to a malicious web site, they will be dropped before they leave their respective autonomous systems. Source and destination ports are also used to classify traffic (Kim et al., 2008).

DPI classifies traffic by analyzing the protocol payload. Recent developments allow primitive classification of HTTPS (encrypted) traffic (Miller et al., 2014). It has been shown FTE can bypass both rule-based analysis and DPI (Dyer et al., 2013; 2015).

"Stateful" firewalls avoid certain attacks by only permitting traffic if the traffic obeys the underlying protocol. For instance, TCP traffic is required to complete the handshake before data packets are allowed through the firewall. Houmansadr et al. (2013b) used an extension of this idea to identify protocol mimicry. By classifying the states of a host protocol, it is possible to identify poor imitations by identifying discrepancies between observed and expected states. Housmansadr et. al used this to find where SkypeMorph differed from Skype. Once differences were identified, simple rules can identify SkypeMorph traffic.

Hidden Markov Models (HMMs) have been used in side-channel analysis to identify Synchrophasor traffic in an encrypted VPN (Zhong et al., 2015a). HMM inference occurs offline, and it requires a large sample of traffic to build the timing model of the unidentified protocol. In the future, HMM detection will likely be performed online.

Statistical analysis is a hybrid approach that attempts to use statistical properties to identify traffic. Chaos theory is one approach to statistical detection (Zhao and Shi, 2012). Entropy has also been used with distributed denial of service (DDoS) detection, but entropy can easily be spoofed (Özçelik and Brooks, 2015). We use statistical analysis to compare the timing models of the original traffic with the traffic we generated.

Our variation of protocol mimicry uses a *protected static* protocol. We use this term to refer to a specific subset of protocols that are prime candidates for protocol mimicry. Static protocols are UDP-based and lack application-layer handshakes (like those in Skype), making them immune to stateful analysis. The final layer of security is choosing protocols that are *protected*. These protocols have high collateral damage for false positives. If Synchrophasor packets are dropped, it can have adverse consequences for the power grid.

4. Hidden Markov models

A Markov model is a tuple $G = (S, T, P)$ where S is a set of states of a model, T is a set of directed transitions between the states, and $P = \left\{p(s_i, s_j)\right\}$ is a probability matrix associated with transitions from state s_i to s_j such that:

$$\sum_{s_j \in S} p(s_i, s_j) = 1, \forall s_i \in S \tag{1}$$

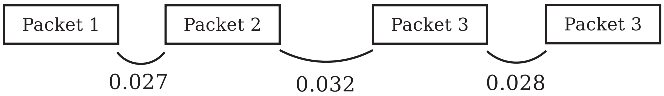
A Markov model satisfies the Markov property, where the next state only depends on the current state. An HMM is a Markov model with unobservable states. A standard HMM (Eddy, 1996; Rabiner, 1989) has two sets of random processes: one for state transition and the other for symbol outputs. HMMs have been used to effectively model time series data (Asadi et al., 2016). A deterministic HMM (Lu, 2012; Lu et al., 2013; Schwier, 2009) is used in this paper, and it has one random processes for state transitions. Different output symbols are associated with transitions with different probability. This representations is equivalent to the standard HMM (Lu et al., 2013; Vanluyten et al., 2008).

4.1. Inferring deterministic HMMs

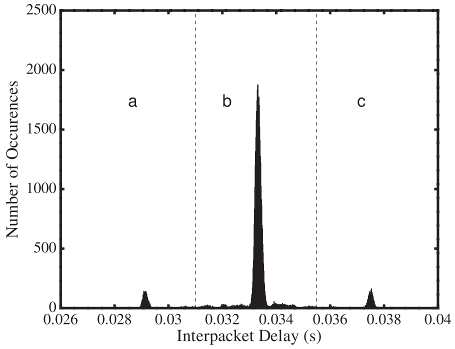
Deterministic HMM inference is depicted in Fig. 2. A stream of network packets, Fig. 2a, is observed. The interpacket delay (time between each packet) is calculated, and the values are plotted in a histogram. This histogram is grouped into different states, and these states manifest themselves as peaks in the histogram. In Fig. 2b, there are three peaks. Each peak is given a unique label. The stream of interpacket delays is re-interpreted using the assigned labels. A stream of labels, as shown in Fig. 2c, is used to infer the deterministic HMM shown in Fig. 2d. Each state in the HMM corresponds to a label. The probability of an ‘a’ output expression in state ‘b’ is given by the number of occurrences of the string ‘ba’ divided by the number of occurrences of the string ‘b’. If there were 1000 occurrences of the string ‘b’, and we know the string ‘ba’ occurred 250 times, then 25% of the time we transitioned to state ‘a’. The full process for inferring deterministic HMMs is provided in Griffin et al. (2011) and Schwier et al. (2009). Given a deterministic HMM, it is possible to generate a stream of packet timings.

4.2. Comparing deterministic HMMs

In Lu et al. (2013), the authors develop a normalized metric space for comparing HMMs, and in Yu et al. (2013) the authors show a method for ensuring an inferred HMM is significant. We use an alternative approach that is tailored to this challenge. Before determining whether two deterministic HMMs are equal, it is desirable to ensure the probability distribution functions (PDFs) used to generate the HMM are equal. To do this, we use the two-sample Kolmogornov–Smirnov (KS) test (Kolmogorov-Smirnov Test, 2008), which tests the null hypothesis (two sets of samples come from the same underlying distribution) against the alternate hypothesis (two sets of samples come from different underlying distributions). The KS statistic is the empirical distribution function F_n, defined below.



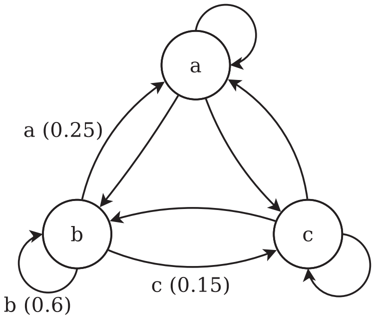
(a) Stream of incoming packets with timing denoted.



(b) The timing values plotted in a histogram.

aba....aacbacab...

(c) The packet timings converted to a stream labels.



(d) The deterministic HMM inferred from the stream of labels.

Fig. 2. Detailed example of how a deterministic HMM is inferred from packet timing.

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(X_i) \tag{2}$$

Here, n refers to the number of identically independently distributed samples (X_i) taken from the sample space (X). Samples (X_i) are randomly chosen observations from Fig. 2a. The indicator function, $I_{(-\infty,x]}(X_i)$, is defined in Eq. (3).

$$I_{[-\infty,x]}(X_i) = \begin{cases} 1, & X_i < x \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The two-sample KS test compares the distance between the two empirical distribution functions using Eq. (4).

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)| \tag{4}$$

The null hypothesis is rejected at the 95% confidence level if the following criterion is met.

$$D_{n,m} > 1.36\sqrt{\frac{n+m}{nm}} \tag{5}$$

To determine whether two deterministic HMMs are equivalent, it is sufficient to show all corresponding states in the deterministic HMM are equivalent. If all states are equivalent, the HMMs are equivalent. To show two states of a deterministic HMM are equivalent, we use the χ^2-test for homogeneity to test if the probability distributions for outgoing state transitions are statistically equivalent. The generic expression for the χ^2 statistic for homogeneity given P populations and C levels of the categorical variable is shown below.

$$\chi^2 = \sum_{i \in P} \sum_{j \in C} \frac{\left(O_{i,j} - E_{i,j}\right)^2}{E_{i,j}} \tag{6}$$

In this representation, $O_{i,j}$ is the number of occurrences observed in the state corresponding to i and the output expression corresponding to j. Similarly, $E_{i,j}$ is the number of *expected* occurrences for the combination of state and output expression. The expected number of occurrences is calculated as shown below in Eq. (7).

$$E_{i,j} = \frac{n_i n_j}{n} \tag{7}$$

Here, n_i is the number of observations in state i, n_j is the number of observations at that level of the categorical variable, and n is the sample size. For threshold testing, the degrees of freedom (DF) is given as follows.

$$DF = (P-1)(C-1) \tag{8}$$

In this work, we compare two states (populations), so P is 2. Therefore, the DF for any given state is simply the number of output expressions (C) minus one.

5. Architecture

Converting TCP-based Tor traffic to the UDP Synchrophasor protocol requires a number of building blocks that were not present in Zhong et al. (2015b). The TCP packet must be converted to several Synchrophasor packets. The packet timing of outgoing packets must be adjusted to model the timing of Synchrophasor traffic. A transport layer converter is required to tunnel TCP-based protocols through UDP protocols while still maintaining TCP's guaranteed delivery. Finally, these building blocks can be linked together to provide a proof-of-concept that can convert Tor's TCP traffic to UDP Synchrophasor traffic.

5.1. Observation-based FTE

Simply sending UDP packets to a specific port isn't enough. Capturing the packet in an analysis tool like Wireshark (2019) will reveal the packet is malformed. While this rises to the level of existing obfuscation PTs, it does not solve the problem. Traditional FTE takes the syntax of a protocol and creates a PCFG to map raw binary data to that protocol's syntax (Dyer et al., 2012). Determining the appropriate PCFG to model a protocol is left as an open research question, which makes it unrealistic to deploy (Dyer et al., 2013; 2015).

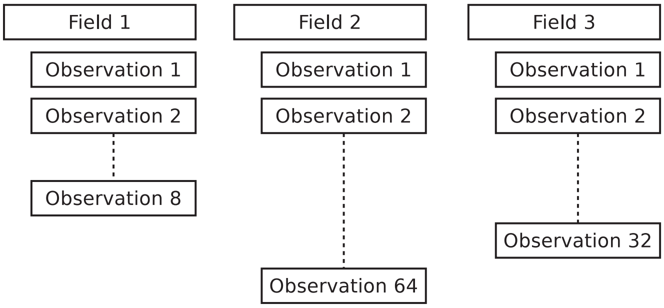


Fig. 3. Example protocol to illustrate observation-based FTE.

We propose observation-based FTE, as an alternative. We collect a substantial amount of traffic and record the unique observations for each field in the protocol. If Alice and Bob want to encode information using observation-based FTE with the protocol shown in Fig. 3, they construct a lookup table with each field and an ordered list of observations. This lookup table is a shared out-of-band. High-entropy (encrypted) information can be encoded using observation-based FTE by construction a packet using observations from the host protocol. With the protocol in Fig. 3, three bits can be encoded in the first field. When Alice receives the message and sees 'observation 5' in 'field 1', she uses the shared lookup table to determine the first three encoded bits are '101' (the binary value of the observation's index). The high-entropy input also ensures each field (and each packet) is independent of the other fields (and packets)[3].

Zhong et al. (2015b) used a primitive version of observation-based FTE that did not consider the upper bound on the information capacity of an FTE channel.

Theorem 1. *For a given protocol, the maximum amount of information that can be encoded in a packet using observation-based FTE is given by:*

$$S = \sum_{\gamma_i \in \Gamma} \log_2(|\boldsymbol{\gamma}_i|) \tag{9}$$

Where $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ *is the set of n fields in the protocol, and* $|\gamma_i|$ *is the number of unique observations in that field.*

Proof. The maximum amount of information that can be encoded in a particular field using observation-based FTE is given by the Shannon entropy of that field.

$$H(\gamma_i) = -\sum_{x \in \gamma_i} p(x) \log_2(p(x)) \tag{10}$$

Each stream of n bits is equally likely.[3] Therefore, the choice of each observation is equally likely. This simplifies Eq. (11) as follows.

$$\begin{aligned} H(\gamma_i) &= -\sum_{x \in \gamma_i} \frac{1}{|\gamma_i|} \log_2\left(\frac{1}{|\gamma_i|}\right) \\ &= -|\gamma_i| \frac{1}{|\gamma_i|} \log_2\left(\frac{1}{|\gamma_i|}\right) \\ &= -\log_2\left(\frac{1}{|\gamma_i|}\right) \\ &= \log_2(|\gamma_i|) \end{aligned} \tag{11}$$

---

[3] Since the data being mapped to the protocol is encrypted using AES encryption and AES produces a high-entropy bitstream (Lyda and Hamrock, 2007), we can assume 0 and 1 are equally likely in practice.

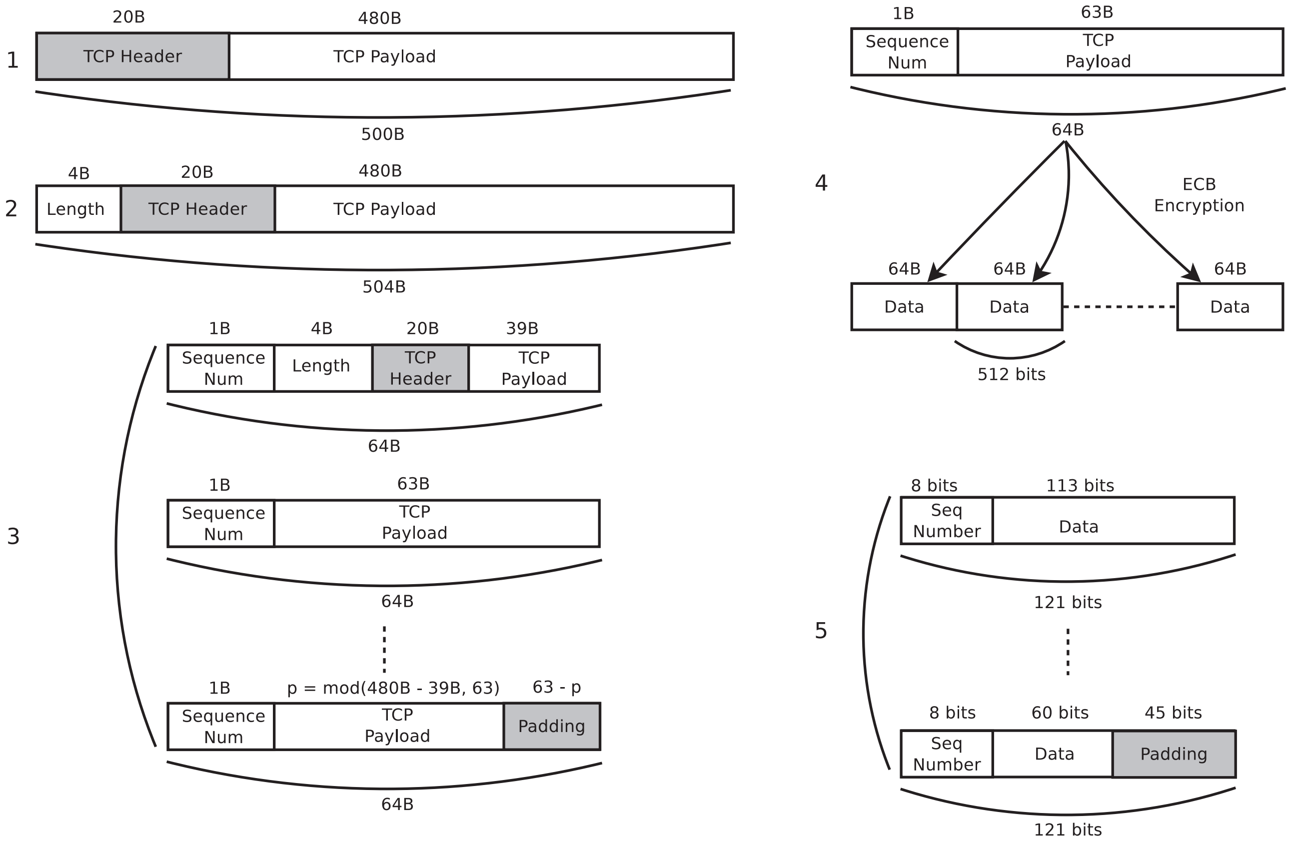


Fig. 4. Process for segmenting TCP packets for transmission.

The amount of information that can be encoded in a single packet is the sum of the information that can be encoded in each field in the packet.

$$S = \sum_{\gamma_i \in \Gamma} \log_2(|\gamma_i|) \tag{12}$$

□

Performing these calculation on the Synchrophasor protocol yields 516 bits that can be encoded in a single UDP packet. Since this is smaller than the typical TCP packet, it is necessary to segment TCP packets for transmission. The optimal average goodput (G_{avg}) can be calculated with Eq. (13), where S is found using Eq. (12), and T_{avg} is the average interpacket delay, which is 0.03334 s for Synchrophasor traffic. This yields an theoretical average goodput of 15,477 bits per second.

$$G_{avg} = \frac{S}{T_{avg}} \tag{13}$$

Segmentation is shown in Fig. 4, where each item below is indicated in the figure:

1. The original TCP packet is taken.
2. The packet length is prepended to the beginning of packet as a four byte unsigned integer.
3. The packet is broken into 63 byte chunks, and each chunk is prepended with a one byte sequence numbers for a total of 64 bytes. The sequence number allows the chunks to be reassembled later into the original TCP packet. Depending on the size of the packet, it is possible there will not be enough payload data to fill the final chunk. In this case, random data is appended to the end.
4. Each 64 byte chunk is encrypted with Electronic Code Book (ECB) encryption[4].
5. The observation-based FTE encodes each 64 byte (512 bit) chunk into a 516 bit UDP payloads using the method previously discussed.

5.2. Packet timing

The HMM timing model described in Section 4 was given to the Protocol Proxy, which queries the timing model for a timing value. When the model is queried, it examines its current state, yields an output expression based on the probability distribution of the current state, and chooses a timing value from the output expression group. The model advances to the chosen state. The Protocol Proxy waits for the allotted time before sending a packet. If there are no packets to send, random data is encoded and sent. These packets are dropped by the server. Sending placeholder packets ensures the correct timing model is emulated when there are no packets to transmit.

This approach to packet timing differs from the NDN covert channel proposed in Ambrosin et al. (2014). Our approach does not convey any information using interpacket delays. Interpacket delays can often be an indicator of a covert channel, so we ensure the timing of the covert channel is statistically identical to the host protocol.

5.3. Protocol Proxy

The Protocol Proxy combines the previous elements to create a covert channel using the host protocol.

---

[4] ECB is used because packets may arrive out of order, which makes cipher block chain impractical.

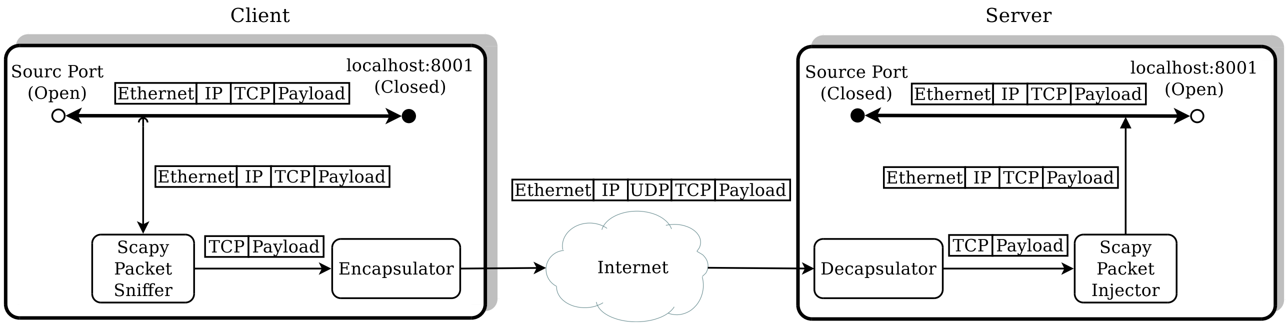


Fig. 5. Architecture for the Protocol Proxy transport converter.

```
sudo iptables -A OUTPUT -p tcp --tcp-flags RST \
  RST -j DROP
```

Listing 1. The iptables command to disable RST packets.

We use Scapy (Biondi, 2008)[5] to capture TCP packets. These captured packet includes both the TCP header and payload. Applications using the Protocol Proxy are configured to send traffic to a closed loopback port, and Scapy sniffs the loopback interface looking for traffic destined to that port. By default, when a closed port receives a TCP packet, it responds with a TCP RST (reset) packet. The RST packet immediately terminates the connection and ceases all communication with the other host. This is a low-level response dictated by the host firewall. The transport converter requires this RST packet not be sent, so the internal firewall (iptables on Linux) was modified accordingly. The following command disables RST packets for all ports on Linux.

In Figure 5, an application on the client sends a packet to a closed local port (8001 in our example). Scapy sniffs the network stack and captures the entire packet. The Ethernet and IP layers are stripped, and the packet is transformed into a UDP payload using observation-based FTE, as described in Section 5.1. After transforming the packet into the host protocol's payload, new Ethernet, IP, and UDP layers are generated so the packet can be sent across the network to the server. All the UDP packets are sent to the port that corresponds to the emulated protocol. The timing of outgoing packets is determined by the HMM described in Section 5.2.

On the server, the Protocol Proxy listens on the predetermined port for the UDP packet. Once it receives the UDP packet, it takes the UDP payload, reverses the observation-base FTE transformation described in Section 5.1, and sends it to a Scapy packet injector. The Scapy packet injector creates new Ethernet and IP layers to make the packet look like it originated from the local machine. Then, Scapy injects the new packet into the local network stack, and it is received by the local application. This provides a bidirectional covert channel that uses actual observations from host protocol.

Since the Protocol Proxy is accessed via an open port, it is trivial to integrate with Tor. Tor natively supports tunneling outgoing connections through a SOCKS proxy, so the Tor client is configured to use the Protocol Proxy port as a SOCKS proxy. On the server, a SOCKS server listens for traffic forwarded from the Protocol Proxy.

6. Experiment setup

Over 770,000 samples were collected from the PMUs in Clemson's RTPIS Laboratory (RTPIS, 2019). These samples were used to model the timing of the protocol as described in Section 5.2. All testing was done in Clemson's security lab with clean installations of Arch Linux (kernel version 4.17.2-1). The experiment setup is shown in Fig. 6. We used `scp` to transfer data over the Protocol Proxy to an SSH server on a remote machine. The Protocol Proxy server was launched using the following command. server# protocol_proxy server 192.168.10.23 8001

Since the Protocol Proxy requires access to raw sockets, it must be executed by a privileged user. The 'server' option tells the Protocol Proxy to expect packets originating from the specified port (8001). The IP address (192.168.10.23) is the IP address of the client that will connect to the server. The next step is to ensure the kernel does not send a reset packet (TCP RST) when packets are sent to the closed port (8001). This is done by executing the `iptables` command shown in Listing 1. Next, the SSH server is set to listen to port 8001 for incoming connections in the `/etc/ssh/sshd_config` file. SSH (OpenSSH_7.7p1) was used for the server. It was necessary to configure a non-standard port to avoid conflict when forwarding the traffic through the Protocol Proxy. The client was launched with the following command. client# protocol_proxy client 192.168.10.24 8001

Again, the application must be executed by a privileged user. The 'client' option tells the transport to expect packets destined for the specified port (8001). The IP address (192.168.10.24) is the IP address of the host executing the program. As with the server, the client does not open the port, so the rules in Listing 1 must be applied to the client to ensure TCP connections are not prematurely terminated.

A one kilobyte data file was transferred from the client to the server using `scp` as shown below. client# scp -P 8001 file 127.0.0.1:file

The destination port was set to 8001 on the client, which was the local port being forwarded to the Protocol Proxy. The traffic between the client to the server was captured, and another HMM was inferred from this generated traffic. This second HMM was compared via the χ^2-test to the original HMM used by the Protocol Proxy. Finally, the baseline goodput was measured by recording the time required to transfer the 1 kilobyte data file.

7. Results

Figure 7 shows the histogram of interpacket delay times for the Synchrophasor traffic captured in Clemson's RTPIS laboratory. The output expressions are labeled in the histogram according to the prominent peaks. Using the techniques described in Section 4, the deterministic HMM in Figure 8 was inferred.

With this HMM, it was then possible to generate Synchrophasor traffic with observation-based FTE and accurate timing. Fig. 9 shows a Wireshark deconstruction of the traffic generated with our Protocol Proxy. Wireshark correctly identifies the Protocol Proxy traffic as Synchrophasor traffic and is able to parse the field val-

---

[5] Scapy is a Python library for packet capture, manipulation, and injection

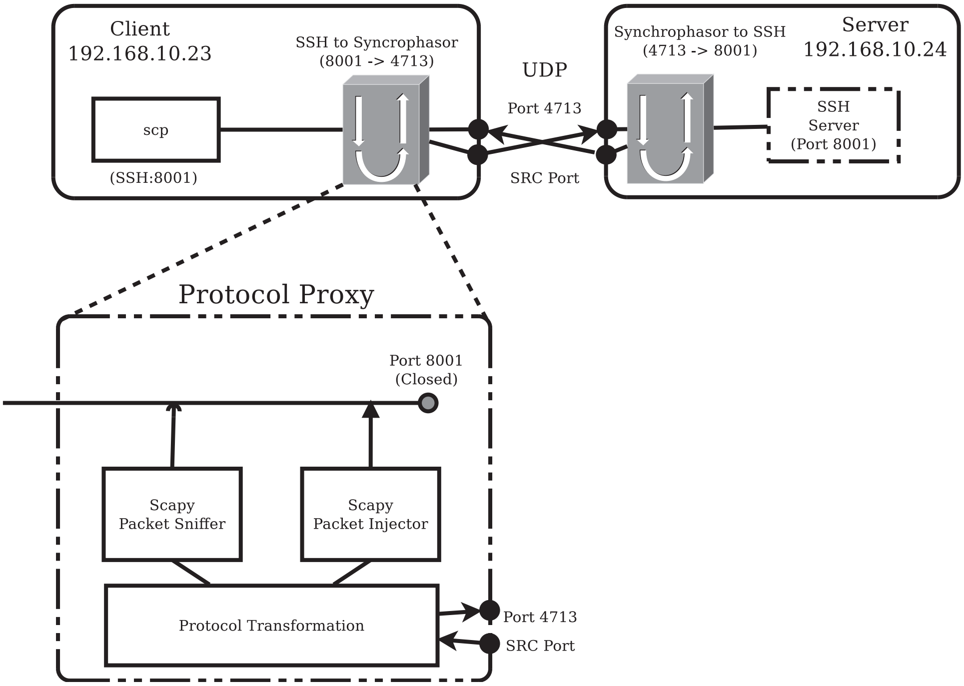


Fig. 6. The Protocol Proxy integrated for use with SCP.

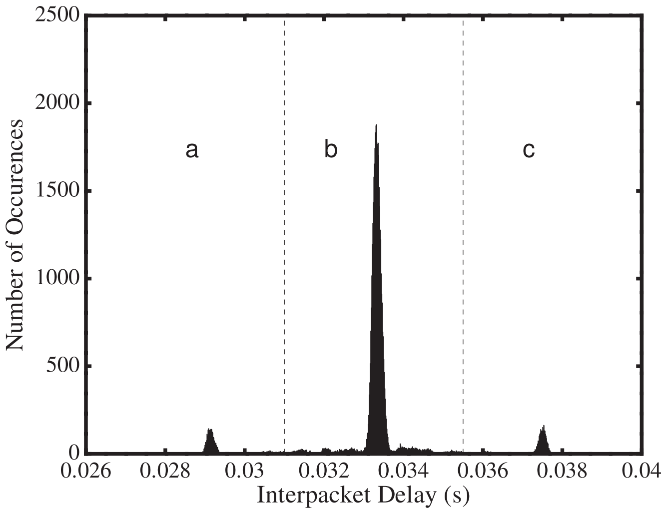


Fig. 7. Histogram of the interpacket delay of real Synchrophasor traffic with states labeled.

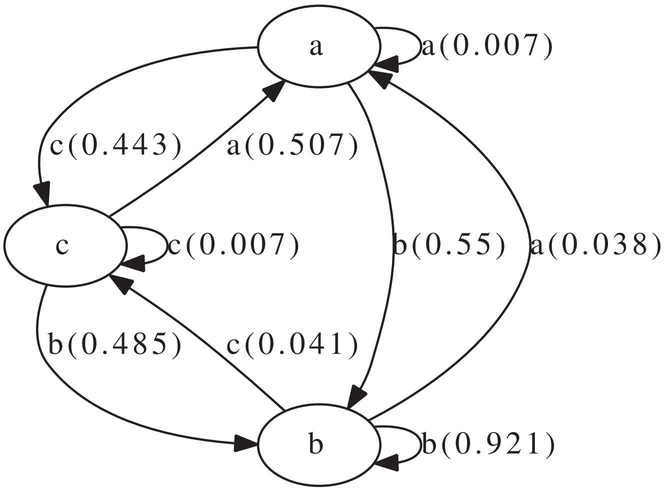
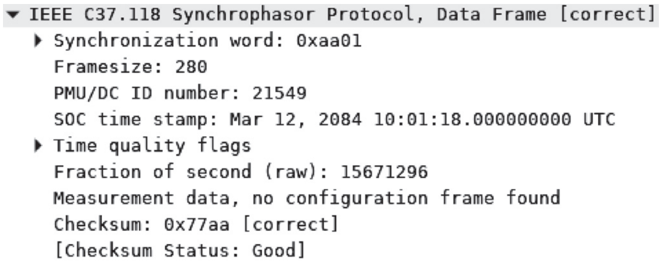


Fig. 8. HMM generated from the interpacket delay of Synchrophasor traffic.



```
▾ IEEE C37.118 Synchrophasor Protocol, Data Frame [correct]
  ▸ Synchronization word: 0xaa01
    Framesize: 280
    PMU/DC ID number: 21549
    SOC time stamp: Mar 12, 2084 10:01:18.000000000 UTC
  ▸ Time quality flags
    Fraction of second (raw): 15671296
    Measurement data, no configuration frame found
    Checksum: 0x77aa [correct]
    [Checksum Status: Good]
```

Fig. 9. Wireshark decoding of the Protocol Proxy traffic.

ues from the payload. The checksum is also correctly calculated. This FTE-generated PMU traffic is accepted by the Phasor Data Concentrators–the hardware PMU datastore (Zhong et al., 2015b). Since packets generated by the Protocol Proxy only use previously observed field values, the Protocol Proxy is syntactically equivalent to the host protocol. Any rule that would detect Protocol Proxy traffic would have false-positives on legitimate PMU traffic, and these false-positives would lead to significant collateral damage.

Fig. 10 shows the histogram of interpacket delay times for the generated traffic with the output expressions labeled. Fig. 11 shows the deterministic HMM inferred from the histogram to model the timing patterns of the Protocol Proxy traffic. Visually, this model appears almost identical to the model used to generate the traffic.

Before determining if the two deterministic HMMs were equal, the two-sample KS test was applied to the two distributions (shown in Figs. 7 and 10). To apply this test, we randomly sample each distribution 100 times and apply the test using the two sets of samples. The p-value for the two-sample KS test was found to be 0.21, so with a threshold of 0.05, we fail to reject the null hypothesis. The interpacket delay times of the Protocol Proxy are

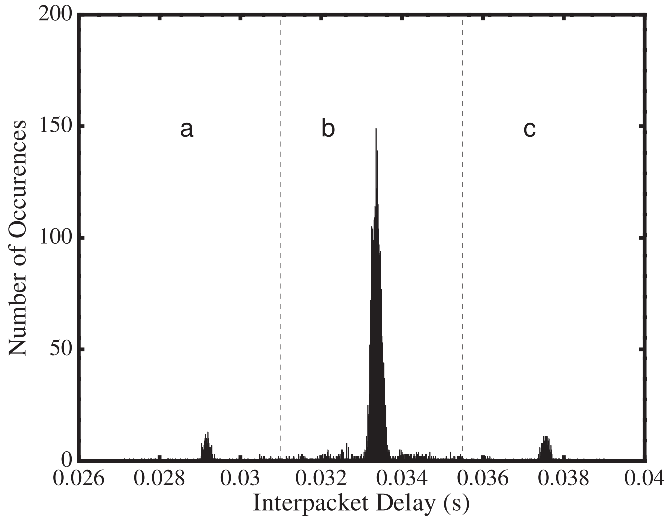


Fig. 10. Histogram of the interpacket delay of generated Synchrophasor traffic with states labeled.

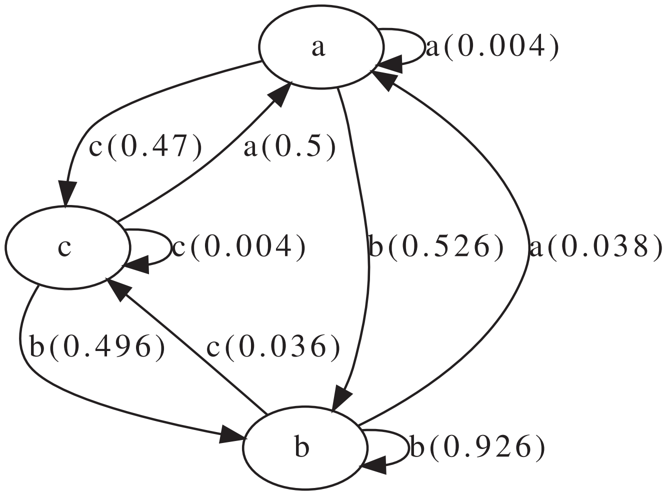


Fig. 11. HMM generated from the interpacket delay of the generated Synchrophasor traffic.

Table 1
State-wise χ^2 test for homogeneity comparing HMMs.

State Comparison	Inferred-Inferred (*p*-value)	Generated-Inferred (*p*-value)
a-a	0.75	0.82
b-b	0.19	0.37
c-c	0.06	0.15

from the probability distribution as the interpacket delay times of the original Synchrophasor traffic.

To determine if the two deterministic HMMs were equal, the HMMs were checked for state-wise equality using the χ^2 test for homogeneity. The p-values for the χ^2 test are shown in Table 1. The first comparison (inferred-inferred) infers two HMMs using 10,000 samples and a random starting point in the original traffic. From these values, we fail to reject the null hypothesis (with an α value of 0.05) for every state and are left to conclude the traffic is *homogeneous*, which means it does not change over time. The second comparison (generated-inferred) infers one HMM from the Protocol Proxy traffic and another HMM from the original Synchrophasor traffic. From these values, we fail to reject the null hypothesis (with an α value of 0.05) for every state and are left to conclude the traffic from the Protocol Proxy is equivalent to the *homogeneous* Synchrophasor traffic.

Table 2
Comparison of observed and theoretical goodput through the Protocol Proxy.

	Baseline	Theoretical	Observed
Goodput	54 Mbps	15,477 bps	182 bps

The baseline goodput (link speed) was determined to be 54 Mbps, while the goodput through the PMU Protocol Proxy was found to be 182.2 bits per second. These values are compared to the theoretical goodput in Table 2. The difference between theoretical and observed goodput is attributed to retransmission and packet overhead (the TCP header is sent through the Protocol Proxy). We measured the goodput through Tor to be around 7.31 Mbps using an online speed test[6].

8. Conclusion and future work

Covert communication techniques must evade several types of threats: rule-based detection, DPI, stateful protocol analysis, side-channel analysis, and statistical analysis. To address these issues, we presented a novel approach for tunneling TCP traffic through *protected static* protocols. Protected protocols have collateral damage associated with false positives, and static protocols are deterministic UDP-based protocols whose pattern never changes. To accomplish this, we introduced (1) an architecture to convert TCP traffic to UDP traffic[7], (2) observation-based FTE to mimic the host protocol's payload, and (3) a deterministic HMM to model the host protocol's interpacket timing. The Wireshark packet capture in Fig. 9 illustrates objectives (1) and (2)–Protocol Proxy traffic is syntactically equivalent to the host protocol. The χ^2 test shows the timing generated by the Protocol Proxy is statistically equivalent to the host protocol.

This extends the simulation present in Zhong et al. (2015b) by developing an architecture to perform the protocol transformations in real-time. The Protocol Proxy provides a universal interface for connecting applications. A theoretical upper-bound for the information capacity of an observation-based FTE channel was derived, which facilitated the improvement of the previous observation-based FTE implementation. During testing, it was found some Linux distributions were incapable of emulating the timing of the host protocol. Arch Linux was chosen for its ability to consistently emulate timing. The cause of this discrepancy is currently being investigated. The Protocol Proxy does not transmit information using the interpacket delay as with the NDN-based covert channel (Ambrosin et al., 2014), and the presence of NDN traffic itself could be enough to indicate a covert channel.

Future work will provide an in-depth security analysis of the Protocol Proxy, decrease the overhead associated with packet retransmissions, and adapt the Protocol Proxy to Tor's PT version 2.0 specification (Internews, 2017). While the Protocol Proxy's goodput was significantly slower than Tor, the use-case is situations when any anomalous traffic could have negative repercussions. This trade-off between performance and detection is justified in those cases. We will also investigate a number of performance improvements. For instance, the entire TCP packet is currently transmitted through the proxy, but in reality much less data is required for most packets. The Protocol Proxy approach may be vulnerable

---

[6] https://speedof.me

[7] The conversion from TCP to UDP is not intended to provide an additional layer of security. It is necessary because the host protocol is UDP-based, and it is an open challenge in PT development.

to semantic analysis using probabilistic context-free grammars, but these techniques are not currently used in the wild for scalability reasons.

The Protocol Proxy is a viable alternative to traditional transports when heightened anonymity is required. While there are a number of improvements that will increase throughput, these preliminary results show it is possible to tunnel a TCP session through a UDP protocol and maintain TCP's guaranteed delivery. Observation-based FTE extends this and ensures the traffic will not be detected by rule-based analysis or DPI. Furthermore, a deterministic HMM ensures the Protocol Proxy accurately models interpacket timing and avoids detection by side-channel analysis. Finally, the choice of a *protected static* protocol ensures stateful protocol analysis is useless while raising the collateral damage associated with false positives.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This material is based upon work supported by, or in part by, the National Science Foundation grants CNS-1049765, OAC-1547245, and CNS-1544910. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The authors gratefully acknowledge this support and take responsibility for the contents of this report. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation, or the U.S. Government.

References

Ambrosin, M., Conti, M., Gasti, P., Tsudik, G., 2014. Covert ephemeral communication in named data networking. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. ACM, pp. 15–26.

Asadi, N., Mirzaei, A., Haghshenas, E., 2016. Creating discriminative models for time series classification and clustering by HMM ensembles. IEEE Trans. Cybern. 46 (12), 2899–2910.

Biondi, P., 2008. Scapy. [Online]. Available: https://scapy.net/.

Cui, X., Tsang, Y.H., Hui, L.C., Yiu, S., Luo, B., 2016. Defend against internet censorship in named data networking. In: 2016 18th International Conference on Advanced Communication Technology (ICACT). IEEE, pp. 300–305.

Dingledine, R., 2011. Ten Ways to Discover Tor Bridges. Technical Report. Technical Report 2011-10-002, The Tor Project, October 2011. https://research. torproject. org/techreports/tenwaysdiscover-tor-bridges-2 11-1-31. pdf, Tech. Rep., 2011. 4, 10.

Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T., 2012. Format-transforming encryption: more than meets the dpi.. IACR Cryptol. ePrint Arch. 2012, 494.

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

Dyer, K.P., Coull, S.E., Shrimpton, T., 2015. Marionette: a programmable network traffic obfuscation system.. In: USENIX Security Symposium, pp. 367–382.

Eddy, S.R., 1996. Hidden Markov models. Current Opin. Struct. Biol. 6 (3), 361–365.

Fifield, D., Lan, C., Hynes, R., Wegmann, P., Paxson, V., 2015. Blocking-resistant communication through domain fronting. Proc. Privacy Enhanc. Technol. 2015 (2), 46–64.

Fridrich, J., 2006. Minimizing the embedding impact in steganography. In: Proceedings of the 8th Workshop on Multimedia and Security. ACM, pp. 2–10.

Fu, Y., Jia, Z., Yu, L., Zhong, X., Brooks, R., 2016. A covert data transport protocol. In: 2016 11th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, pp. 1–8.

Fu, Y., Yu, L., Hambolu, O., Ozcelik, I., Husain, B., Sun, J., Sapra, K., Du, D., Beasley, C.T., Brooks, R.R., 2017. Stealthy domain generation algorithms. IEEE Trans. Inf. Forensics Secur. 12 (6), 1430–1443.

Garnaev, A., Baykal-Gursoy, M., Poor, H.V., 2016. Security games with unknown adversarial strategies. IEEE Trans. Cybern. 46 (10), 2291–2299.

Griffin, C., Brooks, R.R., Schwier, J., 2011. A hybrid statistical technique for modeling recurrent tracks in a compact set. IEEE Trans. Autom. Control 56 (8), 1926–1931.

Guan, Y., Fu, X., Xuan, D., Shenoy, P.U., Bettati, R., Zhao, W., 2001. Netcamo: camouflaging network traffic for QoS-guaranteed mission critical applications. IEEE Trans. Syst. Man. Cybern. Part A 31 (4), 253–265.

Heydari, V., Kim, S., Yoo, S.-M., 2017. Scalable anti-censorship framework using moving target defense for web servers. IEEE Trans. Inf. Forensics Secur. 12 (5), 1113–1124.

Hoang, N.P., Kintis, P., Antonakakis, M., Polychronakis, M., 2018. An empirical study of the i2p anonymity network and its censorship resistance. In: Proceedings of the Internet Measurement Conference 2018. ACM, pp. 379–392.

Houmansadr, A., Brubaker, C., Shmatikov, V., 2013. The parrot is dead: observing unobservable network communications. In: Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, pp. 65–79.

Houmansadr, A., Riedl, T.J., Borisov, N., Singer, A.C., 2013. I want my voice to be heard: Ip over Voice-over-IP for unobservable censorship circumvention.. NDSS.

Tor, 2019. [Online]. Available: https://www.torproject.org/.

Freenet, 2001. [Online]. Available: https://freenetproject.org/author/freenet-project-inc.html.

Brandom, R., 2018. Iran blocks encrypted messaging apps amid nationwide protests. Available: https://www.theverge.com/2018/1/2/16841292/iran-telegram-block-encryption-protest-google-signal.

I2P's Threat Model, 2010. [Online]. Available: https://geti2p.net/en/docs/how/threat-model.

Internews, 2017. New pluggable transport specification version 2.0, draft 2 is out.

Kolmogorov-Smirnov Test, 2008. Springer New York, New York, NY. pp. 283–287. 10.1007/978-0-387-32833-1_214.

Kim, H., Claffy, K.C., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K., 2008. Internet traffic classification demystified: myths, caveats, and the best practices. In: Proceedings of the 2008 ACM CoNEXT Conference. ACM, p. 11.

Kiyavash, N., Koushanfar, F., Coleman, T.P., Rodrigues, M., 2013. A timing channel spyware for the CSMA/CA protocol. IEEE Trans. Inf. Forensics Secur. 8 (3), 477–487.

Kügler, D., 2003. An analysis of GNUnet and the implications for anonymous, censorship-resistant networks. In: International Workshop on Privacy Enhancing Technologies. Springer, pp. 161–176.

Lantern, 2013. Open Internet for all. Available: https://getlantern.org/en US/.

Lu, C., 2012. Network traffic analysis using stochastic grammars.

Lu, C., Schwier, J.M., Craven, R.M., Yu, L., Brooks, R.R., Griffin, C., 2013. A normalized statistical metric space for hidden Markov models. IEEE Trans. Cybern. 43 (3), 806–819.

Lyda, R., Hamrock, J., 2007. Using entropy analysis to find encrypted and packed malware. IEEE Secur. Privacy 5 (2), 40–45.

Miller, B., Huang, L., Joseph, A.D., Tygar, J.D., 2014. I know why you went to the clinic: risks and realization of HTTPs traffic analysis. In: International Symposium on Privacy Enhancing Technologies Symposium. Springer, pp. 143–163.

Mohajeri Moghaddam, H., Li, B., Derakhshani, M., Goldberg, I., 2012. Skypemorph: protocol obfuscation for Tor bridges. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, pp. 97–108.

Moshchuk, A., Gribble, S.D., Levy, H.M., 2008. Flashproxy: transparently enabling rich web content via remote execution. In: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services. ACM, pp. 81–93.

Mosko, M., 2014. Ccnx 1.0 protocol introduction.

Yawning, 2019. obfs4. [Online]. Available: https://github.com/Yawning/obfs4.

Özçelik, İ., Brooks, R.R., 2015. Deceiving entropy based dos detection. Comput. Secur. 48, 234–245.

obfsproxy, 2015. [Online]. Available: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git.

Pluggable Transports, 2020. [Online]. Available: https://www.pluggabletransports.info/.

Psiphon, 2006. [Online]. Available: https://www.psiphon3.com/en/index.html.

Rabiner, L.R., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE 77 (2), 257–286.

Rios, R., Onieva, J.A., Lopez, J., 2013. Covert communications through network configuration messages. Comput. Secur. 39, 34–46.

Roos, S., Schiller, B., Hacker, S., Strufe, T., 2014. Measuring freenet in the wild: censorship-resilience under observation. In: International Symposium on Privacy Enhancing Technologies Symposium. Springer, pp. 263–282.

Schmidt, S., Mazurczyk, W., Kulesza, R., Keller, J., Caviglione, L., 2018. Exploiting ip telephony with silence suppression for hidden data transfers. Comput. Secur. 79, 17–32.

Schuchard, M., Geddes, J., Thompson, C., Hopper, N., 2012. Routing around decoys. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, pp. 85–96.

I2P, 2003. The invisible internet project. [Online]. Available: https://geti2p.net/en/.

Ultrasurf, 2002. [Online]. Available: https://ultrasurf.us/.

Schwier, J., 2009. Pattern recognition for command and control data systems.

Schwier, J.M., Brooks, R.R., Griffin, C., 2011. Methods to window data to differentiate between Markov models. IEEE Trans. Syst. Man Cybern.Part B 41 (3), 650–663.

Schwier, J.M., Brooks, R.R., Griffin, C., Bukkapatnam, S., 2009. Zero knowledge hidden Markov model inference. Pattern Recognit. Lett. 30 (14), 1273–1280.

Smith, R.W., Knight, S.G., 2010. Predictable three-parameter design of network covert communication systems. IEEE Trans. Inf. Forensics Secur. 6 (1), 1–13.

Tsudik, G., Uzun, E., Wood, C.A., 2016. Ac3n: anonymous communication in content-centric networking. In: 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, pp. 988–991.

Vanluyten, B., Willems, J.C., De Moor, B., 2008. Equivalence of state representations for hidden Markov models. Syst. Control Lett. 57 (5), 410–419.

Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., Boneh, D., 2012. Stegotorus: a camouflage proxy for the Tor anonymity system. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, pp. 109–120.

Wiley, B., 2011. Dust: a Blocking-Resistant Internet Transport Protocol. Technical report. http://blanu. net/Dust. pdf.

Winter, P., Lindskog, S., 2012. How the Great Firewall of China is Blocking Tor. USENIX-The Advanced Computing Systems Association.

Winter, P., Pulls, T., Fuss, J., 2013. Scramblesuit: a polymorphic network protocol to circumvent censorship. In: Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society. ACM, pp. 213–224.

Wustrow, E., Swanson, C., Halderman, J.A., 2014. Tapdance: end-to-middle anticensorship without flow blocking.. In: USENIX Security Symposium, pp. 159–174.

RTPIS, 2019. Real-time power and intelliigent systems (rtpis) laboratory. [Online]. Available: http://rtpis.org/.

Wireshark, 2019. https://www.wireshark.org/.

Yao, L., Zi, X., Pan, L., Li, J., 2009. A study of on/off timing channel based on packet delay distribution. Comput. Secur. 28 (8), 785–794.

Yu, L., Schwier, J.M., Craven, R.M., Brooks, R.R., Griffin, C., 2013. Inferring statistically significant hidden Markov models. IEEE Trans. Knowl. Data Eng. 25 (7), 1548–1558.

Yu, L., Wang, Q., Barrineau, G., Oakley, J., Brooks, R.R., Wang, K.-C., 2017. Tarn: a SDN-based traffic analysis resistant network architecture. In: Malicious and Unwanted Software (MALWARE), 2017 12th International Conference on. IEEE, pp. 91–98.

Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Crowley, P., Papadopoulos, C., Wang, L., Zhang, B., et al., 2014. Named data networking. ACM SIGCOMM Comput. Commun. Rev. 44 (3), 66–73.

Zhao, H., Shi, Y.-Q., 2012. Detecting covert channels in computer networks based on chaos theory. IEEE Trans. Inf. Forensics Secur. 8 (2), 273–282.

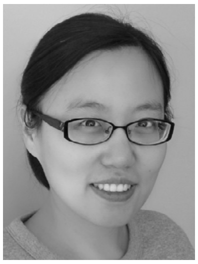
Zhong, X., Arunagirinathan, P., Ahmadi, A., Brooks, R., Venayagamoorthy, G.K., 2015. Side-channels in electric power synchrophasor network data traffic. In: Proceedings of the 10th Annual Cyber and Information Security Research Conference. ACM, p. 3.

Zhong, X., Fu, Y., Yu, L., Brooks, R., Venayagamoorthy, G.K., 2015. Stealthy malware traffic-not as innocent as it looks. In: Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on. IEEE, pp. 110–116.

GNUnet, 2002. GNU's framework for secure peer-to-peer networking. [Online]. Available: https://gnunet.org/.

SnowFlake, 2016. [Online]. Available: https://keroserene.net/snowflake/.

Jonathan G. Oakley (S'14) received the B.S. degree in electrical engineering from Clemson University, Clemson, South Carolina, where he is currently pursuing the Ph.D. Degree in computer engineering with the Holcombe Department of Electrical and Computer Engineering. His research interests include protocol mimicry, network traffic analysis, cryptocurrencies, and Markov processes.



Lu Yu received the B.S. degree in information engineering and the M.S. degree in control theory from Xi'an Jiaotong University, Xi'an, China, and the Ph.D. degree in electrical engineering from Clemson University, Clemson, South Carolina. She is currently a research assistant professor with the Holcombe Department of Electrical and Computer Engineering, Clemson University. Her research interests include cyber security and user privacy and anonymity.

Xingsi Zhong (S'15) received the B.S. degree in math from Jilin University, Changchun, China, in 2010 and the M.S. degree in computer science from the University of Texas Pan-American, Edinburg, TX, USA, in 2013. He is currently working toward the Ph.D. degree in the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA. He is currently a Research Assistant in the Real-Time Power and Intelligent Systems Laboratory. His research interests include side-channels analysis, cyber-physical system security, and computational intelligence.

Ganesh Kumar Venayagamoorthy (S'91-M'97-SM'02) is the Duke Energy Distinguished Professor of Power Engineering and Professor of Electrical and Computer Engineering and Automotive Engineering at Clemson University. Prior to that, he was a Professor of Electrical and Computer Engineering at the Missouri University of Science and Technology (Missouri S&T), Rolla, USA from 2002 to 2011. Dr. Venayagamoorthy is the Founder (2004) and Director of the Real-Time Power and Intelligent Systems Laboratory (http://rtpis.org). He holds an Honorary Professor position in the School of Engineering at the University of Kwazulu-Natal, Durban, South Africa. Dr. Venayagamoorthy received his Ph.D. degree in Electrical Engineering from the University of Natal, Durban, South Africa, in February 2002. He holds a MBA degree in Entrepreneurship and Innovation from Clemson University, SC.Dr. Venayagamoorthy's interests are in the research, development and innovation of smart grid technologies and operations, including intelligent sensing and monitoring, intelligent systems, integration of renewable energy sources, power system optimization, stability and control, and signal processing. He has published over 500 refereed technical articles. His publications are cited over 15,500 times with a h-index of 60 and i10-index of 241. Dr. Venayagamoorthy has been involved in over 70 sponsored projects in excess of US $10 million. Dr. Venayagamoorthy has given over 500 invited keynotes, plenaries, presentations, tutorials and lectures in over 40 countries to date. Dr. Venayagamoorthy is involved in the leadership and organization of many conferences including the General Chair of the Annual Power System Conference (Clemson, SC, USA) since 2013, and Pioneer and Chair/co-Chair of the IEEE Symposium of Computational Intelligence Applications in Smart Grid (CIASG) since 2011. He is currently the Chair of the IEEE PES Working Group on Intelligent Control Systems, and the Founder and Chair of IEEE Computational Intelligence Society (CIS) Task Force on Smart Grid. Dr. Venayagamoorthy has served as Editor/Guest Editor of several IEEE Transactions and Elsevier Journals. Dr. Venayagamoorthy is a Senior Member of the IEEE, and a Fellow of the IET, UK, and the SAIEE.

Richard R. Brooks (M'97 - SM'04) received the B.A. degree in mathematical sciences from Johns Hopkins University, Baltimore, MD, USA, and the Ph.D. degree in computer science from Louisiana State University, Baton Rouge. He is currently a Professor with the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. His research interests include cyber security, adaptive distributed systems and game theory.