COVERT CHANNEL DETECTION USING MACHINE LEARNING METHODS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İMGE GAMZE ÇAVUŞOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

DECEMBER 2019

Approval of the thesis:

COVERT CHANNEL DETECTION USING MACHINE LEARNING METHODS

submitted by **İMGE GAMZE ÇAVUŞOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** ______________

Assist. Prof. Dr. Hande Alemdar
Supervisor, **Computer Engineering, METU** ______________

Examining Committee Members:

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University ______________

Assist. Prof. Dr. Hande Alemdar
Computer Engineering, METU ______________

Assoc. Prof. Dr. Ertan Onur
Computer Engineering, METU ______________

Date: 02.12.2019

ABSTRACT

COVERT CHANNEL DETECTION USING MACHINE LEARNING METHODS

Çavuşoğlu, İmge Gamze
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Hande Alemdar

December 2019, 94 pages

A covert channel is a communication method that misuses legitimate resources to bypass intrusion detection systems. They can be used to do illegal work like leaking classified (or sensitive) data or sending commands to malware bots. Network timing channels are a type of these channels that use inter-arrival times between network packets to encode the data to be sent. Although these types of channels are hard to detect, they are not used frequently due to their low capacity and sensitivity to the network conditions. However, upcoming technologies like 5G and WiFi 6 offer more reliable networks with low latency, which we believe can work in favor of network timing channels and attract hackers to them. Therefore, we also believe that the detection of network timing channels is an increasingly important issue.

In this thesis, we worked with two types of network covert channels: Fixed Interval and Jitterbug. Fixed Interval defines an inter-arrival time for each symbol to be transmitted and send network packets accordingly. On the other hand, Jitterbug does not create new packet traffic; it just delays existing packets for some predefined time. Two channels are very different: Jitterbug creates traffic that is similar to the legitimate network though has lower capacity, and Fixed Interval has a very different traffic shape from the legitimate network but has higher capacity. Our work has shown it is indeed possible to detect these channels with a decision tree with four features called mean, variance, skewness and kurtosis. However, more research is needed to make this system work in the real world.

Keywords: Covert Channel, Covert Channel Detection, Machine Learning, Decision Tree

ÖZ

MAKİNE ÖĞRENMESİ METOTLARI KULLANILARAK ÖRTÜLÜ KANALLARIN TESPİTİ

Çavuşoğlu, İmge Gamze
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Hande Alemdar

Aralık 2019 , 94 sayfa

Örtülü kanallar, saldırı tespit sistemlerini atlatabilmek için meşru kaynakları istismar eden sistemlerdir. Bu kanallar hassas verileri kaçırmak veya zararlı botlara komut yollamak gibi illegal işler için kullanılabilir. Örtülü kanalların bir türü ağ üzerinde çalışan zamanlama kanallarıdır. Ağ üzerinde çalışan zamanlama kanalları, göndereceği veri için; ağ üzerinden gönderilen paketlerin arasındaki zaman farklarından faydalanır. Bu kanalları tespit etmek zor olsa da sağlayabildikleri bant genişliği düşük olduğu ve ağ üzerinde oluşan sıkıntılara duyarlı oldukları için çok fazla tercih edilmezler. Fakat 5G ve WiFi 6 gibi teknolojilerin gelişmesi, gelecekte daha güvenilir iletişim ağlarının oluşturulmasını sağlayacaktır. Biz bu durumun örtülü kanalları saldırganlar için daha cazip hale getirdiğini ve bu sebeple de örtülü kanalların tespitinin önemli bir mesele haline geldiğini düşünüyoruz.

Bu tezde iki çeşit ağ üzerinde çalışan zamanlama kanalları kullandık: Fixed Interval ve Jitterbug. Fixed Interval gönderilecek her bir sembol için ayrı bir (ya da birden fazla) zaman aralığı tanımlıyor ve paketleri aralarındaki zaman farkı tanımladığı aralıklara uyacak şekilde gönderiyor. Jitterbug ise yeni paket yaratmıyor, sadece başkası tarafından gönderilen paketleri kendi tanımladığı zaman aralıklarına göre geciktiriyor. İki kanalın farklı avantaj ve dezavantajları mevcuttur: Jitterbug meşru kaynaklardan gelen trafiğe çok benzeyen bir trafik yaratabiliyor ama ötekine göre bant genişliği daha düşük, Fixed Interval ise Jitterbug'a göre daha yüksek bant genişliğine sahip fakat yarattığı trafik meşru kaynakların yarattıklarından çok farklı. Bu çalışmada, dört istatistiki özelliği (ortalama, varyans, çarpıklık ve basıklık) kullanan karar ağaçlarından faydalanarak meşru ve örtülü (Jitterbug ve Fixed Interval kanalları) ayırt edildi. Fakat, bu sistemin gerçek dünyada çalışabilmesi için üzerinde daha fazla araştırma yapılması gerekecektir.

Anahtar Kelimeler: Örtülü Kanal, Örtülü Kanal Tespiti, Makine Öğrenmesi, Karar Ağacı

Dedicated to my mother, father and sister, who is always caring and patient with me.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Assist. Prof. Dr. Hande Alemdar. The door to Assist. Prof. Dr. Alemdar's office was always open whenever I ran into a trouble spot or had a question about my research or writing. She consistently allowed this paper to be my own work, but steered me in the right direction whenever she thought I needed it.

I would also like to thank the experts and the other students who helped with me with their ideas and other means: Assoc. Prof. Dr. Ertan Onur, Doğanalp Ergenç, Murat Coşar, Feyza Hande Güleç, Gizem Çelik, Burak Eren Dere and the rest of WINS Lab crew.

I would also like to acknowledge Assoc. Prof. Dr. Ertan Onur as the second reader of this thesis, and I am gratefully indebted to him for his very valuable comments on this thesis.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS	
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol Address
LAN	Local Area Network
WAN	Wide Area Network
PCI	Peripheral Device Interconnect
IOT	Internet of Things
CPU	Central Processing Unit
5G	Fifth-Generation
4G	Fourth-Generation
WiFi	Wireless Fidelity

CHAPTER 1

INTRODUCTION

In this chapter, we will introduce the problem, discuss our solution, contributions briefly, and give the outline of this thesis.

1.1 Motivation and Problem Definition

Consider two friends called Alice and Bob; both are in prison [3]. They can send each other messages, but a warden reads these. Alice wants to send her brilliant escape plan to Bob. She also wants to avoid the warden's suspicion so she can neither directly send nor encrypt the message. In this situation, Alice should not let the warden know she is passing a message. One way she can do it successfully is by using **covert channels**.

A covert channel is a communication method that misuses legitimate resources to bypass detection systems. The legitimate resource that is used may be the network, file system, hardware registers, or even Central Processing Unit (CPU). This attack requires two processes: an "insider" process that has the data and at least permission to send it through the legitimate resource and an "outsider" process that can access the resource at least read-only. The outside process may be a program that runs on a remote machine, or just a different thread runs on the same CPU. Also, there is a system like a firewall or intrusion detection system that checks the resource against sensitive data leakage or illegal activities like sending commands to malware bots. In our Alice and Bob analogy, Alice is the insider, Bob is the outsider process, and the warden is the detection system. For instance, Alice may be a process that has access to the intranet of a company, and Bob may be another process that exists on the external network. The warden may be an intrusion detection system that runs on the company's routers (Figure 1.1). If Alice wants to leak the company's sensitive data to Bob, she has to use the external network. Though, the warden carefully monitors the packets that are going from/to the external network. It can report the suspicious packets (like the ones contain sensitive or encrypt data) or even change the content of the packets (i.e., like removing rarely used fields of a packet). In this case, Alice has to encode her data to network packets in a way that the warden can not detect it. To do that, she can, for example, put it to a field of one or more network headers. This way, the detection system will only see one or more legitimate packets. For instance, TCP (Transmission Control Protocol) header has a field called "reserved". This field, as its name implies, is not currently used. Alice can put her data into this field without disrupting the header. Though, it is possible to detect such channels by checking the integrity of packets' fields. For our example, the warden can check whether the reserved field is zero and raise a red flag if it is not. The warden can even nullify this field altogether before passing the packet. Another option is to encode data into inter-arrival times between packets. The detection system can not detect this type of channel by just checking packets. Furthermore, attempts to prevent it may also harm on legitimate communications [4], [5]. For example, Giles and Hajek [6] propose jamming the channel, which may, in turn, cause higher packet delays. Note that, for both channels, detection is likewise possible if the encoding method is known, but it is inefficient to examine each communication channel for every network-related covert channel.

In this thesis, we focused on detecting network timing channels. These types of channels use inter-arrival times between network packets to encode the data to be sent. They assign one or more inter-arrival time(s) to each symbol to be sent, then either send or delay packets according to this mapping. For instance, if data to be sent is binary, the channel can assign $10\ milliseconds\ (ms)$ for the symbol "0" and $20ms$ for the symbol "1". Although they are hard to detect, they suffer low capacity and sensitive to issues like delays caused by network jitters, packet loss, resend, and the difference between clocks of sender and receiver, which lowers their capacity further. These problems limit networks that this attack can be implemented. For instance, implementing network timing channels in mobile networks would probably cause noticeable packet delays.

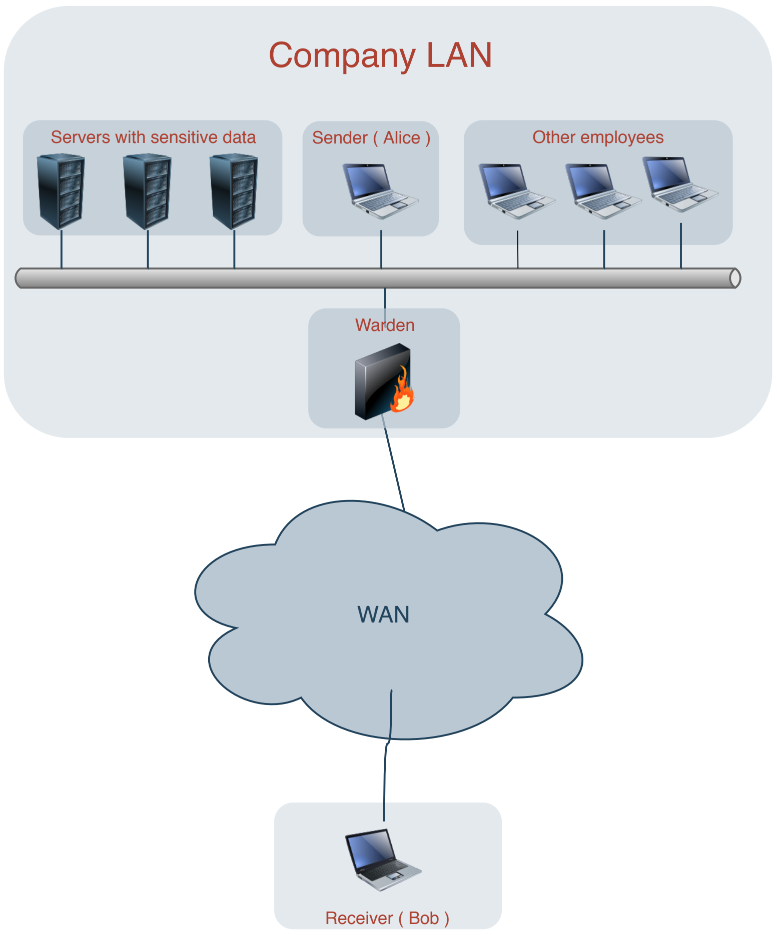


Figure 1.1: A covert channel that uses the network.

As a result, network timing channels are not used frequently. Though, we believe that upcoming newer technologies like Fifth-Generation (5G) [1] or Wireless Fidelity (WiFi) 6 [2] (aka IEEE 802.11ax-2019) can work in favor of network timing channels and attract hackers to them. These technologies offer more reliable networks with low latency even for Internet of Things (IoT) devices[3].

There are other approaches for detecting network timing channels, and some of them can detect nearly every type of network timing channel. Though, most of them are either used for specific types of covert channels or require heavy and complicated calculations. In our work, we aim to define an architecture that is both lightweight

[1] 5G [7], [8], [9] was the newest cellular network technology at the time we wrote this thesis. It offers about a hundred times higher data rate, ten times less latency, energy efficiency, and many more improvements to its predecessor Fourth-Generation (4G).

[2] IEEE 802.11ax-2019 [10], [11], [12] was the newest wireless network technology at the time we wrote this thesis. Compared to predecessor IEEE 802.11ac-2013 its throughput is about four times higher.

[3] IoT [13], [14] is a network that is created by computer and non-computer devices like a mobile phone.

and can distinguish most types of covert channels from legitimate ones. We chose a small subset of covert channels with different characteristics for our work. These are:

- Fixed Interval
- Jitterbug

Fixed Interval defines an inter-arrival time for each symbol to be transmitted and send network packets accordingly and, Jitterbug delays existing packets for some predefined time. Therefore, Jitterbug creates traffic that is similar to the legitimate network though has lower capacity, and Fixed Interval has a very different traffic shape from the legitimate network but has higher capacity.

1.2 Proposed Methods and Models

We aim for our model to work on an architecture that has shallow detectors on its front-end and more sophisticated ones on its back-end (similar to the architecture proposed by Iglesias et al. [15]). Shallow detectors filter incoming or outgoing network traffic and search for covert channels. When they detect one, they send these traffic to advanced detectors and warn operators of the system (network administrators, end-users, firewalls, etc.). Advanced detectors can then determine the traffic is covert or not for sure and its type. After that, operators of the system can take action against the suspected channel like termination, suspension, or reporting to authorities. Moreover, if the risk is too high, operators can suspend the channels even before the advanced detectors make their final decisions. Our model will be a shallow detector in this architecture.

A shallow detector must work on any network (that an attacker can create a covert channel) online or offline, be flexible and understandable. This means our model must work with switches and routers, run fast (i.e., within a minute), and adapt to changing network conditions. To fulfill these requirements, we propose using one or more decision trees with mean, variance, skewness, and kurtosis as features.

Decision trees can classify discrete target values given a set of features [16], [17]. It uses a tree structure, places rules on inner nodes, and target values on leaves. Rules are determined by the decision tree using features. For instance, a tree that uses our features can define one inner node as "mean bigger than $2ms$". Rules that are closer to the root are more generalized, whereas rules that are closer leaves tend to be more specific to a case. The decision tree is robust to errors and can be validated by statistical tests [18]. It also is one of the few machine learning methods that are understandable by humans. A decision tree can even be converted to if-then-else rules. So, apart from the detection, we also used the decision tree to determine importances of our features. As for the features, mean and variance calculate the arithmetic average and the spread of a given distribution. On the other hand, skewness and kurtosis calculate the skewness and heaviness of tails of a given distribution relative to a normal one. These four statistical features can be used to distinguish distributions. In our case, distributions are inter-arrival time series.

Routers enabling the transition between the intranet and the Internet can be used to calculate features. Decision trees can also be run on routers or a computer connected to these routers.

In our work, we implemented a detector for Fixed Interval and Jitterbug covert channels using a decision tree. The detailed descriptions of features, the decision tree, and setup are in Chapter 4.

1.3 Contributions and Novelties

Our contributions are as follows:

- We created network data for two covert channels: Fixed Interval and Jitterbug.
- We implemented a method that can detect Fixed Interval and Jitterbug covert channels. This method can work with the network packets it received from a router or switch (online). It can also detect threats in real-time and warn the operator of the network.

1.4 The Outline of the Thesis

The structure of this thesis is as follows:

- In Chapter 1 and 2, we introduce the covert channel concept, detection problem, and our propose for the solution.
- In Chapter 3, we present the legitimate, Fixed Interval and Jitterbug datasets we used for our research. We also explain how we obtained these datasets.
- In Chapter 4, we explain the algorithms and structures that we used for our solution (the decision tree and the first four moments). We also explained why and how we chose these.
- In Chapter 5, we show the experiments we performed using our method and another one from the literature. Also, in the same chapter, we discuss the experimental results and compare our method with the other one.
- In Chapter 6, we present our conclusion, criticize our work, and talk about improvements we will implement in the future.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we will firstly explain the network timing channels we used for our work. Then, we will summarize our literature research about covert channels, their types, and detection algorithms.

2.1 Covert Channels

We used the following covert channels for our work:

2.1.1 Fixed Interval

This channel defines a different inter-arrival time value for each symbol to be transmitted [19]. For example, for the two symbols (bits "0" and "1") of a binary channel, two intervals may be defined; $10ms$ for the bit "1" and $20ms$ for the bit "0". In this case, to send the bit "0", the channel waits for $20ms$, then sends a packet, and for the bit "0", it repeats the same steps using $10ms$.

Fixed Interval channels are easy to implement. Moreover, if the implementer chooses intervals appropriately, he or she can make the covert channel resistant to network errors. On the other hand, it is not hard to detect this channel due to the reason the traffic it creates is very different from the legitimate ones.

Berk et al. [19] have shown that the distributions of packet delays for legitimate and covert channels are different. Delays in the legitimate channels are closer to the mean, and delays in the covert channels are closer to the defined intervals. Based on this observation, Berk et al. presented the following formula to calculate the probability that the given inter-arrival times belong to a covert channel:

$$P_{CovChan} = \frac{C_\mu}{C_{max}},$$

where C_μ is the number of the packets with the mean (or close to the mean) inter-arrival time, and C_{max} is the number of the packets with the maximum (or close to the maximum) inter-arrival time.

2.1.2 Jitterbug

Jitterbug is a binary channel that manipulates inter-arrival times of existing packets [20]. It takes advantage of software that sends a packet with every keypress (like SSH). Jitterbug delays the press, which in turn delays the packet.

This channel does not assign inter-arrival times to symbols. Instead, it uses a parameter called timing window(w) and interprets $inter\text{-}arrival\ time \mod w = \lfloor w/2 \rfloor$ as the bit "1" and, $inter\text{-}arrival\ time \mod w = 0$ as the bit "0". For instance, if the delay between keypresses are $\{90, 150, 128, 141\}ms$, the timing window is $20ms$ and bits to send are {0, 1, 1, 0} these delays become $\{100, 150, 130, 160\}ms$ [20].

Shah et al. [20] stated that it is better to define multiple timing windows and use them rotationally (i.e., a channel may use $20ms$ for three packets, $30ms$ for four packets, and then repeat the same process). By using multiple timing windows, Jitterbug can prevent cluttering of inter-arrival times within certain ranges, thus can reduce the risk of detection.

Compared to Fixed Interval, Jitterbug is harder to implement. Also, the method it uses to encode bits is more prone to network error conditions. Nevertheless, it is also harder to detect since it creates traffic similar to the legitimate ones [21].

2.2 Related Work

The covert channel concept was introduced by Lampson [22] in the 1970s. In his work, he defined three computer programs, which he called the customer, the service, and the owner of the service. The service is a program that runs safely from the isolated environment created within the customer. It only has access to the necessary resources (input, output, etc.). Also, the system does not allow the customer to access the service's data. The owner, as the name implies, is the creator of the service. Lampson explored the ways for a service to leak data to its owner in this situation. As a result of this research, he revealed storage and covert channels: storage channels transfer the data by storing it to a medium, and covert channels transmit it by changing the runtime of processes (i.e., by changing how fast it creates an output). Later, Common Criteria (DoDI 8500.02) [23] [1] renamed covert channels as timing channels and redefined covert channels as the superclass of timing and storage channels.

Covert storage channels abuse a medium like a file, hardware, network packet, or even audio and CPU temperature to transfer data. Rutkowska [25] implemented a storage channel that encodes the data into the sequence number field of SYN (Synchronize) TCP packets[2]. Later, Scott [26] implemented a similar attack using X509 digital certificates[3]. In his work, he encoded data into the fields that will not break the integrity of the certificate. Okhravi et al. [27] showed that an attacker could create a storage channel using the register that holds the address for the initialization code of a Peripheral Device Interconnect (PCI) device[4]. They also mentioned a type of storage channel that uses a property of files to transmit data in the same study. Deshotels [28] showed how an attacker could create a covert channel using inaudible sounds that can be generated and picked by mobile devices.

Covert timing channels modulate event times for communication. For instance, IP (Internet Protocol Address) Covert Timing Channel (IPCTC) sends a network packet after waiting for some predefined interval time for the bit "1" and keeps silent for the bit "0". Another type of channel called TCP Timestamp Manipulation sends a TCP packet when the least significant bit of the packet's timestamp field matches with the bit to send [1], [29]. Okhravi et al. mention a similar type of timing channel for processes in [27] (Attack Based on the Operating System Timing Value). In such

[1] Common Criteria is a set of standards that security teams use for evaluating the security of products. This standard was known as The Orange Book (DoDD 5200.28-STD) [24] before 2002.

[2] TCP sends a SYN packet to the device that it wishes to establish a connection.

[3] A digital certificate verifies a public key belongs to a system. X.509 is a standard that defines the format of a digital certificate.

[4] PCI is an interface in the motherboard that supports plug-and-play for devices. PCI devices are devices that compatible with this interface.

channels, the sender looks at the last bit of the system time. If that bit is equal to the bit the sender will transmit, it calls the receiver. Wang et al. [30] mentioned a channel that is implemented using Simultaneous Multi-Threaded processors. This covert channel takes advantage of the ability of a processor to run multiple processes in turns. In order to transmit a symbol, the sender process executes an operation that changes its execution time (in the paper, processes use multiply to send "1" and no operation instruction to send "0"). Timing channels can even use the computer's clock to transmit data. Zander et al. [31] describe a timing channel that manipulates the CPU workload. Alterations in the CPU workload change the CPU temperature and eventually skew the clock by a rate that can be noticed by an observer [32].

There are also studies to prevent and detect covert timing channels. Researches about prevention mostly focus on either static analysis of resources or manipulating inter-arrival times. Kemmerer et al. [33] introduced a static analysis method called Shared Resource Matrix. This procedure involves creating a table for all resources in a system: resources are on the first column of the table, and access rights of all operations that use those resources are on other columns. The table even includes indirect accesses. This way, it is possible to perform a risk assessment for the system. Kang et al. [34] introduced a special architecture that uses a buffer between the network and the user. Authors claim that this architecture can reduce network timing channel capacity without decreasing throughput. Another method proposed to prevent covert channels is jamming [6]. This technique, as in [34], attempts to disrupt the covert channels by changing the inter-arrival times of the network packets. The main propose of the jamming is to disrupt covert channels without affecting the legitimate ones. It tries to achieve this by creating a distribution similar to the cumulative distribution of the inter-arrival times of legitimate channels.

On the other hand, detection techniques for covert timing channels involve capturing differences between one or more statistical properties of legitimate and covert communications. Cabuk et al. [35] introduced IPCTC and proposed two detection methods called regularity and ϵ-Similarity. Regularity divides inter-arrival times to non-overlapping groups, calculates standard deviation σ for each group, and then computes a metric using the following formula:

$$regularity = STDEV(\frac{|\sigma_i - \sigma_j|}{\sigma_i}, i < j, \forall i, j),$$

where $STDEV()$ function calculates the standard deviation of all pairwise differences between deviations of groups. ϵ-Similarity firstly sorts inter-arrival times then, calculates pairwise differences between consecutive times and, lastly, computes the percentage of differences that are smaller than a chosen ϵ value. This technique is based on the observation that for covert channels, most of the differences between inter-arrival times tend to be very small. The same does not apply to legitimate channels. The results show that both methods are successful if there is no noise in the channel, and there is only one timing interval. If more than one interval is used (for instance, $40ms$ for three packets then, $60ms$ for four packets and after that $80ms$ for five packets) regularity fails. In the case of noise, the success rate for both methods decreases.

Gianvecchio and Wang [21] used four statistical properties named Kolmogorov-Smirnov test, regularity, entropy, and corrected conditional entropy to detect three types of binary channels: IPCTC, Time-Replay Covert Timing Charmel (TRCTC) and Jitterbug. Kolmogorov-Smirnov test calculates how close two distributions are [36]. Entropy observes repetitive patterns, and corrected conditional entropy is a version of entropy that is adapted for finite data. TRCTC records and divides legitimate inter-arrival times into two groups. After that, it uses values in one group for the bit "1" and the other for the bit "0". Results show that corrected conditional entropy can detect IPCTC and TRCTC, entropy can detect Jitterbug, and Kolmogorov-Smirnov test can detect IPCTC. Regularity could not detect any channel. Authors conclude that corrected conditional entropy and entropy are enough to detect channels IPCTC, TRCTC, and Jitterbug. Shrestha et al. [37] also studied Kolmogorov-Smirnov, regularity, entropy, and corrected conditional entropy. The authors claim that it is not possible to detect covert channels using threshold values for each property presented in the paper [21] if the sampling size is one hundred inter-arrival times or less. In order to improve detection performance, they tried to use these statistical features with SVM (Support Vector Machine). SVM is a machine learning method that separates data points into two classes [38], [39]. In this case, classes are legitimate and covert. Authors tested their method with covert channels from [21] and another one called

L-bits-to-N-packets. L-bits-to-N-packets maps L number of given bits to N number of inter-arrival times [40]. Results showed that the accuracy is above 90% for IPCTC, L-bits-to N-packets, TRCTC when the sample size is two thousand inter-arrival times (this was the value authors of [21] used). The accuracy dropped to 75% and above when the authors changed the sample size to 100. For Jitterbug, accuracy increased from 33% to 48%. Darwish et al. [41] introduced another form of entropy called hierarchical entropy in their work. This technique divides the inter-arrival time list to smaller lists, then selects the one with the lowest entropy and repeats the process. It continues to do so until the currently selected list can not be divided anymore (is a legitimate channel), or the difference between previous and current selected lists exceeds a limit (is a covert channel). Hierarchical entropy successfully detected the binary form of Fixed Interval, but failed for those encoding the data in four and six bits. Later, Darwish et al. used hierarchical entropy with a big data analysis tool called MapReduce [42] to search for covert channels in within a vast amount of channels [43].

Iglesias et al. [15] proposed a series of statistical methods for all types of covert channels related to TCP/IP. To do that, they firstly captured flows of TCP packets (authors defined flow as the group of packets with the same source and destination addresses). Then they applied these methods to each TCP field that can be used for covert communication (except address fields and fields that can be checked by a packet integrity checker) and inter-arrival times of each flow. After that, they create a matrix for each flow using the results of these methods and use it for covert channel detection. Later, the same authors, along with Bernhardt, implemented eight different types of network timing channels and used methods from [15] with a decision tree [44]. The implemented channels are IPCTC, Fixed Interval, Jitterbug, Differential, Huffman Coding, TCP Timestamp Manipulation, One Threshold, and Packet Bursts. Differential adds or subtracts a predefined value from the Time-To-Live field of an IP packet to indicate the bit "1" and preserves the value of the field to indicate the bit "0" [45]. Huffman Coding encodes every symbol as a set of inter-arrival times, according to Huffman encoding [46]. One Threshold defines a threshold value and interprets inter-arrival times lower than this value as the bit "1", and higher ones as the bit "0" [47]. Packet Bursts maps a symbol to a unique number of packets [48].

For each symbol, this channel sends a predefined number of packets simultaneously and then waits for some time before sending another symbol. Results showed that the decision tree could successfully detect these channels. Later, in another work, Iglesias and Zseby [49] questioned whether network timing channels could be considered as outliers and, thus, be detected with unsupervised outlier detection algorithms. For this purpose, they used seven of the channels implemented in [44]: IPCTC, Fixed Interval, Jitterbug, Differential, Huffman Coding, One Threshold, and Packet Bursts. They created a dataset using these channels and legitimate data from MAWI Working Group Traffic Archive [50]. After that, they ran a group of selected outlier detection algorithms and supervised learning methods (using features similar to those in the paper [15]) on this dataset. Results showed that network timing channels do not have extreme characteristics that can be easily detected by outlier algorithms, though detection is possible with supervised ones. Lastly, Iglesias et al. [1] extended the network timing channel set implemented in [44] with ASCII Binary Encoding and Five-Delay Encoding. ASCII Binary Encoding encodes the input symbol as ASCII binary form, then it waits for a predefined time for the bit "0" and sends a packet right away for the bit "1" [51]. Five-Delay Encoding maps every letter of the English alphabet to a unique series five inter-arrival times [51]. They created a dataset of network packets using these channels and various types of files and folders. Then, for each flow, they calculated the following features:

- **The number of unique values (U):** This feature calculates the number of unique inter-arrival times.
- **The total number of packets in the flow ($pkts$)**
- **Mode frequency ($p(Mo)$):** This feature divides the maximum number of recurring inter-arrival time with the total number of inter-arrival times.
- **Estimation of covert byte-equivalent symbol (c):** This feature represents the estimated number of symbols sent in a flow.
- **Multimodality based on kernel density estimations (S_k):** This feature calculates the number of the peaks of the Gaussian kernel that fits inter-arrival times [15], [52].

- **Multimodality based on Pareto analysis (S_S):** This feature represents the number of sharp peaks in the inter-arrival time histogram [15], [53].
- **Average distribution width (w_s):** This feature calculates the mean of the standard deviations of Gaussians computed by the kernel.
- **Sum of autocorrelation coefficients (ρ_A):** Autocorrelation coefficients are calculated by lagging the inter-arrival time series and correlating it with the original one [54]. This feature is calculated by taking the sum of coefficients of lags between one and the number of inter-arrival times. The value of ρ_A is one for regular series.
- **Runs test (T_R):** This feature test inter-arrival times for randomness [55].
- **Sign test (T_S):** Sign test makes a pairwise comparison between inter-arrival time and its one lagged version [56]. Like T_R, it is used to test randomness in [1].
- **Kolmogorov complexity (K):** Kolmogorov complexity is the shortest program that can generate a given string [57], [58]. It is similar to entropy. For the given time series A and its compressed version B, it is estimated as $K = lengthof(A)/lengthof(B)$ in [1].
- **Hurst exponent (H_q):** This feature estimates whether a given time series tends to move in a certain direction [59]. It is used to measure self-similarity in [1].
- **Approximate entropy (H_a):** Approximate entropy is used to measure the regularity of short and noisy time series in literature [60].

Lastly, they gave these features as input to various supervised and unsupervised machine learning algorithms. Results showed that among these, K and c are most significant features, S_k, S_S, T_R and T_S are less significant, U, w_s, H_a, $p(Mo)$, $pkts$ are low significant and, ρ_A and H_q are negligible. However, removing even the negligible features dropped the detection accuracy.

CHAPTER 3

DATASET

In this thesis, we generated and used synthetic datasets for Fixed Interval and Jitterbug.

Our leakage scenario involves a network covert timing channel that is between the intranet of a company and the Internet, as shown in Figure 3.1. In the intranet, there is a "sender" (Alice) who has access to the sensitive data. The sender can also receive from or send packets to the Internet. However, these packets are collected and checked by the warden. The warden can even modify these packets' fields. For instance, the warden can fill an unused field of a network header with zeroes to prevent it from carrying covert data. The receiver can not access the intranet though he or she can receive from or send packets to the sender.

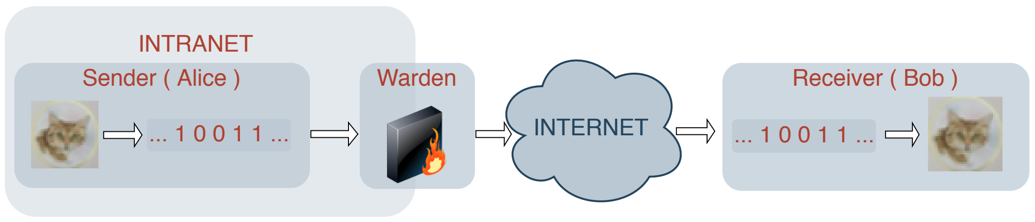


Figure 3.1: Our covert channel setup

As legitimate traffic, we used MAWI Dataset [50]. This dataset has collected samples of real-world traffic from various sample points. We selected Sample Point F for our evaluation. We chose Sample Point F because other points either were terminated a long time ago or did not have daily data. Years available to Sample Point F are from 2006 to 2019. We chose the year 2018 since it is the closest year to the writing of this thesis and has a record for each day.

We used test images from CIFAR-10 [2] in place of sensitive data. This dataset has sixty thousand colored pictures, and each picture consists of 3072 bytes (example pictures can be seen in Figure 3.2). It is suitable for generating different combinations of bits. We used one thousand images for Fixed Interval and two hundred twelve images for Jitterbug. Each channel used the first 1200 bytes of each image.

The receiver side of a covert channel has to "decode" (recover) data from inter-arrival times during or after transmission. However, there may be errors in received data because of the network conditions like jitters or packet losses. We call bit errors caused by these conditions as ”decoding error” and the percentage of erroneous bits in received data as ”decoding error rate”. In our setup, the data is one image. Therefore, we calculated decoding error rates for each image. We intended to examine the effects of errors on a covert channel's performance and detection. For this reason, we ran covert channels with various inter-arrival times. Also, we assumed latency is similar for the detector and receiver. So, we assumed that the receiver and the sender were close to each other. At first, this may seem to contradict our setup. However, it makes sense to place the receiver close to the sender so that the channel is less affected by network problems. Moreover, covert channels can use lower arrival times when the delay is small. Lower inter-arrival times increase Fixed Interval's capacity and decrease Jitterbug's chance of detection.

We used exponential distribution to simulate unexpected network delays. In the literature, this distribution is used for simulating times between events of a Poisson process. The Poisson process is defined as a collection of random events that are independent of each other [61]. The exact time between events is unknown, but the average time is known [62], [63]. We assumed packet arrivals are a Poisson process since they are random and mostly independent. Probability density function for an exponential distribution is defined as $f(x; 1/\beta) = 1/\beta e^{1/\beta}$ [64]. This function only requires one parameter, which is β. We calculated it from legitimate flows we used for each covert channel.

Each flow of the MAWI dataset and covert channels is uniquely defined with protocol, source IP, destination IP, source port and destination port tuple. We used the only TCP and UDP (User Datagram Protocol) flows.

Covert channels are generated using UDP packets. UDP packets are processed on a package basis so they can be easily generated and manipulated.

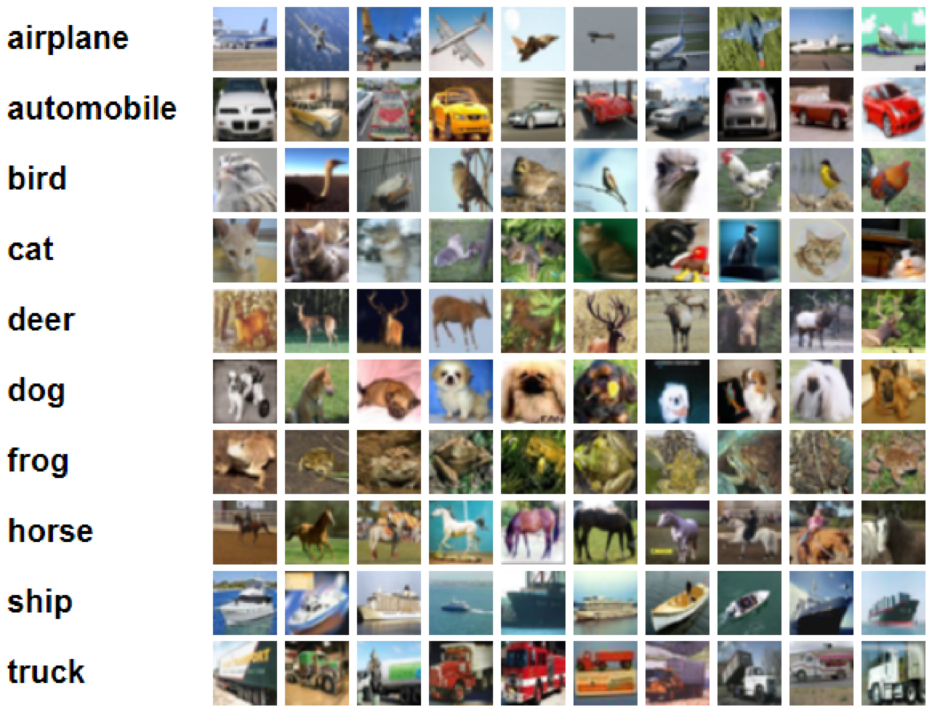


Figure 3.2: Example images from CIFAR-10 (Image taken from [2]).

3.1 Fixed Interval Dataset

For this dataset, we created one flow for each image from CIFAR-10. Covert channels sent each image as a big-endian binary bitstream. Also, channels used two inter-arrival times, and each inter-arrival time represented a bit. We configured Fixed Interval to wait for a predefined time before sending the bit "0" and double of it before sending the bit "1". For instance, if the predefined inter-arrival time is $50ms$, $50ms$ is used to transmit a "1" bit and $100ms$ is used to transmit a "0" bit.

We selected flows with TCP or UDP packets from a random day (8 December 2018 to be precise) of the MAWI dataset as legitimate flows. Packet type and content are not important since only inter-arrival times are processed. We used UDP and TCP due to the reason these are commonly used protocols.

We used following inter-arrival times to send all images: $0.018 - 0.036ms$, $0.035 -$

$0.070ms$, $0.065-0.130ms$, $0.100-0.200ms$ and $0.185-0.370ms$. These inter-arrival times show a pattern; channels with higher inter-arrival times have lower decoding error rates but also have lower capacity. We also will show the effects of inter-arrival times to detection later. Table 3.1 shows the decoding error rate and capacity for each inter-arrival time. We also calculated β as $0.018ms$.

Table 3.1: Fixed Interval Dataset Properties

Inter-Arrival Times (ms)	Decoding Error Rate	Capacity (Kilobits per second)
0.018-0.036	30%	56
0.035-0.070	20%	29
0.065-0.130	10%	15
0.100-0.200	5%	10
0.185-0.370	1%	5

3.2 Jitterbug Dataset

For this dataset, we used SSH flows that we extracted from the legitimate dataset. Jitterbug requires keypresses, so we extracted keypresses from our flows. We filtered keypresses using inter-arrival times since it was not possible to understand which package belongs to a keypress by looking at the contents of the packages. We accepted inter-arrival times equal to or bigger than $30ms$ as a keypress. This heuristic is based on observations from the research made by Killourhy and Maxim [65] and the book written by Tyeito et al. [66]. The first work contains a dataset that keystroke-timing of fifty one people that type a password that has nine characters four hundred times. It includes various of timing data, including delta times between keypresses. The average value for minimum times for each delta time between keypresses is approximately $25ms$. The other work shows that 1% of SSH text sessions have $32ms$ inter-arrival times (Tveito et al., 2009, p. 207). Therefore, we set an optimistic lower limit $30ms$ and extracted inter-arrival times equal to or bigger than that. The images are converted to big-endian bits and sent one after another through multiple sessions. We assumed that the receiver knows the size of each image beforehand. Also, as defined by Shah et al. [20], Jitterbug interprets $inter\text{-}arrival\ time \mod w = \lfloor w/2 \rfloor$ as the bit "1" and $inter\text{-}arrival\ time \mod w = 0$ as the bit "0".

We used following timing windows to send all images: $1ms$, $3ms$, $5ms$, $7ms$, and $9ms$. Like Fixed Interval, channels with higher inter-arrival times have lower decoding. However, unlike Fixed Interval, the capacity of Jitterbug depends only on the number of keypresses. As a result, the capacity is independent of the timing window and different for each flow. All timing windows have the same capacity. Figure 3.3 shows channel capacities for all flows. The average capacity of all flows is approximately 1 bit per second.

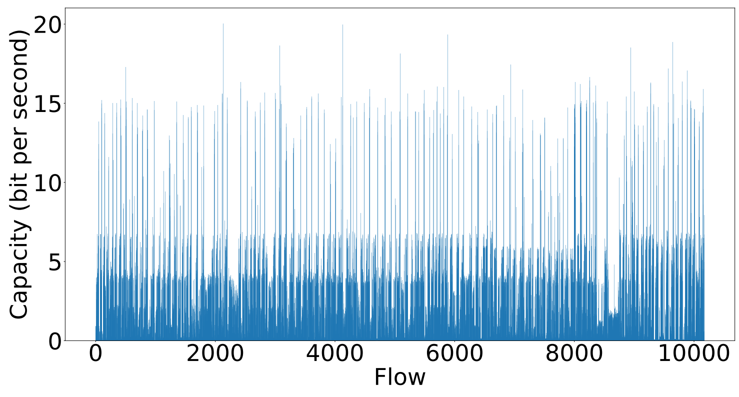


Figure 3.3: Channel capacities for all flows.

Table 3.2 shows the decoding error rate for each inter-arrival time. We also calculated β as $0.25ms$.

Table 3.2: Jitterbug Dataset Properties

Timing Window (ms)	Decoding Error Rate
1	38%
3	15%
5	7%
7	4%
9	2%

CHAPTER 4

PROPOSED METHOD AND FEATURES

Our goal is to build a flexible, fast running and understandable detector, as we explained in Section 1.2. We decided to use machine learning methods to achieve flexibility. Based on past studies, we selected a few of them as candidates: neural network, SVM, unsupervised learning algorithms, ensemble methods, and decision tree.

Neural networks are composed of layers and inter-layer connections. Each connection is associated with a number that is called "weight". Weights determine the output of the network. During the training phase, neural networks adjust weights so that results are close to expected ones. Neural networks are successfully used in many tasks including anomaly and fraud detection (anomaly and fraud detection are similar to the covert channel detection; all of them look for anomalies). Unfortunately, the model build by neural networks is not easily understandable and often considered as black-box.

SVM is used for data sets that can be divided into two separate categories. It treats the input data as points on hyperspace and draws a hyperplane to separate this space into two categories [37]. Unfortunately, SVM also does not produce an understandable model and can not handle multiple target categories. Moreover, SVM tries to use a single hyperplane to separate the space, and the data may not always be categorized using a single hyperplane.

Unsupervised algorithms do not require labeled input. These algorithms split the input data into categories using the statistical properties of the data. This means we do not need to label the training data by hand. Unsupervised algorithms can also handle multiple target classes. Although, it is not always clear why an unsupervised algorithm categorised an input as a class. Moreover, Iglesias et al. have shown that unsupervised outlier detection algorithms are not efficient covert channel detectors.

Ensemble methods combine multiple machine learning algorithms to gain better prediction performance. These methods are less prone to overfitting than single machine learning algorithms. Though ensemble methods also require more time and space. We did not use ensemble methods because we only have four features and need a model that requires less space and time.

The decision tree classifies given features using a tree structure. It is one of the few machine learning methods that build a model understandable by humans. Moreover, the decision tree can be used for feature selection. Also, Iglesias et al. claimed that they achieved the best detection performances with a decision tree [1]. However, decision tree is prone to overfitting thus must be used carefully. We decided to use the decision tree because it is appropriate for the data with categorical target classes, and the model it created is easily understandable.

When selecting features, we firstly evaluated the following: Kolmogorov-Smirnov test, regularity, ϵ-Similarity, entropy and the ones defined by Iglesias et al. in [1]. Although all of them were used in previous studies, Jitterbug was only detected by the latter. However, when we tried to implement features from [1], we ran into some problems. Firstly, calculations took a long time. Secondly, it was not clear how they computed some features. Since our detector must run fast, the working time of calculations is important. Also, our detector requires retraining when network conditions change. It is crucial to finish the training as fast as possible and bring the system to online. Moreover, if the detector fails, maintainers would want to know what is happening on the inside. Debugging the system would be faster and easier with fewer features. Features defined by Iglesias et al. are specialized for detecting various types of covert channels and are better for advanced detection. For these reasons, we looked for other options and found the first four moments (mean, variance, skewness and kurtosis). These show mean, spread and shape of a distribution. Covert channels we encountered either do not agree with the shape of the legitimate traffic (i.e., Fixed Interval) or have to have a higher mean to compensate network delays (i.e., Jitterbug). Therefore, we believe the first four moments can discriminate most types of covert channels.

Detailed descriptions of the decision tree and the first four moments are given the next sections:

4.1 Decision Tree

Decision tree is used for classifying discrete target values given a set of features. It uses a tree, places rules on inner nodes, and target values on leaves. Rules are determined by the decision tree using features. For instance, a tree that uses our features can define an inner node as "mean bigger than $2ms$". Rules are closer to the root are more generalized, whereas rules are closer to leaves tend to be more specific to a case. The decision tree is robust to errors and can be validated by statistical tests [18]. Moreover, it is easily understandable and can even be converted to if-then-else rules.

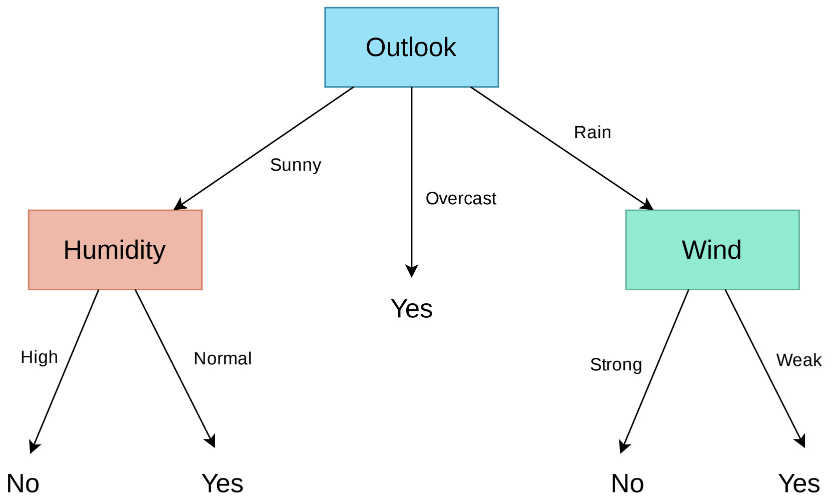


Figure 4.1: A decision tree example. Nodes are features, connections are values for each feature, and leaves are classes.

The decision tree algorithm uses heuristic methods to construct the tree. Heuristic methods are called splitting criteria. The most frequently used ones are Gini Impurity and Information Gain. Gini Impurity calculates the probability that a given classification is wrong. It is calculated as:

$$GiniImpurity = 1 - \sum_{k=1}^{N} p(class_i)$$

where N is the number of target classes, $class_i$ is the ith class, and $p(class_i)$ is the probability that a sample is in $class_i$. Information Gain measures how much a given feature contributes to classification. It is computed as:

$$H(E) = -\sum_{k=1}^{N} p(class_i) * log_2 p(class_i)$$

$$H(E|x) = -\sum_{k=1}^{N} p(class_i|x) * log_2 p(class_i|x)$$

$$InformationGain(x) = H(E) - H(E|x)$$

where $InformationGain(x)$ is the information gain of a given feature x, $H(E)$ is the entropy of outcomes, and $H(E|x)$ is the entropy of outcomes given a feature. Both Information Gain and Gini Impurity is 0.0 if all samples belong to a class and 0.5 if the number of samples in all classes are equal. The splitting criterion a decision tree will use is given to it by the user.

The decision tree algorithm creates the tree from top to bottom during the training phase. During the construction of a level, the algorithm firstly computes a value for all features using the given splitting criterion. After that, it selects the feature with the highest (Information Gain) or the lowest value (Gini Impurity) as a node. The algorithm continues to create new nodes until values for all features become zero (all training samples are classified). After the training phase, the tree remains unchanged.

Sometimes a decision tree specializes for the given training data and loses its generalization abilities. This problem is called overfitting. Generally, decision trees are pruned during or after the training phase to avoid overfitting. Pruning removes some branches of the decision tree without decreasing the accuracy too much. Other options include limiting the maximum depth of the tree or the maximum number of samples that are required to create a leaf node [18].

4.2 The First Four Moments

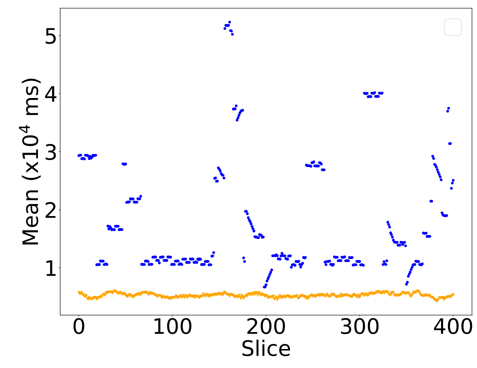
We utilized four features called mean, variance, skewness and kurtosis for our purpose. Mean calculates the arithmetic average, variance calculates the spread, skewness calculates the skewness and kurtosis calculates the amount of data in tails of a given distribution. These are called the first four moments of statistics [67], [68], [69].

Each of these features is calculated using last predefined consecutive inter-arrival times (for instance, the last fifty consecutive inter-arrival times). That is, we defined a "window" which takes ten, twenty, or forty consecutive inter-arrival times and computes features using them (we will call "window" as "slice" in order to distinguish it from Jitterbug's timing window parameter). The number of consecutive inter-arrival times to be used may be from two to the number of packets minus one.

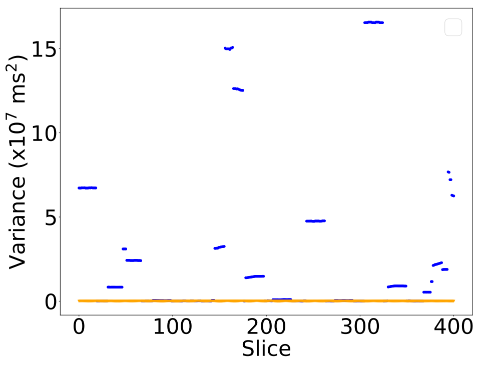
Some mean, variance, skewness and kurtosis values for a legitimate and two different Fixed Interval channels are given in Figure 4.2 and 4.3. These are calculated using twenty inter-arrival times. Figure 4.2 and 4.3 show that the mean and the skewness are good candidates for detecting a Fixed Interval channel. Since the shape of the legitimate and Fixed Interval's are different, the skewness can distinguish them. The mean becomes the main distinguishing feature if inter-arrival times of Fixed Interval are significantly smaller or bigger than the mean of the legitimate channel. This situation is present in Figure 4.2. A decision tree also supports these results. Features weighted according to their contribution in classification are shown in Figure 4.4. These weights are called "feature importances". Feature importances are calculated using Gini Importance. Gini Importance is an extended version of the Gini Impurity. It takes a value between zero and one. Features that contribute more to the classification have higher Gini Importance [64]. Figure 4.4b shows that the Fixed Interval with $0.065ms$ and $0.130ms$ inter-arrival times is detected using the skewness. This is due to the fact that Fixed Interval and the legitimate channel have similar means, as shown in the 4.3a. On the other hand, the Fixed Interval with $0.035ms$ and $0.070ms$ inter-arrival times is detected using the mean (Figure 4.4a). Figure 4.2a shows that the mean is slightly different for the covert and the legitimate channels. Another point to note is that the values of the legitimate channel's features are more scattered than

Fixed Intervals' as seen in Figure 4.2, 4.3. This shows the legitimate channels tend to be more chaotic than Fixed Intervals'.

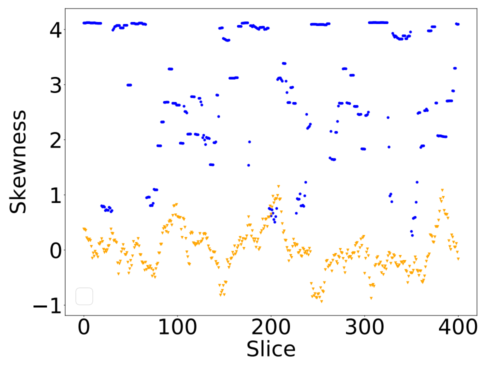
Figure 4.5, 4.6, and 4.7 shows major differences between Jitterbug and an SSH legitimate channel are the mean and the variance. Jitterbug must use timing windows that are high enough to avoid unexpected delays. As a result, Jitterbug has higher inter-arrival times. Also, inter-arrival times that Jitterbug creates are multiples (or multiple plus the half) of the timing window. Consequently, the variance of the inter-arrival times are smaller and more regular for Jitterbug channels. So it makes sense that the mean and the variance are the main distinguishing features. The shape of Jitterbug is similar to the legitimate channel (Figure 4.5c, 4.6c, 4.5d and 4.6d). As a result, values of skewness and kurtosis are more close to each other and less frequently used (Figure 4.7). Though, the decision tree uses all features more or less for the detection of Jitterbug.



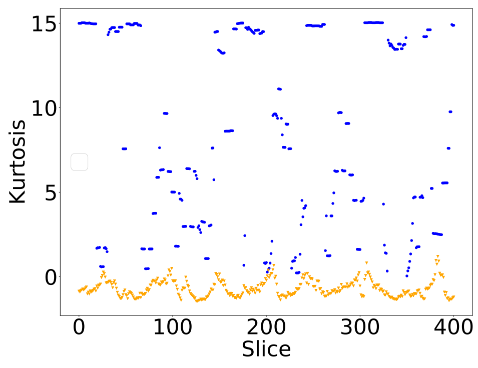
(a) Mean



(b) Variance

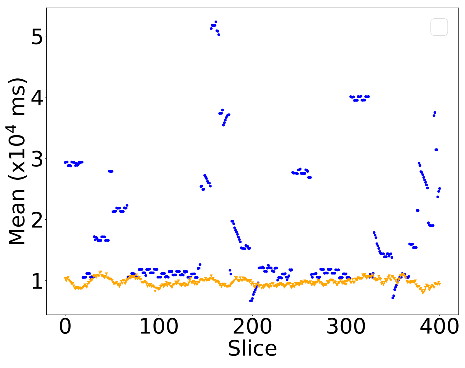


(c) Skewness

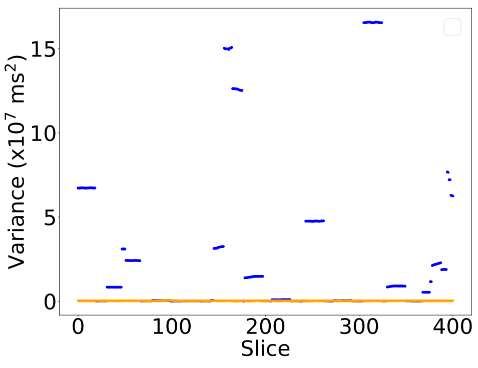


(d) Kurtosis

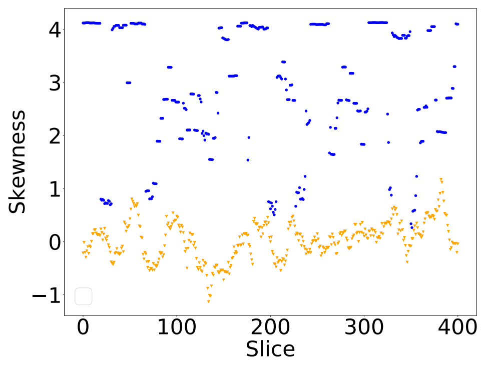
Figure 4.2: Some mean, variance, skewness and kurtosis values for a legitimate (blue dots) and a Fixed Interval (orange dots) channel. These are calculated using twenty consecutive inter-arrival times. Inter-arrival time for the covert channel is $0.035ms$ (for the bit "1" and $0.070ms$ for the bit "0").



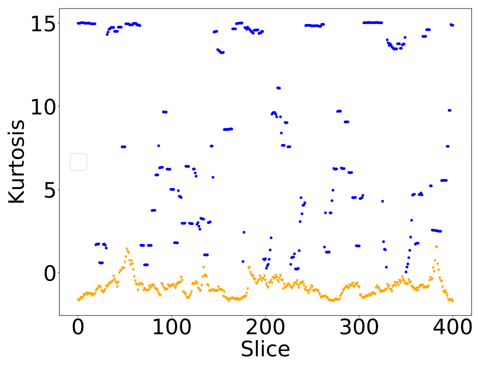
(a) Mean



(b) Variance

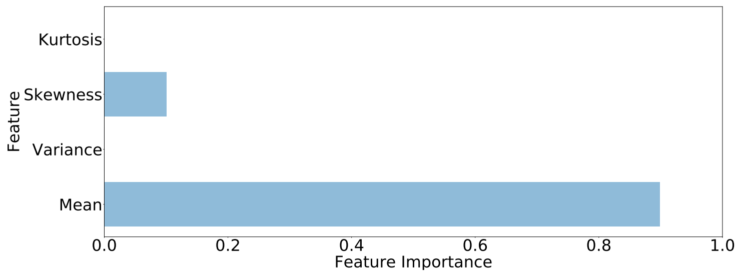


(c) Skewness

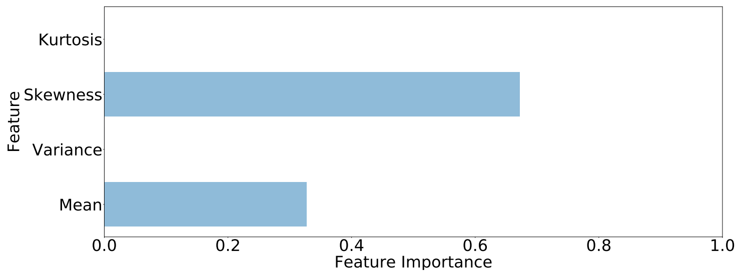


(d) Kurtosis

Figure 4.3: Some mean, variance, skewness and kurtosis values for a legitimate (blue dots) and a Fixed Interval (orange dots) channel. These are calculated using twenty consecutive inter-arrival times. Inter-arrival time for the covert channel is $0.065ms$ (for the bit "1" and $0.130ms$ for the bit "0").

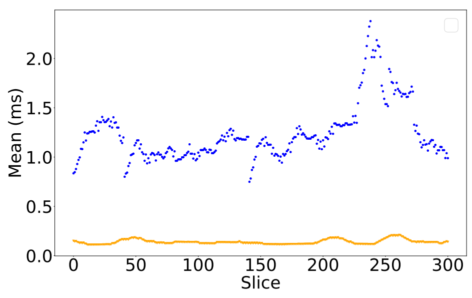


(a) Feature importances for a Fixed Interval channel with $0.035ms$ and $0.070ms$ inter-arrival times.

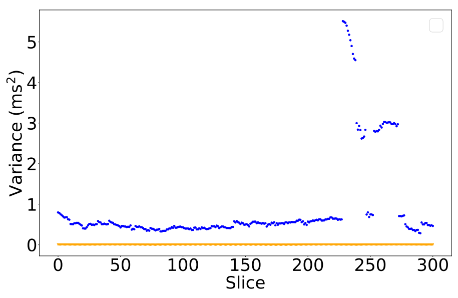


(b) Feature importances for a Fixed Interval channel with $0.065ms$ and $0.130ms$ inter-arrival times.

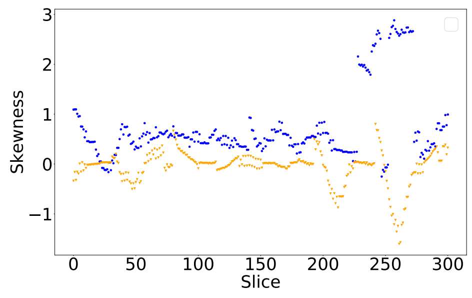
Figure 4.4: Feature importances for two Fixed Intervals. Each importance takes a value between zero and one. Features that contribute more to classification have higher importance.



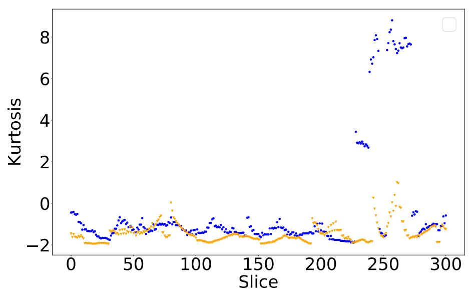
(a) Mean



(b) Variance

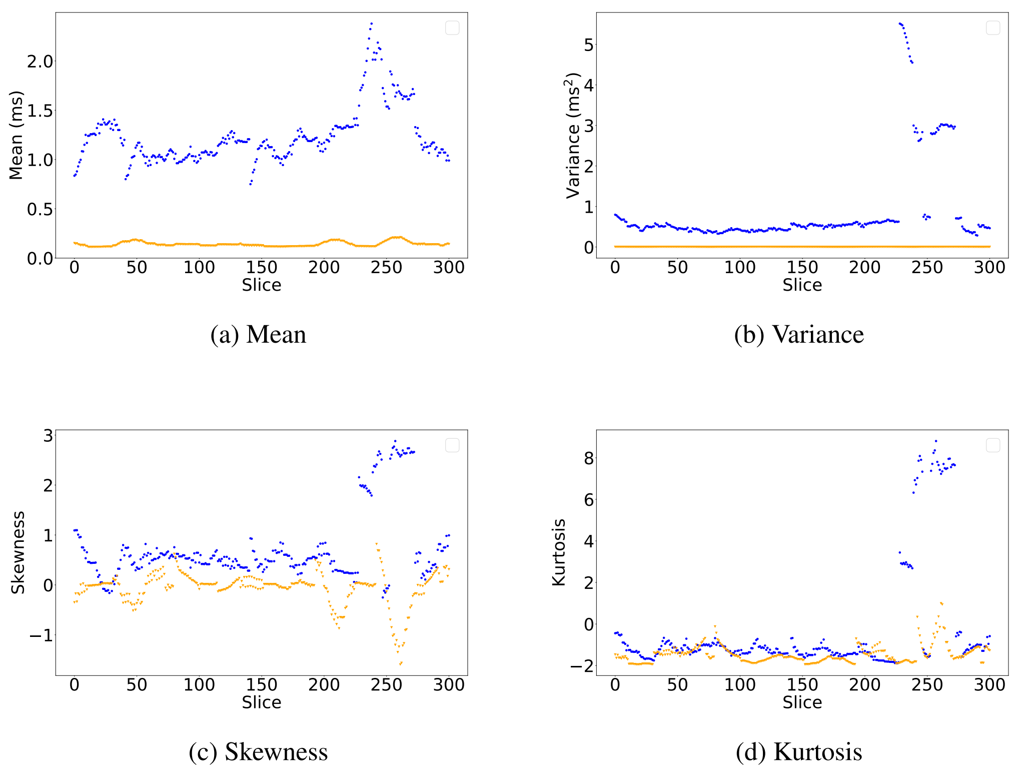


(c) Skewness



(d) Kurtosis

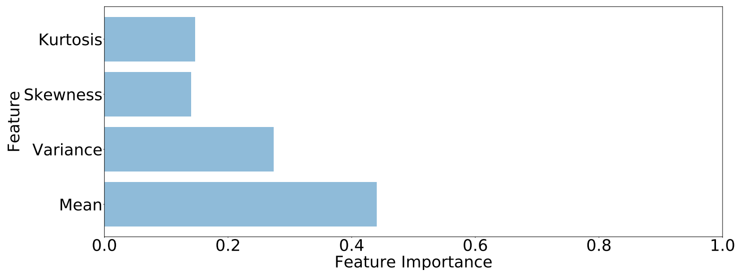
Figure 4.5: Some mean, variance, skewness and kurtosis values for a legitimate (blue line) and a Jitterbug (orange line) channel. These are calculated using twenty consecutive inter-arrival times. The timing window of the covert channel is $1ms$.



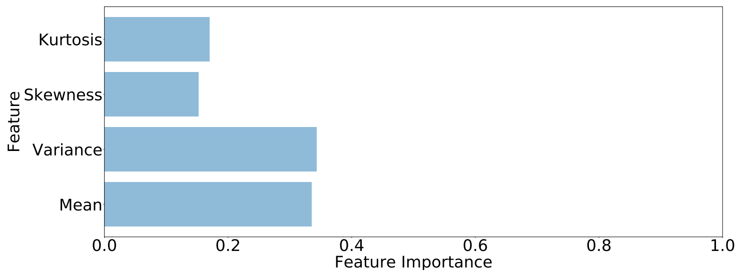
(a) Mean (b) Variance

(c) Skewness (d) Kurtosis

Figure 4.6: Some mean, variance, skewness and kurtosis values for a legitimate (blue line) and a Jitterbug (orange line) channel. These are calculated using twenty consecutive inter-arrival times. The timing window of the covert channel is $5ms$.



(a) Feature importances for a Jitterbug with $1ms$ time window.



(b) Feature importances for a Jitterbug with $5ms$ time window.

Figure 4.7: Feature importances for two Jitterbugs. Each importance takes a value between zero and one. Features that contribute more to classification have higher importance.

CHAPTER 5

EXPERIMENTS AND DISCUSSIONS

In this section, we will first explain the experiments we performed to test our hypothesis explained in Chapter 4. After that, we will discuss results of these the experiments.

5.1 Experiments

In these experiments, we wished to see how accuracy, precision, recall and F1 scores [70], [71] of detection changes for different configurations of covert channels, features, and decision trees. We also aimed to find the minimum number of network packets we could detect a covert channel with high probability. In other words, we aimed to see how soon can we detect a covert flow. We used Fixed Interval, Jitterbug, and legitimate flows from our dataset for experiments.

We performed a different experiment for each important parameter that covert channels, features, and a decision tree have. For each experiment, we used only covert flows with the same inter-arrival times (time window for Jitterbug) and legitimate flows. Before running experiments, we calculated features using each flow in our dataset. After that, we brought the features with the same configurations together to create a list. We did the same for legitimate flows. Then, we ran each experiment using a decision tree and features we calculated. In order to avoid biases, we shuffled each feature list during an experiment. We also shuffled the legitimate and the covert feature lists we intend to use together before giving them to the decision tree. Furthermore, we kept the number of covert and legitimate features given to the tree equal to prevent the decision tree from deciding in favor of the class that had more samples. Also, we created a new tree for each run. Each experiment was performed twenty times with five-fold cross-validation. We calculated the arithmetic average of the results from all runs to find the final result.

In order to find the minimum number of network packets that detection can be made, we calculated features from a flow using ten, twenty and forty consecutive inter-arrival times. As we mentioned in Chapter 4, we call a group of these inter-arrival times a "slice". We divided each flow into slices. Each slice other than the first starts with the second element of the previous group. For example, if we were to use three consecutive inter-arrival times and our inter-arrival times were $\{0.56, 1.01, 2.13, 0.6, 0.51\}ms$; the first group of features would be calculated using $\{0.56, 1.01, 2.13\}ms$, the second one would be calculated using $\{1.01, 2.13, 0.6\}ms$ and the third one would be calculated using $\{2.13, 0.6, 0.51\}ms$. We created three different feature lists using ten, twenty, and forty slices for covert flows with the same configurations. Three different lists were also created for legitimate channels similarly. In each experiment, we used features calculated using the same slice. Moreover, we calculated the results of each experiment according to the slice used in that experiment.

We also modified the following parameters for the decision tree to improve detection performance or observe differences between performances: the splitting criterion that tree uses, the minimum number of samples that a leaf requires (MSL), and the splitter. As we mentioned in Chapter 4, the splitting criterion is used to construct a tree. It is one of the factors that determine the structure of the tree. We used Information Gain and Gini Impurity splitting criteria for our experiments. The MSL specifies at least how many samples must lead to a node in order for that node to be defined as a leaf. For instance, if the MSL is five samples for a decision tree, then all leaves should be accessible by at least five or more samples. The MSL is a parameter that affects both the structure and the runtime of the tree. Smaller MSL values lead to shorter runtimes but lower prediction performance. On the other hand, bigger MSL values have higher prediction scores but cause longer runtimes and are more susceptible to overfitting. The splitter specifies how the tree will select a node. Possible options for this parameter are the best or the random. The best selects the most optimal feature using methods described in Section 4.1. The random selects a feature randomly. The best option has higher accuracy than the other. Though, it runs significantly slower. Moreover, selecting options randomly instead of selecting the best may have an positive impact on overfitting.

In addition to these, we aimed to see how our feature behaves for a flow. To do that, we trained a decision tree using the two-thirds of legitimate and covert channels' feature lists. Then, we randomly selected ten legitimate and ten covert flows that do not have features on these lists. After that, we gave each flow's features one by one to the tree and got its decision. For every inter-arrival time and timing window configuration and every type of the flow (legitimate or covert), we created a new decision tree. We also tested legitimate flows with trees that are created using different configurations of covert channels. For each flow, we calculated how many times the right and wrong decisions were made. We also calculated how many times the decision of the tree has changed. The latter helped us understand how accurate and stable the decisions of the tree were.

Lastly, we compared our features with other features in the literature that can detect Fixed Interval and Jitterbug. For this purpose, we chose features defined by Iglesias et al. in [1]. These are:

- **The number of unique values (U):** This feature calculates the number of unique inter-arrival times.
- **The total number of packets in the flow ($pkts$)**
- **Mode frequency ($p(Mo)$):** This feature divides the maximum number of recurring inter-arrival time with the total number of inter-arrival times.
- **Estimation of covert byte-equivalent symbol (c):** This feature represents the estimated number of symbols sent in a flow.
- **Multimodality based on kernel density estimations (S_k):** This feature calculates the number of the peaks of the Gaussian kernel that fits inter-arrival times.
- **Multimodality based on Pareto analysis (S_S):** This feature represents the number of sharp peaks in the inter-arrival time histogram.
- **Average distribution width (w_s):** This feature calculates the mean of the standard deviations of Gaussians computed by the kernel.

- **Sum of autocorrelation coefficients (ρ_A):** Autocorrelation coefficients are calculated by lagging the inter-arrival time series and correlating it with the original one. This feature is calculated by taking the sum of coefficients of lags between one and the number of inter-arrival times. The value of ρ_A is one for regular series.
- **Runs test (T_R):** This feature test inter-arrival times for randomness.
- **Sign test (T_S):** Sign test makes a pairwise comparison between inter-arrival time and its one lagged version. Like T_R, it is used to test randomness in [1].
- **Kolmogorov complexity (K):** Kolmogorov complexity is the shortest program that can generate a given string. It is similar to entropy. For the given time series A and its compressed version B, it is estimated as $K = lengthof(A)/length\ of(B)$ in [1].
- **Hurst exponent (H_q):** This feature estimates whether a given time series tends to move in a certain direction. It is used to measure self-similarity in [1].
- **Approximate entropy (H_a):** Approximate entropy is used to measure the regularity of short and noisy time series in literature. We used a different version of entropy called sample entropy [72] in our calculations. Compared to the approximate entropy, the results of sample entropy is less erroneous.

We were able to implement all but the c feature. We could not be able to implement c due to the reason its calculation method was not specified. Experiments with these features had the same steps as the ones that we did with our features.

We performed the experiments using Python programming language. Moreover, we used Python's libraries for mean, variance, skewness, kurtosis, decision tree, accuracy, precision, recall, and F1 score [64].

5.1.1 Experiments with Fixed Interval

We performed experiments with Fixed Interval channels using combinations of the following parameters:

- **Inter-Arrival Times:** This parameter determines inter-arrival times that covert channels use. The values used for these experiments are $0.018 - 0.036ms$, $0.035 - 0.070ms$, and $0.065 - 0.130ms$.

- **Splitting Criterion:** The splitting criterion specifies the measurement method that is used to determine tree nodes. The values used for these experiments are Gini Index and Information Gain.

- **Splitter:** The splitter specifies how the decision tree will select a node. The values used for these experiments are best and random.

- **MSL:** This parameter specifies at least how many samples must lead to a node in order for that node to be defined as a leaf. The values used for these experiments are five million, five hundred thousand, and five thousand samples. We did not use values below five thousand samples to reduce the risk of overfitting.

- **Slice:** The "window" that we used to calculate features. The values used for these experiments are ten, twenty and, forty inter-arrival times.

For the experiments with the features in [1], we set **inter-arrival times** as $0.018 - 0.036ms$, $0.035 - 0.070ms$, and $0.065 - 0.130ms$, **the splitting criterion** as Gini Impurity, **the splitter** as best, **the MSL** as five thousand samples and **the slice** as forty inter-arrival times.

Before running each experiment, we used z-scaling to normalize features. We did this in order to obtain slightly better results.

5.1.2 Experiments with Jitterbug

We performed experiments with Jitterbug channels using combinations of the following parameters:

- **Timing Window:** This parameter determines the timing windows that covert channels use. The values used for these experiments are $1ms$, $3ms$, $5ms$, $7ms$, and $9ms$.

- **Splitting Criterion:** The splitting criterion specifies the measurement method that is used to determine tree nodes. The values used for these experiments are Gini Index and Information Gain.

- **Splitter:** The splitter specifies how the decision tree will select a node. The values used for these experiments are best and random.

- **MSL:** This parameter specifies at least how many samples must lead to a node in order for that node to be defined as a leaf. The value used for these experiments is ten samples.

- **Slice:** The "window" that we used to calculate features. The values used for these experiments are ten, twenty and, forty inter-arrival times.

For the experiments with the features in [1], we set **the timing window** as $1ms$, $3ms$, $5ms$, $7ms$ and $9ms$, **the splitting criterion** as Gini Impurity, **the splitter** as best, **the MSL** as ten samples and **the slice** as forty inter-arrival times.

5.2 Results and Discussion

Section 5.2.1 shows results for the experiments with Fixed Interval channels. Section 5.2.2 shows results for the experiments with Jitterbug channels. In Section 5.2.3, we discuss these results.

In each section, we presented detection results and compared these results with covert channel decoding performances. We also put feature importances for each samples and slices. Moreover, we presented calculation times one slice of each covert channel.

5.2.1 The Results for Fixed Interval

In order to make detection results more understandable, we present them in six different tables. Each table shows results for a covert channel configuration and a splitting criterion. Each table shows the accuracy, precision, recall, and F1 score for each **splitter**, **MSL**, and **slice** combination.

We also gave results related to flow detection, feature importances, calculation times and, comparisons with covert channel decoding performances in this section.

Table 5.1: The results for channels with $0.018ms$ and $0.036ms$ inter-arrival times. The splitting criterion is Gini Impurity.

Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	86.95	79.35	99.90	88.45
		20	90.77	84.51	99.83	91.54
		40	93.24	88.23	99.80	93.66
	500000	10	97.43	95.76	99.26	97.48
		20	98.18	99.57	96.78	98.16
		40	98.56	99.91	97.20	98.54
	5000	10	98.83	98.31	99.38	98.84
		20	99.52	99.31	99.73	99.52
		40	99.84	99.80	99.89	99.84
Random	5000000	10	57.16	39.98	51.18	43.12
		20	57.68	37.49	59.68	45.22
		40	67.90	54.79	72.95	61.19
	500000	10	74.73	68.87 18	85.99	76.02
		20	81.90	76.52	94.41	84.14
		40	85.33	79.68	94.82	86.10
	5000	10	88.87	84.76	95.18	89.60
		20	92.06	88.81	97.90	92.85
		40	94.25	91.45	98.91	94.80

Table 5.2: The results for channels with $0.018ms$ and $0.036ms$ inter-arrival times. The splitting criterion is Information Gain.

Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	86.89	79.24	99.97	88.41
		20	90.70	84.33	99.97	91.49
		40	93.16	87.97	99.98	93.59
	500000	10	97.52	98.68	96.32	97.48
		20	98.43	99.54	97.30	98.41
		40	98.09	99.93	96.25	98.05
	5000	10	98.81	98.34	99.30	98.82
		20	99.51	99.23	99.80	99.52
		40	99.84	99.76	99.93	99.84
Random	5000000	10	57.76	39.39	57.22	45.49
		20	63.69	50.57	67.33	56.29
		40	61.58	43.03	64.97	50.83
	500000	10	74.89	68.43	87.79	76.51
		20	79.47	73.28	90.42	80.29
		40	87.89	83.71	95.28	88.75
	5000	10	88.53	84.37	94.94	89.29
		20	93.83	90.87	97.71	94.11
		40	96.14	94.55	98.80	96.46

Table 5.3: The results for channels with $0.035ms$ and $0.070ms$ inter-arrival times. The splitting criterion is Gini Impurity.

Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	84.99	76.96	99.90	86.94
		20	87.77	80.48	99.73	89.08
		40	93.52	89.54	98.56	93.83
	500000	10	98.31	99.45	97.16	98.30
		20	99.36	99.78	98.94	99.36
		40	97.97	99.97	95.97	97.93
	5000	10	99.49	99.18	99.80	99.49
		20	99.80	99.70	99.90	99.80
		40	99.93	99.90	99.96	99.93
Random	5000000	10	57.92	39.73	61.69	47.46
		20	70.34	58.36	75.89	64.81
		40	65.59	46.60	66.55	54.01
	500000	10	76.61	71.20	87.95	78.13
		20	82.62	75.99	92.58	83.09
		40	88.53	83.87	97.67	89.87
	5000	10	90.13	86.37	95.74	90.75
		20	93.32	90.03	98.30	93.82
		40	97.84	96.61	99.28	97.90

Table 5.4: The results for channels with $0.035ms$ and $0.070ms$ inter-arrival times. The splitting criterion is Information Gain.

Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	84.94	76.86	99.96	86.90
		20	87.62	80.16	99.98	88.98
		40	93.35	88.87	99.12	93.71
	500000	10	98.49	99.41	97.56	98.48
		20	99.74	99.64	99.84	99.74
		40	98.30	99.96	96.64	98.27
	5000	10	99.48	99.22	99.75	99.48
		20	99.78	99.69	99.88	99.78
		40	99.93	99.90	99.96	99.93
Random	5000000	10	53.34	35.01	53.38	40.43
		20	64.80	51.00	67.82	56.73
		40	66.53	50.14	72.23	58.33
	500000	10	79.99	76.96	86.63	81.15
		20	79.19	73.22	96.00	82.43
		40	87.49	82.10	97.79	88.95
	5000	10	90.89	87.43	95.93	91.42
		20	93.65	90.87	98.04	94.12
		40	96.91	94.95	99.30	97.03

Table 5.5: The results for channels with $0.065ms$ and $0.130ms$ inter-arrival times. The splitting criterion is Gini Impurity.

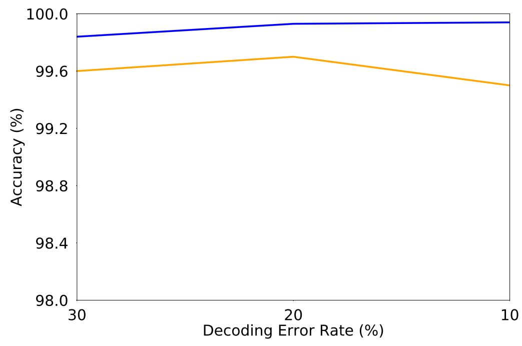
Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	81.60	75.17	94.36	83.68
		20	89.04	83.75	96.89	89.84
		40	94.26	90.74	98.58	94.50
	500000	10	98.12	97.20	99.11	98.14
		20	98.36	99.81	96.89	98.33
		40	97.76	99.97	95.55	97.71
	5000	10	99.49	99.20	99.78	99.49
		20	99.79	99.64	99.94	99.79
		40	99.94	99.91	99.97	99.94
Random	5000000	10	58.89	42.31	58.55	47.65
		20	60.30	41.87	62.19	48.88
		40	60.03	42.04	66.87	50.77
	500000	10	79.25	73.20	94.10	82.04
		20	85.91	81.52	94.08	87.03
		40	89.51	85.36	97.76	90.76
	5000	10	87.97	83.84	94.41	88.76
		20	94.39	91.94	97.58	94.62
		40	96.13	94.34	99.20	96.50

Table 5.6: The results for channels with $0.065ms$ and $0.130ms$ inter-arrival times. The splitting criterion is Information Gain.

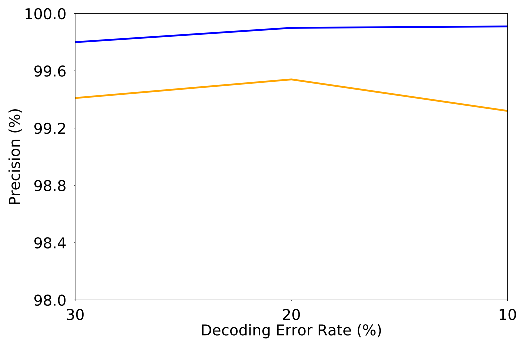
Splitter	**MSL**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Best	5000000	10	81.31	74.63	94.94	83.56
		20	88.52	82.01	98.69	89.58
		40	94.13	90.14	99.10	94.41
	500000	10	99.21	99.18	99.25	99.21
		20	99.22	99.76	98.69	99.22
		40	98.21	99.95	96.46	98.17
	5000	10	99.47	99.19	99.75	99.47
		20	99.78	99.65	99.90	99.78
		40	99.93	99.89	99.98	99.93
Random	5000000	10	56.30	38.42	59.85	45.63
		20	62.18	44.38	65.94	52.07
		40	71.61	57.04	75.99	64.27
	500000	10	79.97	74.40	92.11	82.07
		20	85.76	80.24	95.54	87.05
		40	89.20	84.69	97.83	90.44
	5000	10	88.76	84.82	94.69	89.44
		20	93.62	90.55	97.93	93.99
		40	95.64	93.73	99.15	96.12

Table 5.7: The results for features from Iglesias et al. [1]. The splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.

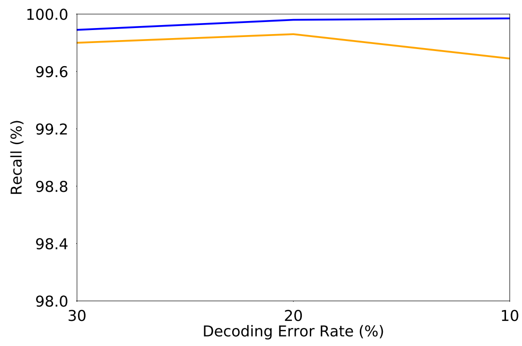
Inter-Arrival Times (ms)	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
0.018 and 0.036	99.60	99.41	99.80	99.60
0.035 and 0.070	99.70	99.54	99.86	99.70
0.065 and 0.130	99.50	99.32	99.69	99.50



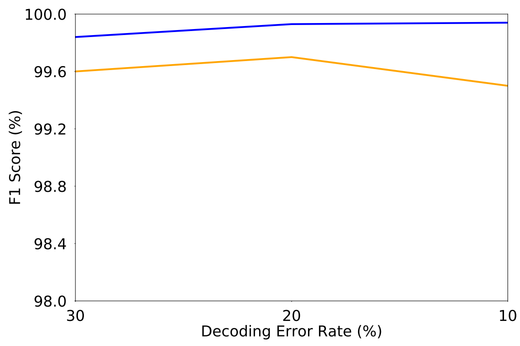
(a) Accuracy



(b) Precision



(c) Recall

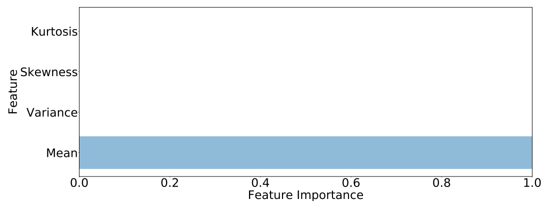


(d) F1 Score

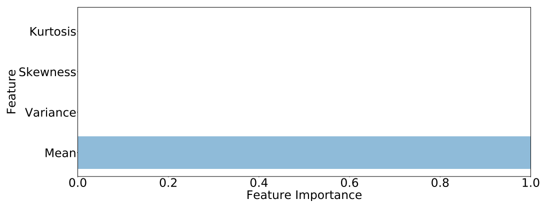
Figure 5.1: Changes in detection scores and decoding error rates for each inter-arrival time configuration. Inter-arrival times increase from left to right on the horizontal line and down to up on the vertical line. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times. The blue line represents results for the first four moments. The orange line represents results for features from Iglesias et al. [1].

Table 5.8: Feature calculation times for one slice of a legitimate channel and Fixed Interval channels.

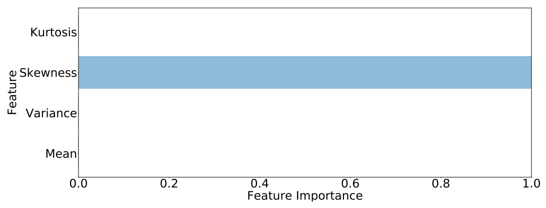
Flow Type	**Feature Type**	**Slice**	**Calculation Time (ms)**
Legitimate	The First Four Moments	10	0.393
		20	0.403
		40	0.467
	Features from Iglesias et al. [1]	10	113.722
		20	37.685
		40	75.604
The covert channel with $0.018ms$ and $0.036ms$ inter-arrival times	The First Four Moments	10	0.705
		20	0.466
		40	0.417
	Features from Iglesias et al. [1]	10	88.413
		20	152.406
		40	644.143
The covert channel with $0.035ms$ and $0.070ms$ inter-arrival times	The First Four Moments	10	0.408
		20	0.400
		40	0.398
	Features from Iglesias et al. [1]	10	70.031
		20	154.955
		40	379.426
The covert channel with $0.065ms$ and $0.130ms$ inter-arrival times	The First Four Moments	10	0.400
		20	0.406
		40	0.423
	Features from Iglesias et al. [1]	10	57.963
		20	87.742
		40	387.103



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

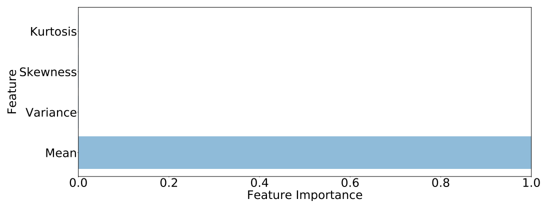


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

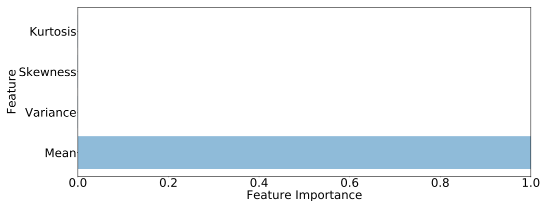


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

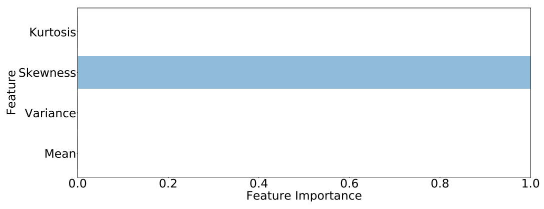
Figure 5.2: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five million samples and each slice consists of ten inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

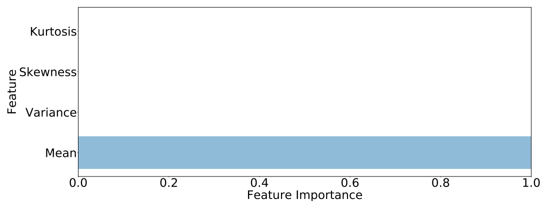


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

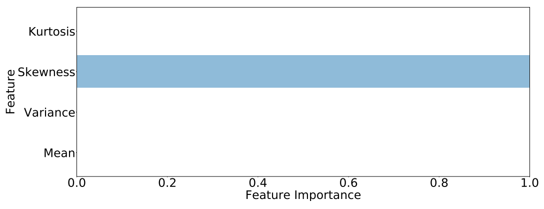


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

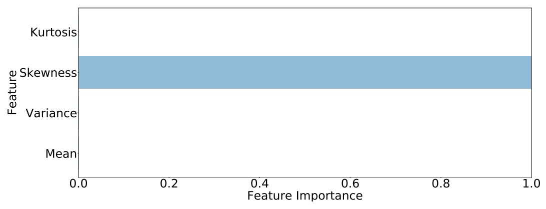
Figure 5.3: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five million samples and each slice consists of twenty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

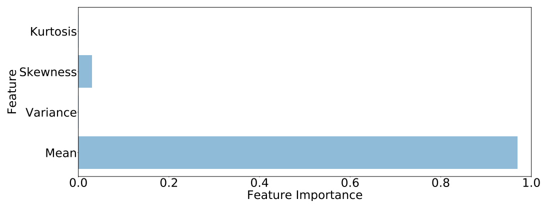


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

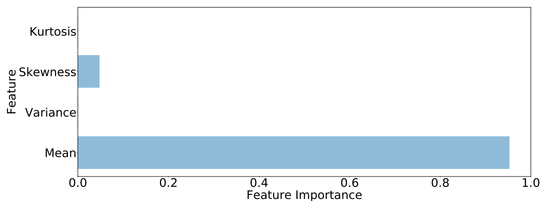


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

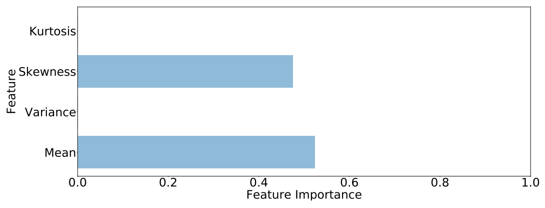
Figure 5.4: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five million samples and each slice consists of forty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

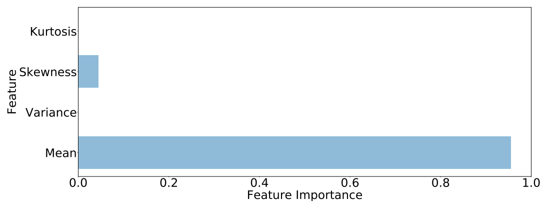


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

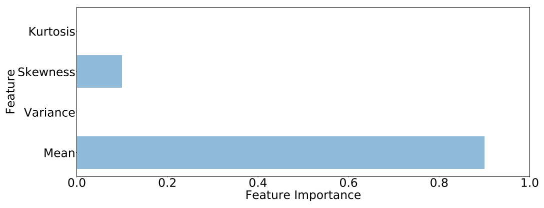


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

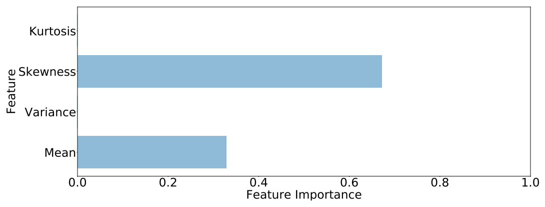
Figure 5.5: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five hundred thousand samples and each slice consists of ten inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

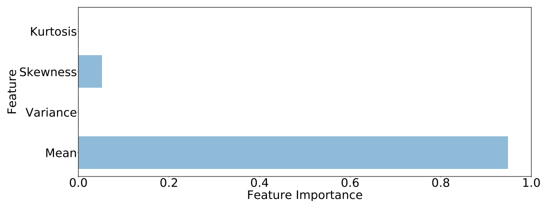


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

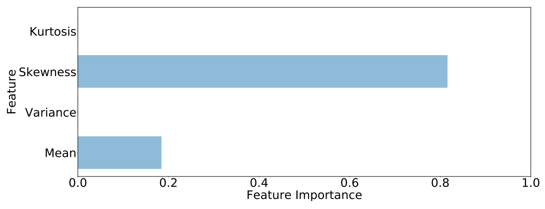


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

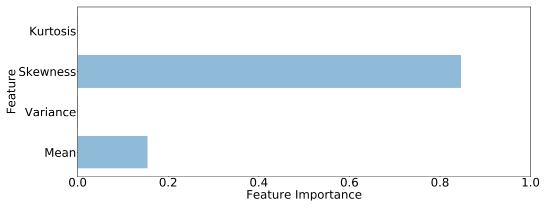
Figure 5.6: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five hundred thousand samples and each slice consists of twenty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

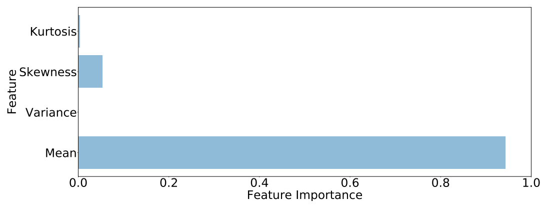


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

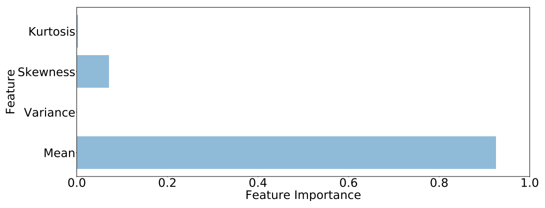


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

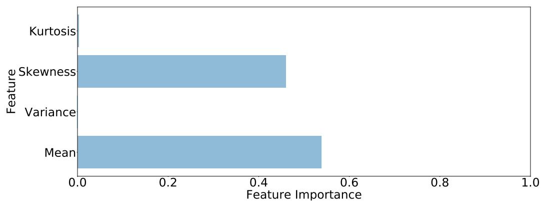
Figure 5.7: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five hundred thousand samples and each slice consists of forty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

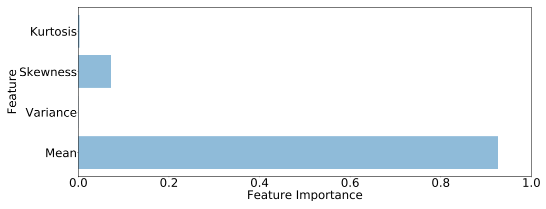


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

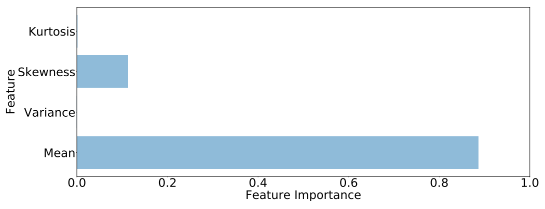


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

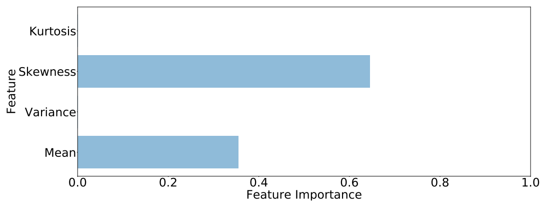
Figure 5.8: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of ten inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

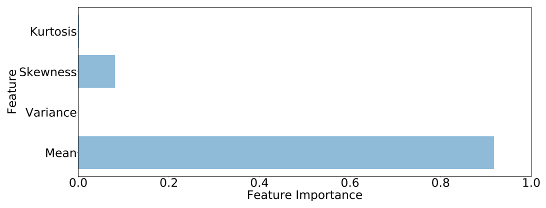


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

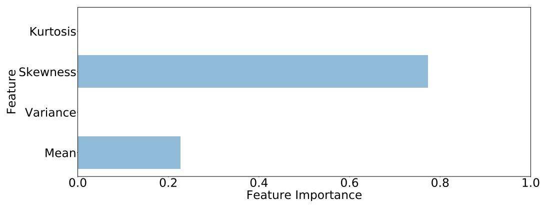


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

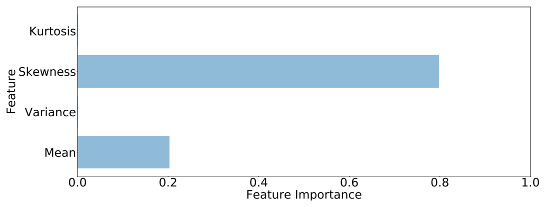
Figure 5.9: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of twenty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.

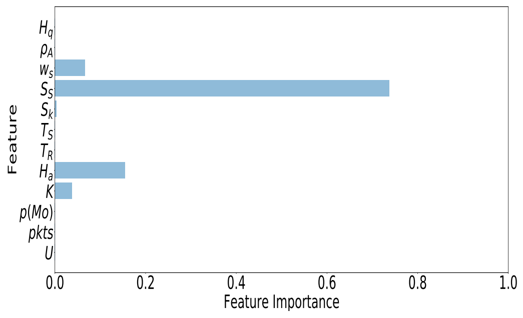


(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.

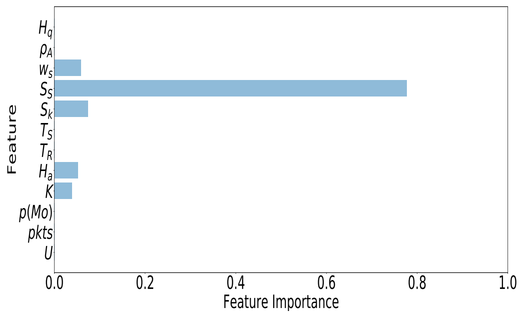


(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

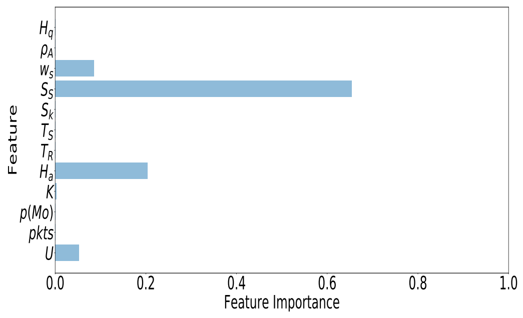
Figure 5.10: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.



(a) Feature importances for Fixed Interval channels with $0.018ms$ and $0.036ms$ inter-arrival times.



(b) Feature importances for Fixed Interval channels with $0.035ms$ and $0.070ms$ inter-arrival times.



(c) Feature importances for Fixed Interval channels with $0.065ms$ and $0.130ms$ inter-arrival times.

Figure 5.11: The feature importances for features from Iglesias et al. [1]. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.

Table 5.9: The results for the detection of types of flows using the first four moments. The covert channel uses $0.018ms$ and $0.036ms$ inter-arrival times. The splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	459	0	0
	2	1689	76	10
	3	5939	0	0
	4	1832	0	0
	5	6510	0	0
	6	61	0	0
	7	220	0	0
	8	6592	0	0
	9	280	0	0
	10	1454	0	0
Covert	1	9553	9	8
	2	9563	0	0
	3	9562	1	2
	4	9562	1	2
	5	9561	2	2
	6	9551	12	8
	7	9562	0	0
	8	9531	32	18
	9	9555	7	8
	10	9538	25	8

Table 5.10: The results for the detection of types of flows using the first four moments. The covert channel uses $0.035ms$ and $0.070ms$ inter-arrival times. The splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	459	0	0
	2	1759	6	2
	3	5939	0	0
	4	1832	0	0
	5	6510	0	0
	6	61	0	0
	7	220	0	0
	8	6591	1	2
	9	280	0	0
	10	1454	0	0
Covert	1	9562	0	0
	2	9546	17	4
	3	9563	0	0
	4	9563	0	0
	5	9561	2	2
	6	9563	0	0
	7	9562	0	0
	8	9563	0	0
	9	9562	0	0
	10	9558	5	2

Table 5.11: The results for the detection of types of flows using the first four moments. The covert channel uses $0.065ms$ and $0.130ms$ inter-arrival times. The splitting criterion is Gini Impurity, the splitter is best, the MSL is five thousand samples and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	459	0	0
	2	1765	0	0
	3	5939	0	0
	4	1832	0	0
	5	6510	0	0
	6	61	0	0
	7	220	0	0
	8	6592	0	0
	9	280	0	0
	10	1454	0	0
Covert	1	9562	0	0
	2	9563	0	0
	3	9563	0	0
	4	9563	0	0
	5	9563	0	0
	6	9562	1	2
	7	9562	0	0
	8	9563	0	0
	9	9562	0	0
	10	9563	0	0

5.2.2 The Results for Jitterbug

The results for Gini Impurity **splitting criterion** is in Table 5.12, and for Information Gain **splitting criterion** is in Table 5.13. Each table shows the accuracy, precision, recall, and F1 score for each **timing window**, **splitter**, and **slice** combination.

We also gave results related to flow detection, feature importances, calculation times and, comparisons with covert channel decoding performances in this section.

Table 5.12: The results for Jitterbug channels. The splitting criterion is Gini Impurity.

Timing Window (ms)	**Splitter**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
1	Best	10	83.71	83.71	83.71	83.71
		20	82.33	82.33	82.33	82.33
		40	84.41	84.41	84.41	84.41
	Random	10	68.54	68.54	68.54	68.54
		20	71.73	71.73	71.73	71.73
		40	75.73	75.73	75.73	75.73
3	Best	10	79.75	79.75	79.75	79.75
		20	80.75	80.75	80.75	80.75
		40	83.33	83.33	83.33	83.33
	Random	10	68.43	68.43	68.43	68.43
		20	71.24	71.24	71.24	71.24
		40	75.11	75.11	75.11	75.11
5	Best	10	79.38	79.38	79.38	79.38
		20	80.49	80.49	80.49	80.49
		40	83.15	83.15	83.15	83.15
	Random	10	69.12	69.12	69.12	69.12
		20	71.15	71.15	71.15	71.15
		40	75.22	75.22	75.22	75.22

(*table continues*)

Table 5.12: continued.

7	Best	10	78.74	78.74	78.74	78.74
		20	78.56	78.56	78.56	78.56
		40	80.43	80.43	80.43	80.43
	Random	10	68.24	68.24	68.24	68.24
		20	71.06	71.06	71.06	71.06
		40	73.86	73.86	73.86	73.86
9	Best	10	80.77	80.77	80.77	80.77
		20	80.14	80.14	80.14	80.14
		40	82.20	82.20	82.20	82.20
	Random	10	69.28	69.28	69.28	69.28
		20	72.19	72.19	72.19	72.19
		40	75.51	75.51	75.51	75.51

Table 5.13: The results for Jitterbug channels. The splitting criterion is Information Gain.

Timing Window (ms)	**Splitter**	**Slice**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
1	Best	10	83.75	83.75	83.75	83.75
		20	82.37	82.37	82.37	82.37
		40	84.51	84.51	84.51	84.51
	Random	10	68.55	68.55	68.55	68.55
		20	71.82	71.82	71.82	71.82
		40	75.54	75.54	75.54	75.54

(*table continues*)

Table 5.13: continued.

3	Best	10	79.77	79.77	79.77	79.77
		20	80.81	80.81	80.81	80.81
		40	83.42	83.42	83.42	83.42
	Random	10	68.26	68.26	68.26	68.26
		20	71.46	71.46	71.46	71.46
		40	75.09	75.09	75.09	75.09
5	Best	10	79.41	79.41	79.41	79.41
		20	80.52	80.52	80.52	80.52
		40	83.24	83.24	83.24	83.24
	Random	10	69.02	69.02	69.02	69.02
		20	71.10	71.10	71.10	71.10
		40	75.02	75.02	75.02	75.02
7	Best	10	78.70	78.70	78.70	78.70
		20	78.60	78.60	78.60	78.60
		40	80.50	80.50	80.50	80.50
	Random	10	68.36	68.36	68.36	68.36
		20	71.10	71.10	71.10	71.10
		40	74.09	74.09	74.09	74.09
9	Best	10	80.84	80.84	80.84	80.84
		20	80.18	80.18	80.18	80.18
		40	82.28	82.28	82.28	82.28
	Random	10	69.43	69.43	69.43	69.43
		20	72.19	72.19	72.19	72.19
		40	75.77	75.77	75.77	75.77

Table 5.14: The results for features from Iglesias et al. [1]. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Timing Window (ms)	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
1	74.94	74.94	74.94	74.94
3	73.44	73.44	73.44	73.44
5	72.30	72.30	72.30	72.30
7	72.67	72.67	72.67	72.67
9	72.88	72.88	72.88	72.88

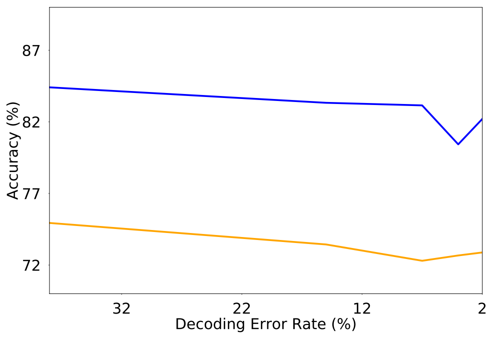
Table 5.25: Feature calculation times for one slice of a legitimate channel and Jitterbug channels.

Flow Type	**Feature Type**	**Slice**	**Calculation Time (ms)**
Legitimate	The First Four Moments	10	0.394
		20	0.407
		40	0.397
	Features from Iglesias et al. [1]	10	19.103
		20	74.779
		40	193.578
The covert channel with $1ms$ timing window	The First Four Moments	10	0.394
		20	0.409
		40	0.663
	Features from Iglesias et al. [1]	10	57.004
		20	139.118
		40	55.508

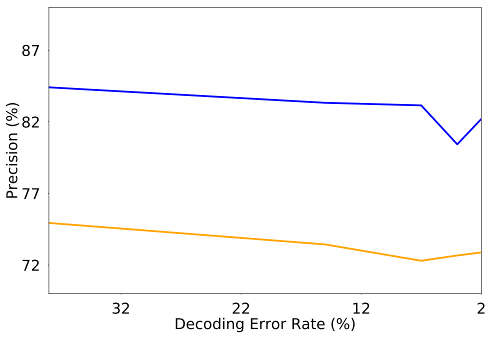
(*table continues*)

Table 5.25: continued.

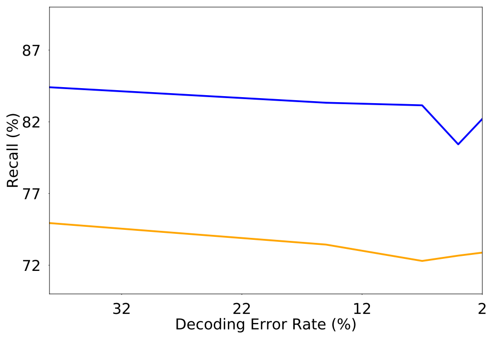
The covert channel with $3ms$ timing window	The First Four Moments	10	0.398
		20	0.417
		40	0.397
	Features from Iglesias et al. [1]	10	59.415
		20	137.509
		40	55.481
The covert channel with $5ms$ timing window	The First Four Moments	10	0.446
		20	0.412
		40	0.573
	Features from Iglesias et al. [1]	10	56.882
		20	134.425
		40	0.584
The covert channel with $7ms$ timing window	The First Four Moments	10	0.481
		20	0.412
		40	53.871
	Features from Iglesias et al. [1]	10	54.170
		20	138.958
		40	52.840
The covert channel with $9ms$ timing window	The First Four Moments	10	0.620
		20	0.443
		40	0.592
	Features from Iglesias et al. [1]	10	57.166
		20	142.354
		40	54.619



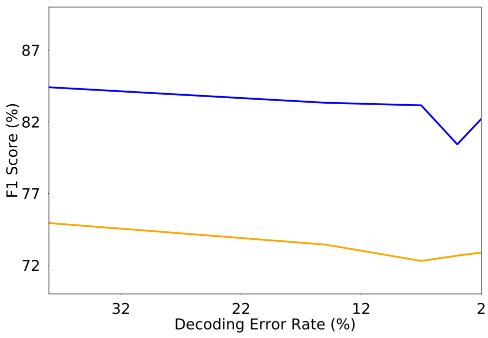
(a) Accuracy



(b) Precision

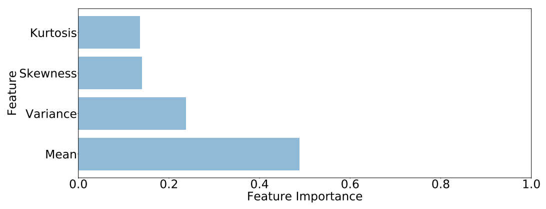


(c) Recall

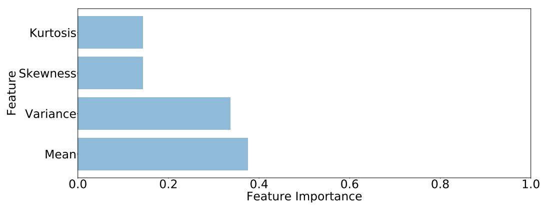


(d) F1 Score

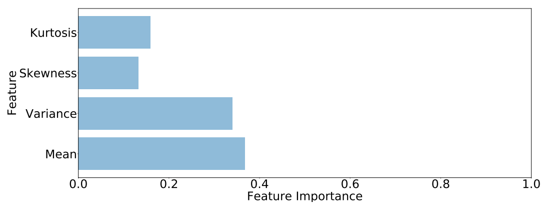
Figure 5.12: Changes in detection scores and decoding error rates for each timing window configuration. Timing windows increase from left to right on the horizontal line and down to up on the vertical line. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times. The blue line represents results for the first four moments. The orange line represents results for features from Iglesias et al. [1].



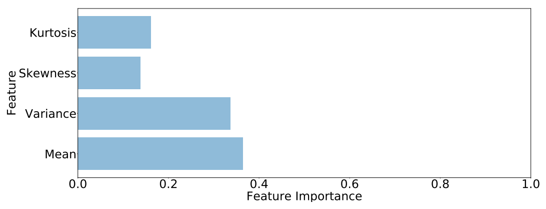
(a) Feature importances for Jitterbug channels with $1ms$ time window.



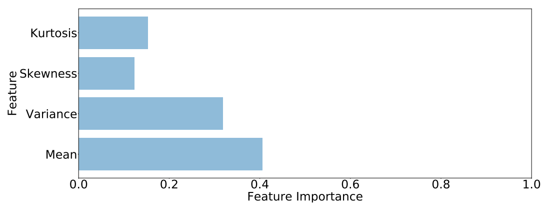
(b) Feature importances for Jitterbug channels with $3ms$ time window.



(c) Feature importances for Jitterbug channels with $5ms$ time window.

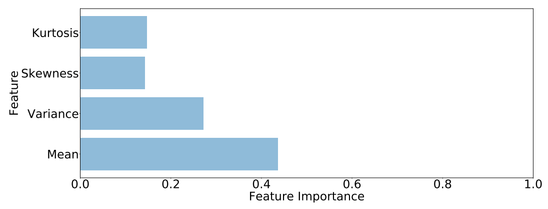


(d) Feature importances for Jitterbug channels with $7ms$ time window.

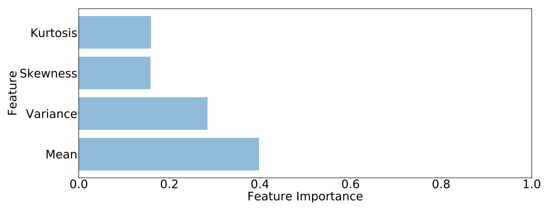


(e) Feature importances for Jitterbug channels with $9ms$ time window.

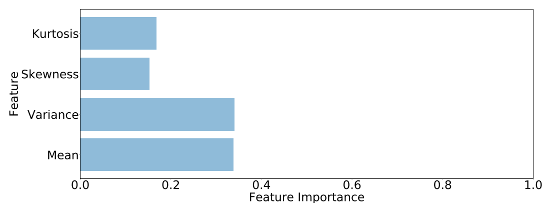
Figure 5.13: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, and each slice consists of ten inter-arrival times.



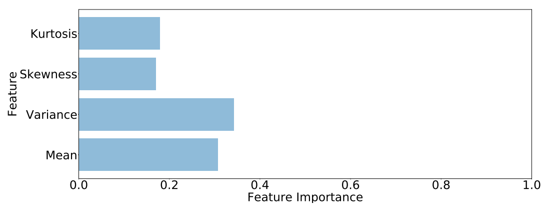
(a) Feature importances for Jitterbug channels with $1ms$ time window.



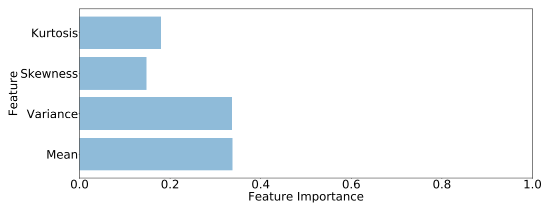
(b) Feature importances for Jitterbug channels with $3ms$ time window.



(c) Feature importances for Jitterbug channels with $5ms$ time window.

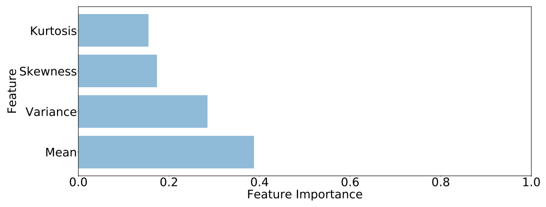


(d) Feature importances for Jitterbug channels with $7ms$ time window.

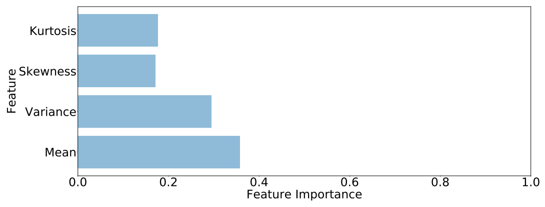


(e) Feature importances for Jitterbug channels with $9ms$ time window.

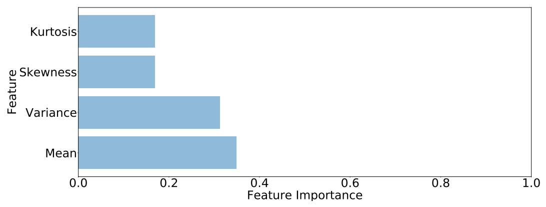
Figure 5.14: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, and each slice consists of twenty inter-arrival times.



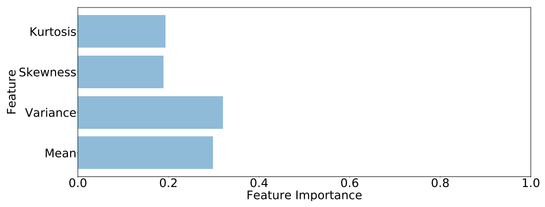
(a) Feature importances for Jitterbug channels with $1ms$ time window.



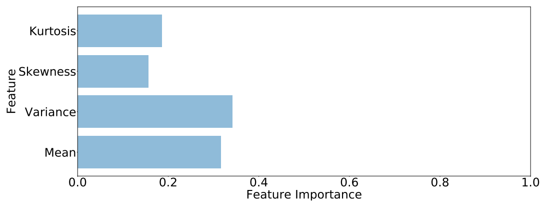
(b) Feature importances for Jitterbug channels with $3ms$ time window.



(c) Feature importances for Jitterbug channels with $5ms$ time window.

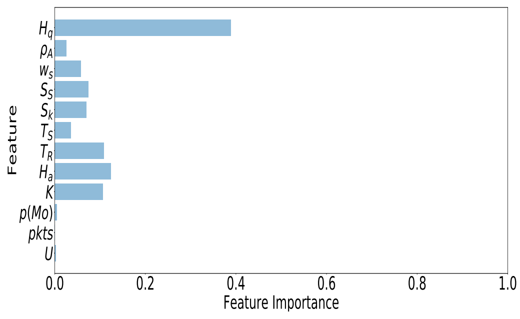


(d) Feature importances for Jitterbug channels with $7ms$ time window.

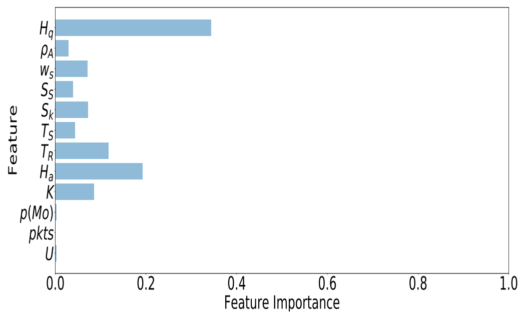


(e) Feature importances for Jitterbug channels with $9ms$ time window.

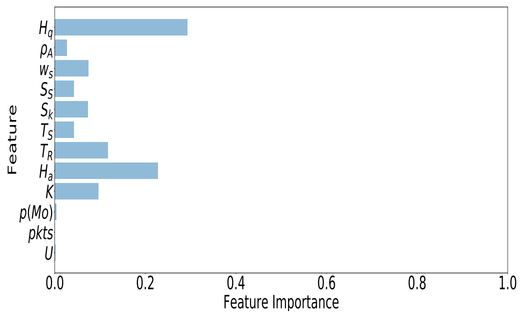
Figure 5.15: The feature importances for the first four moments. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.



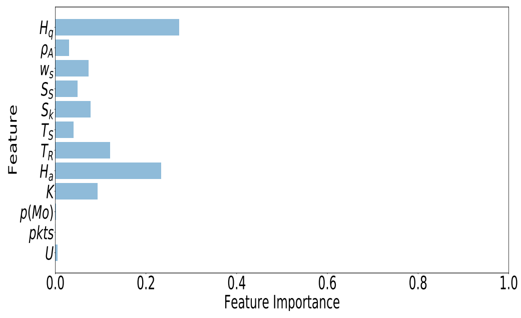
(a) Feature importances for Jitterbug channels with $1ms$ time window.



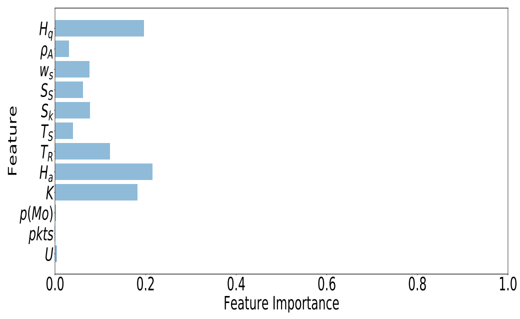
(b) Feature importances for Jitterbug channels with $3ms$ time window.



(c) Feature importances for Jitterbug channels with $5ms$ time window.



(d) Feature importances for Jitterbug channels with $7ms$ time window.



(e) Feature importances for Jitterbug channels with $9ms$ time window.

Figure 5.16: The feature importances for features from Iglesias et al. [1]. Detection scores were obtained using the following configuration: the splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Table 5.15: The results for the detection of types of flows using the first four moments. The covert channel has $1ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	66	8	14
	2	1476	399	514
	3	143	0	0
	4	48	22	31
	5	0	41	0
	6	258	228	180
	7	149	140	80
	8	20	17	11
	9	22	34	12
	10	59	42	28
Covert	1	2509	10	14
	2	7	0	0
	3	377	14	16
	4	143	5	2
	5	56	0	0
	6	2565	447	248
	7	86	142	98
	8	28	9	2
	9	85	58	48
	10	436	463	218

Table 5.16: The results for the detection of types of flows using the first four moments. The covert channel has $3ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	61	13	17
	2	1462	413	529
	3	143	0	0
	4	32	38	25
	5	19	22	4
	6	261	225	186
	7	131	158	90
	8	16	21	16
	9	19	37	16
	10	55	46	33
Covert	1	2502	17	20
	2	7	0	0
	3	372	19	24
	4	139	9	11
	5	56	0	0
	6	2514	498	306
	7	96	132	77
	8	31	6	6
	9	116	27	24
	10	429	470	257

Table 5.17: The results for the detection of types of flows using the first four moments. The covert channel has $5ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	59	15	22
	2	1484	391	534
	3	143	0	0
	4	49	21	22
	5	7	34	2
	6	252	234	197
	7	140	149	79
	8	13	24	8
	9	39	17	14
	10	63	38	32
Covert	1	2509	10	12
	2	7	0	0
	3	371	20	19
	4	145	3	5
	5	54	2	4
	6	2437	575	328
	7	104	124	87
	8	26	11	6
	9	104	39	23
	10	461	438	273

Table 5.18: The results for the detection of types of flows using the first four moments. The covert channel has $7ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	58	16	18
	2	1435	440	492
	3	143	0	0
	4	49	21	28
	5	9	32	4
	6	270	216	149
	7	142	147	68
	8	12	25	8
	9	32	24	13
	10	58	43	23
Covert	1	2504	15	22
	2	7	0	0
	3	379	12	13
	4	144	4	6
	5	56	0	0
	6	2054	958	471
	7	96	132	56
	8	11	26	6
	9	126	17	10
	10	434	465	218

Table 5.19: The results for the detection of types of flows using the first four moments. The covert channel has $9ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	68	6	10
	2	1463	412	483
	3	138	5	2
	4	43	27	36
	5	9	32	5
	6	257	229	181
	7	132	157	94
	8	22	15	10
	9	23	33	18
	10	50	51	38
Covert	1	2513	6	8
	2	7	0	0
	3	366	25	24
	4	148	0	0
	5	55	1	2
	6	2134	878	352
	7	103	125	66
	8	20	17	4
	9	90	53	29
	10	429	470	265

Table 5.20: The results for the detection of types of flows using features from Iglesias et al. [1]. The covert channel has $1ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	53	21	23
	2	666	1209	662
	3	112	31	20
	4	64	6	6
	5	38	3	4
	6	381	105	140
	7	136	153	102
	8	29	8	12
	9	56	0	0
	10	87	14	14
Covert	1	2112	407	419
	2	3	4	1
	3	286	105	69
	4	126	22	27
	5	25	31	13
	6	1829	1183	766
	7	110	118	94
	8	27	10	10
	9	115	28	25
	10	683	216	264

Table 5.21: The results for the detection of types of flows using features from Iglesias et al. [1]. The covert channel has $3ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	43	31	34
	2	634	1241	691
	3	118	25	32
	4	70	0	0
	5	40	1	2
	6	372	114	157
	7	158	131	91
	8	31	6	8
	9	55	1	2
	10	87	14	21
Covert	1	2123	396	445
	2	5	2	4
	3	204	187	70
	4	124	24	35
	5	32	24	13
	6	1785	1227	848
	7	146	82	67
	8	26	11	13
	9	103	40	36
	10	695	204	264

Table 5.22: The results for the detection of types of flows using features from Iglesias et al. [1]. The covert channel has $5ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	54	20	23
	2	579	1296	644
	3	137	6	10
	4	69	1	2
	5	40	1	2
	6	378	108	150
	7	187	102	94
	8	32	5	6
	9	53	3	5
	10	92	9	13
Covert	1	2105	414	444
	2	2	5	1
	3	199	192	73
	4	122	26	32
	5	30	26	23
	6	1862	1150	939
	7	135	93	87
	8	35	2	4
	9	112	31	32
	10	683	216	247

Table 5.23: The results for the detection of types of flows using features from Iglesias et al. [1]. The covert channel has $7ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	52	22	22
	2	621	1254	661
	3	133	10	16
	4	70	0	0
	5	38	3	3
	6	373	113	166
	7	184	105	92
	8	37	0	0
	9	53	3	3
	10	91	10	12
Covert	1	2205	314	366
	2	3	4	4
	3	220	171	94
	4	111	37	39
	5	24	32	19
	6	1928	1084	989
	7	132	96	84
	8	33	4	0
	9	98	45	50
	10	712	187	266

Table 5.24: The results for the detection of types of flows using features from Iglesias et al. [1]. The covert channel has $9ms$ timing window. The splitting criterion is Gini Impurity, the splitter is best, and each slice consists of forty inter-arrival times.

Flow Type	**Index of Flow**	**Number of Correct Estimates**	**Number of Incorrect Estimates**	**Number of Decision Changes**
Legitimate	1	54	20	26
	2	482	1393	560
	3	138	5	10
	4	20	50	27
	5	40	1	2
	6	356	130	153
	7	142	147	94
	8	29	8	11
	9	55	1	1
	10	83	18	12
Covert	1	2107	412	414
	2	5	2	3
	3	268	123	127
	4	120	28	39
	5	50	6	12
	6	1659	1353	819
	7	132	96	98
	8	5	32	10
	9	45	98	44
	10	439	460	303

5.2.3 Discussion

The results show that the splitter parameter has the most effect on the scores regardless of the type and parameters of covert channels and features. This means that the splitter parameter is the one that affects the structure of the tree the most. Moreover, the detection performance of a tree with a random splitter is much lower than the one with the best splitter. The slice also has significant effects on detection; slices with more inter-arrival times have significantly higher detection scores. Surprisingly, using different splitting criteria did not ultimately lead to significant changes. The detector achieved the best result for all covert channels when the splitter was the best, the MSL was small, and a slice consisted of forty inter-arrival times.

For Fixed Interval channels, scores are above 90%. This means we can use our features to detect Fixed Interval channels that transmit binary data with two different inter-arrival times. Moreover, detection scores are similar for all Fixed Interval channels as we can see in Figure 5.1 and the tables between Table 5.1 and 5.6. So, even if the attacker settles for higher decoding errors to avoid being caught, it would have no effect. Furthermore, we can use the same parameters to detect Fixed Interval channels with different inter-arrival times. We were able to achieve high scores with a small number of network packages when the splitter was best and the MSL was low. Table 5.1 shows an example of this. We achieved 98.83% accuracy only using ten inter-arrival times (thus, eleven network packets). This means that our system can detect a Fixed Interval channel only after eleven packets.

On the other hand, detection results for Jitterbug channels is significantly lower than Fixed Interval channels. This is because Jitterbug is more similar to the legitimate channel than Fixed Interval. Still, scores are above 80%. Table 5.12, 5.13 and Figure 5.12 show that the timing window has a small impact on detection performance. The reason for this is that some timing window values can generate inter-arrival times that are a bit like legitimate traffic. However, this effect is too small for an attacker exploit. We achieved the highest detection performance when the splitter was the best and a slice consisted of forty inter-arrival times. The results for Jitterbug indicate that our system requires a large number of packages to detect this covert channel. Though Jitterbug itself also requires a large number of packages. A solution may be to use multiple flows with a small number of packages, but since Jitterbug does not create flows, it is not easy to do so. Moreover, users tend to use one SSH session at a time. As a result, Jitterbug would probably have to wait sometime between sessions. This will cause the data transmission to slow down considerably.

Table 5.7 and 5.14 show the detection scores for features from Iglesias et al. [1]. From Table 5.7 and Figure 5.1, we can see for Fixed Interval channels, the results that these features achieved are similar to the first four moments. On the other hand, Table 5.14 and Figure 5.12 show that for Jitterbug channels, the first four moments perform significantly better.

Feature importances for the first four features and Fixed Interval show that the mean and the skewness are used for detection. Since the shape of the Fixed Interval traffic is different from the legitimate traffic, it affected the skewness, a feature that measures shape. Normally, the shape also affects the kurtosis, but the decision tree uses only one of these features. For some covert channels, the mean of the inter-arrival times are also different from the legitimate channels'. Feature importances for features from Iglesias et al. [1], show that the S_S is the dominant feature. This means that the number of sharp peaks in the inter-arrival time histogram is different for Fixed Interval and legitimate channels. The S_S is most likely affected by the shape difference between channels. Theoretically, entropy-related features (K and H_a) is also different for Fixed Interval and legitimate channels because Fixed Interval channels have repetitive inter-arrival times and legitimate channels are more random. Entropy-related features probably were negatively affected by unexpected network delays.

Among the first four moments, the decision tree, chose mean and variance as dominant features to detect Jitterbug channels. This happened because the shape of Jitterbug channels is similar to the legitimate one. Also, the traffic Jitterbug creates has a higher mean. Moreover, inter-arrival times that it creates are multiples (or multiple plus the half) of the timing window. On the other hand, H_q is dominant among the features of Iglesias et al. [1]. H_q is a way to measure self-similarity. Also, importances for K, H_a and T_R are a little higher than the others. These features also measure self-similarity and repetitiveness. This means that features from Iglesias et al. [1] captured repetitive patterns in the Jitterbug.

Table 5.9, 5.10 and, 5.11 show that our method can determine the type of a flow with high accuracy. Moreover, incoming network packages can not easily change its decision. This means that our method can determine whether a channel is legitimate or Fixed Interval using a small number of packages. Unfortunately, we can not say the same for Jitterbug. Table 5.15, 5.16, 5.17, 5.18 and, 5.19 indicate that there is no specific pattern for classification of a flow. Our method accurately identifies some of them, accidentally identifies some as legitimate, and can not decide what the others are. The same goes for legitimate channels. In order to see whether this situation is specific to our method, we also tried to classify flows with features from Iglesias et al. [1]. Table 5.20, 5.21, 5.22, 5.23 and, 5.24 show that the situation is similar for these features.

The results for all covert and legitimate flows state that the differences in inter-arrival times or timing windows have small effects on classification. This result is consistent with the results of our experiments with all flows.

The network latencies and inter-arrival times we used in our work are close to the LAN (Local Area Network). The LAN allows the covert channels to operate with low latencies. Lower latencies increase Fixed Interval channels' capacity and decrease the probability to detect them. Moreover, it is also possible to decrease the probability to detect Jitterbug channels'. We can not infer this from our results though theoretically, lower latencies mean lower timing windows. Lower timing windows would create inter-arrival times closer to the inter-keypress times. As a result, detection rates would become lower. The LAN is the worst-case scenario for our detector. On the other hand, if Fixed Interval or Jitterbug tries to operate on wider networks, they would have problems due to higher rates of packet loss and latencies. For instance, if the sender tries to transmit the message to another LAN through the MAN (Metropolitan Area Network), it would have to create higher inter-arrival times to compensate with latencies. The worst case scenario for a covert network timing channel is to operate on WAN (Wide Area Network). The latencies on WAN can be up to $100ms$. This means that Fixed Interval and Jitterbug would have to create inter-arrival times up to the second resolution. As a result, they would cause huge delays that even an ordinary user can notice.

Our model has to collect data from routers and switches. In order to run fast, it must be on or close to these devices. There are several ways to implement our model on routers and switches. The first one is to use specific hardware and software. This allows us to implement our model as we want, but it is costly. Another way is to use a generic coding language like P4 [73]. This method is cheaper but slows down the packet forwarding process because network packets are processed before forwarding. In order to fix this, we could try to replace the decision tree with rules created using this tree, but there would still be a certain amount of delay. We could also mirror the ports using hardware or software and send mirrored packets to another computer. This way, the packet forwarding process would not slow down. However, under heavy load, some mirrored packets may be lost. Nevertheless, this technique is used for monitoring the network, diagnosing error or detecting network intrusions. For instance, the Cisco intrusion detection module [74] processes the packets that routers mirror to it.

Table 5.8 and 5.25 show calculation times for the first four moments and features from Iglesias et al. [1]. The time to calculate a slice of the latter is much higher that the first one. If we use routers or switches to calculate features from Iglesias et al., users would experience high delays. This will not be the case when packets are collected via mirroring. Even so, there will still be significant delays in delivering detection results.

For Jitterbug detection, we must know which packets belong to keypresses. In our work, we assumed that our detector did. However, in real world the detector must find a way to detect keypresses. Zhang and Paxson [75] observed that interactive network traffic is different than other types of network traffic. They found out that interactive connections have a higher number of small packets. Also, the inter-arrival time distribution of interactive connections is different than other types of network traffic. Moreover, the keystroke packets tend to be smaller than 20 bytes. They developed an algorithm based on these facts. There are also works on the properties of keystroke dynamics [76]. We must do more research before before deciding which algorithm to apply and how to apply it.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we tried to detect Fixed Interval with two inter-arrival times and Jitterbug with a single timing window using the first four moments and a decision tree. In order to do that, we firstly created a dataset that contains Fixed Interval and Jitterbug channels. Then, using this dataset, we performed some experiments. During these experiments, we tried different configurations for covert channels, features, and decision trees to find out the best configuration and detection performance. The accuracy, precision, recall and F1 scores for Fixed Interval channels were above 90%. This did not change even for a small number of network packages. On the other hand, scores for Jitterbug were slightly worse (above 80%) , but still plausible. Moreover, when we compared detection performances with the decoding performances, we found out that inter-arrival times and timing windows have a little or no impact on detection performances. This means we can use decision trees with similar parameters to detect channels with different configurations but the same type.

We also compared our features' detection performance and calculation times with the ones defined by Iglesias et al. [1] (except c). For Fixed Interval, detection scores were similar. On the other hand, for Jitterbug, the first four moments outperformed the features from the paper [1]. Also, the timing results showed that the computational time of our features is significantly lower than the features in the [1] paper.

In addition to these, we tried to classify some flows in accordance with the real-life scenario. The results showed that our method could correctly and consistently discriminate legitimate and Fixed Interval flows. Unfortunately, for Jitterbug flows the result was not so good. Our features reacted differently for each flow and did not achieve consistent behavior. The same was true for the features from the paper [1].

6.2 Future Work

Although we come this far, we still have a long way to go. In the future, we plan to work on detection flow we will work on better detection of Jitterbug flow. we also plan to work with different versions of Fixed Interval and Jitterbug. One possible variety is using multiple inter-arrival times and timing windows. For instance, we can create a Jitterbug channel with $1ms$, $3ms$ and $5ms$ timing windows. This channel then can "rotate" these timing windows while sending data. Another variety may be sending multiple symbols instead of binary symbols. This way, a covert channel may send data using less inter-arrival times, thus would reduce the chance of detection.

We also intend to expand our dataset with other types of files and folders (text, pdf, etc.). After that, we plan to experiment with other types of channels.

We would like to see our system work in the real world in the near future.

REFERENCES

[1] F. I. Vázquez, R. Annessi, and T. Zseby, “Analytic study of features for the detection of covert timing channels in network traffic,” *Journal of Cyber Security and Mobility*, vol. 6, no. 3, pp. 225–270, 2017.

[2] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” tech. rep., Citeseer, 2009.

[3] R. Greenstadt, “Covert channels towards a qual project,” 2008. Available at https://www.cs.drexel.edu/~greenie/eecs_files/ccslides.pdf, access date 2019-12-18.

[4] I. S. Moskowitz and M. H. Kang, “Covert channels-here to stay?,” in *Proceedings of COMPASS’94-1994 IEEE 9th Annual Conference on Computer Assurance*, pp. 235–243, IEEE, 1994.

[5] S. Cabuk, *Network covert channels: Design, analysis, detection, and elimination.* PhD thesis, Purdue University, 2006.

[6] J. Giles and B. Hajek, “An information-theoretic and game-theoretic study of timing channels,” *IEEE Transactions on Information Theory*, vol. 48, no. 9, pp. 2455–2477, 2002.

[7] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang, “What will 5g be?,” *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[8] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, “5g: A tutorial overview of standards, trials, challenges, deployment, and practice,” *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[9] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, “A survey on low

latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, 2018.

[10] B. Bellalta, "Ieee 802.11 ax: High-efficiency wlans," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 38–46, 2016.

[11] S. McCann, "Status of project ieee 802.11ax," 2019. Available at http://www.ieee802.org/11/Reports/tgax_update.htm, access date 2019-11-16.

[12] B. Vijay and B. Malarkodi, "High-efficiency wlans for dense deployment scenarios," *Sādhanā*, vol. 44, no. 2, p. 33, 2019.

[13] R. H. Weber, "Internet of things–need for a new legal environment?," *Computer law & security review*, vol. 25, no. 6, pp. 522–527, 2009.

[14] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, p. 1101, 2012.

[15] F. Iglesias, R. Annessi, and T. Zseby, "Dat detectors: uncovering tcp/ip covert channels by descriptive analytics," *Security and Communication Networks*, vol. 9, no. 15, pp. 3011–3029, 2016.

[16] İlyas Çiçekli, "Decision tree learning," 2016. Available at https://web.cs.hacettepe.edu.tr/~ilyas/Courses/BIL712/lec02-DecisionTree.pdf, access date 2019-06-27.

[17] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

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

[19] V. Berk, A. Giani, G. Cybenko, and N. Hanover, "Detection of covert channel encoding in network packet delays," *Rapport technique TR536, de lUniversité de Dartmouth*, vol. 19, 2005.

[20] M. B. Gaurav Shah, Andres Molina, “Keyboards and covert channels,” *USENIX Security Symposium*, vol. 15, 2006.

[21] H. W. Steven Gianvecchio, “Detecting covert timing channels: An entropy-based approach,” *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 307–316, 2007.

[22] B. Lampson, “A note on the confinement problem,” 1973.

[23] T. C. C. Project, “Common methodology for information technology security evaluation, common methodology for information technology security evaluation, version 3.1 revision 5,” tech. rep., 2017.

[24] U. D. of Defense, “Trusted computer system evaluation, the orange book,” tech. rep., 1985.

[25] J. Rutkowska, “The implementation of passive covert channels in the linux kernel,” in *Chaos communication congress, chaos computer club eV*, vol. 16, 2004.

[26] C. Scott, “Network covert channels: Review of current state and analysis of viability of the use of x.509 certificates for covert communications,” *Technical Report: Department of Mathematics-University of London.*, 2008.

[27] S. T. K. Hamed Okhravi, Stanley Bak, “Design, implementation and evaluation of covert channel attacks,” *In Technologies for Homeland Security (HST), 2010 IEEE International Conference*, pp. 481–487, 2010.

[28] L. Deshotels, “Inaudible sound as a covert channel in mobile devices,” in *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*, 2014.

[29] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, “Covert messaging through tcp timestamps,” in *International Workshop on Privacy Enhancing Technologies*, pp. 194–208, Springer, 2002.

[30] Z. Wang and R. B. Lee, “Covert and side channels due to processor architecture,” in *2006 22nd Annual Computer Security Applications Conference (ACSAC’06)*, pp. 473–482, IEEE, 2006.

[31] S. Zander *et al.*, “Performance of selected noisy covert channels and their countermeasures in ip networks,” *Centre for Advanced Internet Architectures Faculty of Information and Communication Technologies*, 2010.

[32] S. J. Murdoch, “Hot or not: Revealing hidden services by their clock skew,” in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 27–36, ACM, 2006.

[33] R. A. Kemmerer and T. Taylor, “A modular covert channel analysis methodology for trusted dg/ux/sup tm,” in *Proceedings 12th Annual Computer Security Applications Conference*, pp. 224–235, IEEE, 1996.

[34] M. H. Kang, I. S. Moskowitz, and S. Chincheck, “The pump: A decade of covert fun,” in *21st Annual Computer Security Applications Conference (ACSAC’05)*, pp. 7–pp, IEEE, 2005.

[35] C. S. Serdar Cabuk, Carla E. Brodley, “Ip covert timing channels: Design and detection,” *Proceedings of the 11th ACM conference on Computer and communications security*, pp. 178–187, 2004.

[36] Y. Dodge, “Kolmogorov–smirnov test,” *The concise encyclopedia of statistics*, pp. 283–287, 2008.

[37] F. R. H. S. Pradhumna L. Shrestha, Michael Hempel, “A support vector machine-based framework for detection of covert timing channels,” *IEEE Transactions on Dependable and Secure Computing,(1)*, pp. 1–1, 2016.

[38] T. Evgeniou and M. Pontil, “Support vector machines: Theory and applications,” in *Advanced Course on Artificial Intelligence*, pp. 249–257, Springer, 1999.

[39] B. Schölkopf, A. J. Smola, F. Bach, *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[40] S. H. Sellke, C.-C. Wang, S. Bagchi, and N. Shroff, “Tcp/ip timing channels: Theory to implementation,” in *IEEE INFOCOM 2009*, pp. 2204–2212, IEEE, 2009.

[41] O. Darwish, A. Al-Fuqaha, M. Anan, and N. Nasser, "The role of hierarchical entropy analysis in the detection and time-scale determination of covert timing channels," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 153–159, IEEE, 2015.

[42] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[43] O. Darwish, A. Al-Fuqaha, G. B. Brahim, and M. A. Javed, "Using mapreduce and hierarchical entropy analysis to speed-up the detection of covert timing channels," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1102–1107, IEEE, 2017.

[44] F. Iglesias, V. Bernhardt, R. Annessi, and T. Zseby, "Decision tree rule induction for detecting covert timing channels in tcp/ip traffic," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pp. 105–122, Springer, 2017.

[45] S. Zander, G. Armitage, and P. Branch, "An empirical evaluation of ip time to live covert channels," in *2007 15th IEEE International Conference on Networks*, pp. 42–47, IEEE, 2007.

[46] J. Wu, Y. Wang, L. Ding, and X. Liao, "Improving performance of network covert timing channel through huffman coding," *Mathematical and Computer Modelling*, vol. 55, no. 1-2, pp. 69–79, 2012.

[47] W. C. Gasior, "Network covert channels on the android platform," 2011.

[48] X. Luo, E. W. Chan, and R. K. Chang, "Tcp covert timing channels: Design and detection," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pp. 420–429, IEEE, 2008.

[49] F. Iglesias and T. Zseby, "Are network covert timing channels statistical anomalies?," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, p. 81, ACM, 2017.

[50] C. Sony and K. Cho, "Traffic data repository at the wide project," in *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, pp. 263–270, 2000.

[51] E. J. Castillo, X. Mountrouidou, and X. Li, "Time lord: Covert timing channel implementation and realistic experimentation," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 755–756, ACM, 2017.

[52] B. W. Silverman, "Using kernel density estimates to investigate multimodality," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 43, no. 1, pp. 97–99, 1981.

[53] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary physics*, vol. 46, no. 5, pp. 323–351, 2005.

[54] E. Parzen, "On spectral analysis with missing observations and amplitude modulation," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 383–392, 1963.

[55] J. V. Bradley, "Distribution-free statistical tests," tech. rep., 1968.

[56] H. B. Mann, "On a test for randomness based on signs of differences," *The Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 193–199, 1945.

[57] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *International journal of computer mathematics*, vol. 2, no. 1-4, pp. 157–168, 1968.

[58] M. Li, P. Vitányi, *et al.*, *An introduction to Kolmogorov complexity and its applications*, vol. 3. Springer, 2008.

[59] A. Carbone, G. Castelli, and H. E. Stanley, "Time-dependent hurst exponent in financial time series," *Physica A: Statistical Mechanics and its Applications*, vol. 344, no. 1-2, pp. 267–271, 2004.

[60] S. M. Pincus, "Approximate entropy as a measure of system complexity.," *Proceedings of the National Academy of Sciences*, vol. 88, no. 6, pp. 2297–2301, 1991.

[61] "Poisson point process," 2019-10-22. Available at https://en.wikipedia.org/wiki/Poisson_point_process, access date 2019-11-16.

[62] W. Koehrsen, “numpy.random.exponential,” 2019-01-21. Available at https://towardsdatascience.com/the-poisson-distribution-and-poisson-process-explained-4e2cb17d459, access date 2019-06-24.

[63] “Exponential distribution,” 2019-10-10. Available at https://en.wikipedia.org/wiki/Exponential_distribution, access date 2019-11-16.

[64] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[65] K. S. Killourhy and R. A. Maxion, “Comparing anomaly-detection algorithms for keystroke dynamics,” in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pp. 125–134, IEEE, 2009.

[66] A. Tveito, A. M. Bruaset, and O. Lysne, *Simula Research Laboratory: By Thinking Constantly about it.* Springer Science & Business Media, 2009.

[67] D. Zwillinger and S. Kokoska, *CRC standard probability and statistics tables and formulae.* Crc Press, 1999.

[68] S. W. AuYeung, “Finding probability distributions from moments,” *Master’s Thesis, Imperial College, London*, 2003.

[69] K. K. Lai, L. Yu, and S. Wang, “Mean-variance-skewness-kurtosis-based portfolio optimization,” in *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS’06)*, vol. 2, pp. 292–297, IEEE, 2006.

[70] J. Taylor, *Introduction to error analysis, the study of uncertainties in physical measurements.* 1997.

[71] G. Hackeling, *Mastering Machine Learning with scikit-learn.* Packt Publishing Ltd, 2017.

[72] J. S. Richman and J. R. Moorman, “Physiological time-series analysis using approximate entropy and sample entropy,” *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 278, no. 6, pp. H2039–H2049, 2000.

[73] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[74] CISCO, “Chapter: Intrusion detection system module,” 2019. Available at https://www.cisco.com/c/en/us/td/docs/routers/7600/Hardware/Hardware_Guides/7600_Series_Router_Module_Guide/modguid/09intrus.html, access date 2019-12-10.

[75] Y. Zhang and V. Paxson, “Detecting backdoors.,” in *USENIX Security Symposium*, Citeseer, 2000.

[76] J. Ilonen, “Keystroke dynamics,” *Advanced Topics in Information Processing–Lecture*, pp. 03–04, 2003.