Covert Timing Channel Attack Detection and Localization Using Machine Learning Frameworks

by © Shorouq Al-Eidi

A dissertation submitted to the School of Graduate Studies in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

2021

St. John's Newfoundland

Abstract

A covert timing channel is method that utilize to bypass security rules and transfer illegal data across several networks. It conceals critical data from the networks it targets by using traffic inter-arrival times. These channels are utilized by hostile tactics in a variety of damaging situations, including the exposure of economic and military secrets. Because they are not utilizing a new system, existing computer systems may be targeted to spread malware or worms without being discovered. Data leakage from cyber-attacks is on the rise, making the use of covert timing channels a significant network security issue that is getting more complex and pervasive.

Therefore, identifying and mitigating the usage of covert timing channels is crucial in today's information technology architecture and network security. Many private and public companies are trying to develop techniques for identifying and eliminating covert timing channels. These approaches would benefit from an information security decision support system that was developed on top of them to assist protect the IT infrastructure.

This dissertation makes significant advances fully automated covert timing channel identification and reduces the amount of data that might be transmitted over such channels. It provides a range of dependable and quick detection methods for successfully thwarting hidden timing channels. Images and sequential time series are used in these detection systems and different machine learning and deep learning methods. This study varies from others in the recent literature in the following ways: it combines different input data with various supervised and unsupervised machine learning methods, achieving noteworthy results, and giving important insight into how other solutions are used to develop realistic detection methods for detecting such channels in a variety of applications.

The dissertation also introduces a novel method for precisely identifying traffic flow segments carrying covert communications. This accurate identification substantially minimizes overt traffic interruptions caused by non-malicious apps. Furthermore, it enhances the Quality of Service (QoS) that is compromised when the whole traffic flow is lost since it is highly disruptive to the QoS of the overt traffic of legitimate applications, which may include the majority of the packets in the dropped flows.

The performance of the proposed methods in this research was tested and compared using various configurations of covert timing channel attacks ranging from simple to stealthy channels based on various sophisticated defensive strategies. These assaults also made use of various sizes of hidden messages. Our comparative analysis demonstrates a possible path for developing effective covert channel detection models utilizing a variety of input data and techniques, eliminating the requirement for robust and diverse concealed network traffic behavior.

Acknowledgements

I've spent almost five years of my life sitting in my office working on my thesis, building up to this moment, when I write the last lines of my thesis. This lengthy trip has been pleasant and unforgettable for me thanks to the support of my supervisors, family, and friends. As this chapter of my life draws to a close, I would want to thank many people who have helped shape who I am today.

First and foremost, I want to express my gratitude to my supervisors, Dr. Yuanzhu Chen and Dr. Omar Darwish, for their support during the program. Dr. Chen, his support and confidence in me, led me through difficult moments and significantly enhanced my graduate experience.

When I was nearly lost, Dr. Omar Darwish helped me find my path. He created numerous possibilities for me, even some I believed were unattainable, allowing my academic research to thrive. He is receptive to new research ideas and has given me a lot of leeway to experiment with new concepts in my study area. Because of him, I eventually fell in love with my study topic and have since really loved working on my thesis.

I would also want to thank Dr. Cheng Li for his ongoing assistance in guiding me through my studies at Memorial University.

I would want to thank my parents, sisters, and brothers for never giving up on me, even though they were hundreds of miles away, and for always being there for me when I was irritated or homesick. I would not have achieved as much in my life if it hadn't been for their unconditional love and unshakable faith in me.

Finally, I want to express my heartfelt gratitude to my husband and kids; they are the real cause for every accomplishment in my research trip. They are always there to cheer me up and stick by me through good and bad times; nothing is complete without them. They have always been sympathetic and helpful when I have been under stress, and they have always accommodated me while trying to meet deadlines. I dedicate this work to them as a thank you for all they have done.

I would like to dedicate this thesis to Zain's AlEidi soul

Contents

List of Tables

List of Figures

Chapter 1

Introduction

How important is our personal information's protection, and why leaking such information could put our daily financial processes at risk? In today's highly developed Internet technology, we may access various online services at any time and from any location. These services necessitate the sharing of personal information. While this kind of development improves various online services, it exposes our data to danger and vulnerability. For instance, in the event of data leakage in a hospital system, the attackers can use the data to hack associated bank accounts [1]. The main reason behind the vulnerability of leaking such personal data; the common mistake behind setting password keys inspired by birthdays or important events in peoples' lives. [2].

Fortunately, recently network defense tactics are becoming more stringent, making it harder for attackers to penetrate without detecting from network security systems [3]. However, stealing information from computers remains a popular target for attackers. And new techniques are being developed daily to serve malevolent objectives. There is a specific very high concealment technique known as *covert channel* among those numerous ways of breaching. The attackers establish secret communication using this technique by manipulating the time of packets that are transferred between the sender server on the target system.

Lampson proposed the covert channel in 1973 [4], with the primary goal of transmitting secret information between communication parties unusually without being observed by onlookers. The covert channel developed with the advent of the Internet as a channel that sends covert communications in violation of the communication security system in a network environment, such as intrusion detection systems and firewalls. The covert routes not only guarantee to hide the communication content, but they also safeguard the identities of both communicating parties.

Due to the covert channels' ability to transmit data without being detected by common network security countermeasures, it is utilized in various application areas. These channels may be used for sensitive information transfers such as military, communication in hazardous situations, or streaming application transmission in a suspect network. [5, 6].On the contrary, covert channels may be utilized by people with malevolent intents to communicate and share information. Coordination of acts of violence and terrorism and control protocols may be utilized inside covert channels to enable attackers to upload a newer version of a malware attack and choose different encryption or covert-signing method. The malware used covert channels on Gmail apps to hide such as command-and-control activities and evaded detection by security measures.

The well-known Distributed Denial of Service attacks on major internet companies like CNN and E-trade are a prime illustration of the risks presented by covert channels. These websites were automated with the thousands of agents that communicated using covert channels over networks. Another illustration of its risk is the 2002 allegations of suspected covert, hidden communication of plans or instructions to terrorist group members operating inside the united states through the Internet. Many of these messages are believed to have been transmitted in the covert channel and concealed within seemingly innocuous files [7]. As a consequence, identifying hidden channels that may compromise security measures if used maliciously is challenging.

There are two types of covert channels based on the kind of network resources utilized for communication. [8, 9]:

- *Covert Storage Channels (CSCs):* The secret information is concealed in covert storage channels by utilizing payload fields in the protocol [10] such as packets header fields, IP Identification fields, and Initial Sequence Number (ISN) to hide and send covert information.
- *Covert Timing Channels(CTCs):* Covert timing channels encrypt secret information and enable covert communication by using the time interval between packets [11–13].

An attacker that emphasizes stealth will benefit greatly from CTCs such as i) the channel may be part of a normal traffic network. As a consequence, by establishing a network connection, the attacker avoids arousing suspicions; ii) the packet content itself does not need to alter in any manner, reducing the chance of detecting by packet header analysis or deep inspection of a packet; iii) The stream's network destination does not have to be the covert receiver. To monitor the time information, the receiver may be placed anywhere along the network route.

Due to these benefits, an observer may not be having any information to gain knowledge that information communication exists. With the rising of computers usage in everyday activities and the complexity of protocols, CTCs are becoming an imminent threat to the confidentiality and integrity of information. Furthermore, with the increasing sensitivity of data in various application domains, leaking confidential data can have dangerous to the institution whose data was leaked.

1.1 Research motivation and problem definition

For various security reasons, covert channels represent a system security hazard. Of course, the first reason is a *secrecy issue*, as hidden channels might be utilized for the secret transmission of vital information. This concerns major organizations that want to keep business secrets secretive. This confidential information can get into or out of the organization through covert channel contact. The latest concept about cloud computing is this case. As companies store enormous quantities of data on the cloud, they need to ensure the cloud is safe. Organizations should use preventative techniques and detecting methods to protect their privacy against assaults and breaches.

Another reason is an *economic concern*, given that covert channels may use an existing system to transmit data without having to pay for the service. This is usually the situation when a Trojan Horse infects a computer. Furthermore, hidden channels usually utilize the system operations and resources, which causes degrades the system's performance. For these and other reasons, the US Department of Defense and National Computer Security Center [14, 15] have incorporated covert channel analysis in their evaluation criteria for classifying secure systems.

Defenses may be classed as either a preventive or a detection procedure against covert channels. A broad spectrum of preventive strategies was offered to minimize the usage by clearing off the plausibility of a hidden channel or decreasing the channel's ability by reducing the hidden channel bandwidth at a reasonably low rate. The channel is not used to transmit information properly. Prevention occurs at the network's edge and is used to interrupt traffic flows that may include hidden information about all network activity [16]. While prevention may successfully eliminate covert channels, it significantly impacts applications that need a high degree of Quality of Service (QoS), such as streaming video.

Covert channel detection methods are frequent techniques in network systems used to monitor harmful activities. It is desirable to detect hidden channels. [17] for four reasons: 1) Detection serves as a disincentive to covert channels by providing a method to discourage their use. [18]. 2) The majority of covert channel identification methods depend on information from system analysts. As a result of human error, many hidden channels may go undiscovered. Detection can help in recording channel activities. 3) Covert channel protection may be costly and affect service quality for high-performance systems [19]. 4) At the commencement of the elimination phase, detection techniques may be employed. The elimination methods will only be activated if there is any abnormal behavior in networks. For these reasons, detection methods that can identify and detect network traffic carrying concealed information are appealing options.

Researchers have contributed significantly to detecting covert communications, with their research primarily focusing on CSCs. In CSCs, the covert channel behavior is bounded by the communication platform's protocols, making identifying them less difficult. CTCs, on the other hand, exhibit stochastic behavior (resulting from overt traffic behavior that changes depending on network applications), making detection more difficult. CTCs, therefore, constitute a grave danger to cyber-security and feed the need for a reliable and sophisticated detection model to identify high-confidence CTCs with dynamic network environments. CTCs, as they are major dangers to cyber-security, are focused on in this dissertation.

CTC detection has been the subject of many research investigations. In general, the majority of this research suggested detection methods based on statistical analyses to differentiate covert from overt activity. [20–25]. The vast of these statistical tests begin by monitoring network traffic behavior, then extracting statistical features of covert and overt traffic and comparing those attributes to detect anomalies and uncover concealed communication. While these methods have shown good results, they do have limitations, which are discussed below.

Most methods are designed only to detect one or two CTCs and are not efficient for detecting other channels. While broader CTC detection methods are designed to distinguish more than one type of CTC, they are too sensitive when faced with the high variation of network traffic. Furthermore, broader methods cannot be applied to online network traffic with higher QoS requirements, especially when the detector requires information about the covert message embedding procedure to identify which statistical test to trust. Designing a broader detection method is not practically possible because it tries to detect covert communication by testing the traffic against all possible cases of covert channels, which needs more resources and processing time.

The majority of existing statistical techniques presume that CTC detection is feasible with significant quantities of hidden traffic. The training data for such techniques include at least 2,000 covert packets of network traffic. Such a huge amount of covert network traffic could not be accessible, especially when CTCs are utilized to leak confidential data such as user passwords, which are commonly less than 200 bits long. Furthermore, when the quantity of covert messages injected into overt traffic is little, the statistical detection technique fails to identify CTCs. If a small covert message size is utilized, the distribution of hidden information in network traffic may not be representative of the population. Another major limitation of current detection methods is that when an attacker tries to mimic overt traffic in order to evade detection, they become less reliable and unpredictable. Due to the significant variance in network data and its pattern changes over time, statistical detection methods would be ineffective in detecting CTCs.

Due to the viability of recognizing the presence of CTCs, machine learning has also been utilized in numerous detection methods [9,26,27]. These methods train and construct classifier models based on network statistical traffic characteristics using a labeled collection of overt and covert traffic flows. Following that, the models are utilized to classify new traffic flows as either overt or covert. While machine learning methods have shown good results, they still have disadvantages, such as the need for long calculations and more knowledge of network traffic statistical features to treat classifiers. Furthermore, when the message size is decreased to a smaller size, these methods can no longer adequately classify covert traffic because the size of the traffic becomes too small to represent the covert communication population accurately.

In addition, current detection methods lack a mechanism for identifying the hidden part of traffic (i.e., a collection of packets) inside a traffic flow. One of the essential objectives is identifying the parts of communication flows containing the concealed messages. It can drop just the harmful part of traffic flows while allowing the rest to pass through.

Most of the constraints of existing detection methods, as mentioned above, must be addressed to identify CTCs effectively. In this research, we aim to offer novel techniques for overcoming the limits of existing methodologies and detecting the existence of CTCs regardless of the time scale on which they are buried in the total data stream. The section that follows covers the dissertation's main objectives and contributions.

1.2 Research objectives and main contributions

This dissertation provides investigative assistance for the confidentiality of data; This entails examining CTC communication through the lens of digital forensics. Digital forensics is a post-mortem examination [28]. The main objective of digital forensics is to identify and establish the existence of a security policy breach. Then, covert mitigation defense mechanisms or traffic blocking applications are used to stop the flow of covert traffic. Since we analyze after the event, performance is irrelevant for designing detection methods for covert communications channels.

The detection approaches presented in the dissertation use different machine learning algorithms and typologies of neural networks arranged to provide powerful network classifiers suitable for most of the tasks characterizing CTC activities. In conclusion, the major contributions of this dissertation are described below:

- Generating a covert timing channel called Binary Covert Timing Channel (BCTC). This channel uses packet timing to hide the covert data over the network effectively.
- Examining the impact of varying time delays on covert channel detection and determining the time delay threshold that correctly distinguishes covert traffic from overt traffic and aids in assessing the degree of the security risk posed by covert channels.
- Generating datasets with three unique time delay configurations to simulate three distinct kinds of covert timing channel assaults with various degrees of complexity, including different covert message sizes.
- Introducing a new technique for viewing traffic inter-arrival times that transforms them into colorful two-dimensional images. This conversion transformed the covert channel detection issue into an image classification problem. The

properties of time series data may be determined using this technique by analyzing the colored image for various aspects such as color and texture at the image's corresponding places.

- Presenting a reliable SnapCatch detection model to classify and locate covert timing channels using image processing and machine learning techniques. The proposed approach employs accurate and robust image-based features rather than statistical-based features in the classification process. This option offers a generic model for detecting covert timing channels.

- Proposing a new method for locating traffic segments using image processing and machine learning techniques seems to be covert channels. This method enables warnings about the existence of covert timing channels to activate covert mitigation defensive mechanisms or traffic blocking applications, significantly increasing service quality as it declines by discontinuing the whole traffic flow.

- Using a convolutional neural network to improve the performance of the SnapCatch detection model without the need for pre-training (CNN). The invention of the 2D-CNN technique allows for the automated extraction and selection of high-level features that improve the identification of covert timing channels while keeping execution time to a minimum.

- Developing a covert timing channel detector based on utilizing a recurrent neural network (RNN) with sequence traffic inter-arrival times to build temporal feature representations directly from data. The proposed model learns features directly from raw data and then finds pattern sequences within sequential data, consistent with their high detection capabilities. In real-time applications, the proposed method provides an efficient solution.

- Developing a covert timing channel detector based on sequence traffic inter-arrival timings using a one-dimensional convolutional neural network (1D-CNN). The proposed model is used to find spatial pattern inside the sequential data and extract representation features that determine how the temporal modeling of spatial characteristics affects the performance of the covert channel detection model.

- Developing hybrid detection models based on combining network models of RNN and 1D-CNN. The hybrid models integrate various information scales and investigate whether further improvements are required to provide a more accurate covert timing channel detection model. These hybrid models enable researchers to examine how the arrangement of different network layers impacts the performance of detection models.

- Developing and evaluating autoencoders as one-class and multi-class detection models with two input data types: images and sequential time series. This comparative assessment research sheds light on how various autoencoder methods can be utilized to construct an effective unsupervised covert timing channel detection model.

- Providing application guide for deep learning-based covert timing channel detection in detail. This guidance will help the network security sector because it offers a feasible path for developing successful covert channel detection models based on various methods, reducing the need for more varied covert network traffic behavior.

1.3 Thesis outline

The remainder of this dissertation is organized as follows.

The second chapter looks at the analytical evaluations of covert timing channel design methods, emphasizing each method's fundamental components and performance. Furthermore, this chapter provides a brief overview of detection techniques for such channels, which are classified into two types: statistical and machine learning. Finally, it provides cybersecurity detection model approaches based on image processing, machine learning, and deep learning techniques. In the next chapters, we lay the groundwork for future research by comparing existing research results.

Chapter 3 introduces the design and implement a covert timing channel that uses timing packets of legitimate traffic to hide covert data. This chapter also discusses finding the threshold of time delay and exploring how this threshold affects the channel efficacy based on channel accuracy and bit rate. After studying the performance of the proposed covert channel and generating different attack configurations, Chapter 4 depicts a SnapCatch detection model. This model improves on existing statistical-based detection approaches by identifying hidden time channels in a new manner utilizing image processing and machine learning techniques.

Chapter 5 presents an entirely new approach to deep learning, which relies on the usage of a 2D-CNN method to enhance the SnapCatch model's performance and utilize the resilience of CNN to extract image features automatically. In the same chapter, we evaluate the CNN model by conducting experiments to measure its detection accuracy and robustness to detect covert channels.

Chapter 6 introduces a deep learning framework for detecting covert timing channels using time sequence data with RNN and 1D-CNN to use these techniques' robustness to learn a hierarchical features representation and automatically detect such channels from raw data. The same chapter also presents the hybrid detection models can enhance the model's ability to identify hidden timing channels by extracting detailed spatial-temporal information from the raw data. When developing an unsupervised covert timing channel detection model, a comparison of five autoencoder models as a one-class and multi-class classifiers are conducted in Chapter 7. This dissertation comes to a close in Chapter 8 with a summary of the findings and recommendations.

Chapter 2

Related Work

This chapter starts with a quick rundown of CTC design problems, then a short overview of statistical and machine learning approaches to CTC detection. Finally, the application of image processing, machine learning techniques, and deep learning approaches is emphasized in traffic network classification and cybersecurity threat identification.

2.1 Covert timing channel design

CTCs are divided into two categories: passive and active. Passive channels generate no extra network traffic. Instead, they encode the message using genuine traffic's inter-packet delays. Active channels, on the other hand, generate their network stream using specifically designed inter-packet delays. Passive channels seem more difficult to detect because the attacker does not need to establish a new network connection channel; nevertheless, their capacity is limited by the data network traffic.

The most popular CTC construction technique is based on inter-arrival time. Most research on CTCs relates to it since it sends messages by changing the time interval between network packets. The On-Off channel, BER channel, GAS channel, ZAN channel, Time Replay Channel, Jitterbug, Model-based, and other highly representative covert timing channels are briefly discussed in this section.

2.1.1 On-Off channel

Cabuk et al. [20] presented a channel based on using a certain time interval to organize sending the covert data between the sender and receiver. To transmit a covert bit 1, the covert transmitter sends a packet in the midst of the time interval, while when sending bit 0, the covert transmitter stays quiet and does not send any packets. Receiving a packet during the time interval was regarded as covert bit 1 on the receiver side; while aren't received any packets within this period, the receiver interprets it as a covert bit 0.

2.1.2 One threshold (GAS) channel

Gasior et al. suggested the structure of the GAS channel [29]. The threshold value is the key to the GAS channel. When the inter-arrival times between the two packets were higher than the threshold, the sender and receiver signaled that the bit transmitted was 1. While if it is smaller than the threshold, the bit transmitted was 0.

2.1.3 Fixed time interval (BER) channel

Berk et al. [30] described a simple binary fixed delay technique for covert timing channels. In this channel, the sender sets two time intervals values: t_0 and t_1. After then, the secret information must be encoded and sent in binary form. When 1 bit was sent, the delay between packets was chosen from t_1; when zero bit was sent, the delays between packets were chosen from t_0. The average time delay of overt packet arrivals was used to determine the values of t_0 and t_1. This average was used as a cut-off point to categorize the delays as zero (when they were less than the average) or one (when they were more than the average). The recipient then follows the identical steps to get the secret information.

2.1.4 Time Replay channel

Cabuk et al. [22] created the time replay channel to avoid covert channels from being detected using statistical tests. Before establishing a threshold, the sender in this channel collected a collection of packet inter-arrival times from overt communication. Based on these criteria, the sender divides the time interval into two categories. Inter-arrival times are larger than the threshold from a group (T_1), while time intervals less than the threshold constitute a second interval (T_2). The time delay for an interval between the two packets to being sent is determined from the T_1 when the sender sends 1. If the sender decided to send 0, the delay time for the interval between the two packets was picked from $T2$.

2.1.5 Jitterbug

Shah et al. proposed a covert timing channel using a hardware device [31]. A Jitterbug is a device that serves as an interface between a machine and its keyboard. Because of its position, it may selectively record and delay each keystroke made by the user. Every keystroke in a device makes a packet sent in SSH's interactive mode. The advantage of using a Jitterbug was that the computer's integrity was never compromised. Jitterbug determined a timing window w to choose the time delay for encoding each binary symbol. Jitterbug also uses a random sequence, s, to avoid inter-packet delays from congregating around of w. Every value in s falls between $[0, w-1]$ and $[0, w-1]$. A Jitterbug encoded the symbols in binary code by delaying a keystroke such that the resulting inter-packet delays met the requirements.

$$(IPD_i - s_i) \mod w = \begin{cases} 0 \pm \frac{w}{4} & \text{for 0 bit} \\ \frac{w}{2} \pm \frac{w}{4} & \text{for 1 bit} \end{cases}$$

2.1.6 Model based channel

Gianvecchio et al. developed a model-based covert timing channel [13]. The authors developed a technique for creating a covert timing channel with the similar statistical behavior of real-world network traffic. To develop the MBCTC, they began by studying a certain kind of traffic and putting it into distribution. The covert message is concealed in symbols translated to inter-arrival intervals using the inverse distribution function. They decoded the data using a cumulative distribution function based on a distribution that would alter over time to represent any changes in traffic times.

2.1.7 Other covert timing channels

Hovhannisyan et al. [32] developed a cover channel that has the potential to increase covert capacity substantially. They created a method for transmitting concealed data via channels that specifically communicate information. They detailed the design of the covert channel for two protocols, UDP and TCP, and utilized a novel method to decrease the bit error rate. Their channel is more capable and has a lower bit error rate than other channels. This study also showed that the CTC is more hazardous than previously believed, requiring more countermeasures.

Packet order was also given as an example of a CTC entity on the Internet of Things. In packet-reordering-based CTCs, covert information is encoded using the order of packets as a certain number of a packet stream between communicating parties, which is achieved by embedding covert messages inside the ordering of overt traffic. El-Atawy and Al-Shaer [33] proposed a covert route that increased the possibility for evasion by utilizing certain permutations of packets to enhance resilience and mimic overt traffic distribution. They selected certain combinations of successive packets to improve channel stability and reliability. In order to stay undetected, they are also altering the distribution of their channel usage to imitate real internet traffic. An attacker will have to pay a significant price to expose the CTC because of the high expense of buffering and processing out-of-order packets in the midst of a huge amount of background traffic. The right coding scheme is selected improves the properly received code-words under typical working circumstances, reducing error rates to 10%.

2.2 Covert timing channel detection methods

Because the basic idea of CTC implementation is to encrypt the secret data using the packet's inter-arrival times, most CTC detection researches focus on analyzing the traffic inter-arrival times. CTC defense techniques are classified into two types: prevention and detection. Prevention aims to destroy a channel's potential by reducing its capacity and making it impractical. Detection techniques, on the other hand, attempt to identify currently operational hidden channels. This dissertation focuses on the identification of CTCs. This section discusses methods for identifying CTCs using statistical and machine learning technologies.

2.2.1 Methods for detecting CTC based on statistical tests

CTC detection methods are primarily concerned with the analysis of network traffic. Most CTC statistics-based detection methods analyze network traffic behavior, extract statistical features of covert and overt traffic, then compare those characteristics to find anomalies and covert communication. In this section, we examined the effectiveness of several statistical techniques for detecting CTCs in the literature.

Cabuk et al. [34] proposed Sigma similarity as a shape-based detection technique. This method computed and compared input symbol data from normal network traffic. A large percentage of deviations below the Sigma value suggests the presence of hidden channels.

Berk et al. [30] proposed a method for determining covert channels, which use traffic inter-arrival times. This method was based on the assumption that an attacker sends covert data using the maximum bandwidth of the covert channel. After sending the data, the inter-arrival time's probability distribution was computed and compared to the actual distribution of normal network traffic. Using this method, the traffic data with a bimodal or multi-modal distribution would indicate the presence of a hidden channel.

In [21], A binary CTC was identified using the histogram of normal and covert traffic. Simple CTC algorithms may be determined using traffic distribution and histogram testing; however, these tests cannot identify robust and sophisticated CTC algorithms. Consequently, more effective statistical tests for identifying such channels, such as Kolmogorov Smirnov, Regularity, Entropy, and Corrected Conditional

Entropy, were created and researched. The remainder of this section delves into the details of each of these examinations.

2.2.1.1 Regularity test

In [22], Cabuk et al. used the fingerprints left in the traffic stream by the covert channel embedding procedure. To assess the regularity of traffic, they devised a metric based on the variance formula. The measure was based on the assumption that CTCs would provide regular patterns in packet inter-arrival times, while overt traffic might be random and lack any patterns. The inter-arrival timings of packets were split into neighboring blocks for the regularity test, and the regularity score for each block was calculated. To compute the score, a sample block was split into n successive sub-blocks, and the standard deviation $\sigma_i; i = 1$ to n was measured for each. The relative difference in deviation was then computed block by block, then normalized individually for each sub-block. Finally, the regularity score was computed using the standard deviation values of all normalized data.

$$R = Std\left(\frac{\sigma_x - \sigma_y}{\sigma_x}\right) \forall\, x, y \tag{2.1}$$

If the regularity score is low, that indicates the traffic flow is very regular, suggesting the possibility of CTC presence.

2.2.1.2 Kolmogorov Smirnov test

Gianvecchio et al. [35] used the Kolmogorov Smirnov (KS) test to determine if the distribution of a set of covert inter-arrival times is similar to normal traffic set. A change in the distributions would indicate the existence of a CTC in the set sample. The KS test is a non-parametric method used to determine if two sets are drawn from the same distribution. One of the most significant benefits of this test is that it makes no assumptions about the sample distribution.

$$D_{KS} = sup_x |F(x) - G(x)| \tag{2.2}$$

2.2.1.3 Entropy tests

Gianvecchio et al. [35] used entropy of a network flow as a measure to detect CTCs. Entropy was utilized to identify changes in a channel's shape caused by timing channels, while conditional entropy was used to determine its regularity. The authors predicted that combining these two methods would effectively identify known timing channels while being robust new channels in the future. The following section describes the methods for calculating entropy and correcting condition entropy.

- **Entropy**

Given a sample of successive inter-arrival times of packets from a network flow, each packet was mapped to one of a finite set of symbols, with the probability of each symbol as p_i. The sample's entropy, H, is then computed as:

$$H = -\sum_{i=1}^{m} p(X_i) \log_2 p(X_i) \tag{2.3}$$

This equation indicates that the entropy is high when all the symbols are equally probability.

The use of entropy to identify CTCs necessitates a difference in the probability of the covert and legitimate streams. The covert stream's entropy is reduced as a consequence of the difference. Because each packet value in a covert binary channel translates using two symbols for the concealed message, with much less than would be anticipated normal traffic. As a result, in the CTC traffic, the probability for all other symbols is close to zero.

The only method to guarantee that this probability difference results and maximum the entropy for the covert data presence. Gianvecchio et al. did this by creating a traffic histogram with bins matching the n potential symbols. The range of bins was determined using a large set of training data packets. Each bin has an equal number of training packets to achieve an equal chance for each symbol. CTCs are detected by testing samples of inter-arrival times to see whether their entropy is lower than that of overt traffic.

- **Conditional entropy** Because the entropy of the covert packet stream may be identical to that of the actual packets, utilizing entropy to identify CTCs has limitations, particularly when CTC can create traffic with a structure similar to actual traffic. Consequently, Gianvecchio et al. suggested conditional entropy as a method for detecting regularity in a stream by detecting unexpected traffic correlations. A series of questions accompany the symbol (bin number) X. Such that x_i is the ith value in the sequence, the conditional entropy of symbol Xx_i given is:

$$H(x_n|x_1,\ldots,x_{n-1}) = H(x_n|x_1,\ldots,x_n) - H(x_n|x_1,\ldots,x_{n-1}) \tag{2.4}$$

- **Corrected entropy**

Due to issues with small sample sizes, Gianvecchio et al. advocated for the use of adjusted entropy rather, which is computed as follow:

$$AE_{En} = E + pre(x_1) \times E \tag{2.5}$$

Where pre (x_1) is the proportion of training bins that include precisely one inter-packet delay. For limited samples, however, the precise entropy rate cannot be obtained. To solve the issue, Gianvecchio et al. calculated the corrected conditional entropy, which is defined as :

$$CCE_{(x_n|x_1,\ldots,x_{n-1})} = CE(x_n|x_1,\ldots,x_{n-1}) + prec(x_n)E(x_1) \tag{2.6}$$

where $E(x_1)$ is the first order entropy, and $CE(x_m|n_1,\ldots,x_{n-1})$ is the conditional entropy of sequence of random variables.

2.2.1.4 Other detection tests

The KullbackeLeibler test was used as a CTC detection method [36]. The KL divergence test was calculated from two probability density functions $f1(.)$ and $f2(.)$ as shown in the equation below:

$$D_{KL}(P||G) = -\sum_{x} f1(x) \log\left(\frac{f1(x)}{f2(x)}\right) \tag{2.7}$$

The KL test specifies the minimal extra information measured based on a logarithmic scale that is required for $f2(.)$ to model $f1(.)$ correctly. If the test score is more than a specified threshold number, it suggests the possibility of a CTC. Similarly, in [37] the authors computed the KL divergence statistic for the covert and overt communication streams. If this KL value was more than the threshold, the traffic was labeled as covert. The authors of [24] expanded on the work of Gianvechhio et al. by adding two new statistical tests for identifying CTCs, the KL divergence, and Welch's t-test. They demonstrated that since it was a non-parametric test based on the difference of means, Welch's t-test was better suited to identify the Jitterbug method.

Darwish et al. [38] developed a hierarchical entropy approach for identifying the presence of CTCs in the flow of inter-arrival times to address the limitations of utilizing flat statistical-based analysis for detecting CTCs. Their findings revealed that the approach delivers considerably higher accuracy than flat entropy solutions. Darwish et al. [39] presented a method to enhance the speed of CTC identification by utilizing the hierarchical entropy. To parallelize the detection procedure, they used the

MapReduce method. When compared to the sequential hierarchical entropy method, the hierarchical entropy algorithm using MapReduce showed a strong capacity to identify CTCs.

To summarize, several detection methods use statistical tests published in the literature to identify CTCs. However, due to legal traffic's great variety and complexity, current statistical methods are not trustworthy and effective enough to identify such channels, particularly when the covert data size is small and channel behavior resembles normal traffic.

2.2.2 Methods for detecting CTC based machine learning

Machine learning algorithms have been utilized in many CTC detection approaches due to their ability to identify the existence of CTCs effectively. In general, machine learning techniques utilize statistical features obtained from overt and covert traffic flows to train machine learning classifiers, which are then used to identify new traffic flows to either overt or covert traffic.

Zander et al. [40] used the decision tree classifier as a CTC detection technique. The decision tree was trained utilizing various statistical characteristics collected from traffic flows in their method. The model's efficacy was evaluated using a collection of both overt and covert traffic to detect a pattern in packet inter-arrival times. The model was shown to be successful in identifying CTCs based on the findings of the assessment. Likewise, Iglesias et al. [27] also used a decision tree classifier to identify CTCs by utilizing a collection of statistical characteristics of traffic as features. To minimize significant computing expenses during the detection phase, the model was trained offline. In addition, decision trees have been used in [41] to detect CTCs.

Félix et al. [26] used unsupervised machine learning to explore whether CTCs may be identified as strong abnormalities based on their statistical characteristics, such as outlier algorithms. Their findings revealed that CTC-containing flows seldom had high values or unusual density variations. Because all covert flows are (mild) distance-outliers, but not all are covert flows, unsupervised techniques are ineffective in detecting them.

Félix et al. [42] used supervised and unsupervised machine learning techniques, a CTC detection system was created. They discovered that CTCs had substantial histogram distance-based outliers, but they couldn't tell them apart from ordinary traffic owing to the vast diversity of forms in regular traffic. After doing an extensive study, the authors concluded that combining supervised and unsupervised techniques is the best way to build accurate and reliable CTC detectors (also known as semi-supervised methods).

The Support Vector Machine (SVM) learning classifier has also been widely used in the CTCs detection research domain. This is because SVM has an exploration and classification power beyond checking for statistical properties of traffic. Sohn et al. [43] showed that simple covert channels encoded. An SVM classifier was used to successfully identify a simple covert channel contained in the sequence number fields of TCP/IP protocol headers.

Recently, Shrestha et al. [9] used the SVM classifier; a reliable CTC detection method was developed. They used four kinds of statistical characteristics produced by four covert timing channel methods to construct their model. Machine learning methods fared well in identifying CTCs, according to the method's assessment findings. In [44], To identify different types of CTCs, a novel detection method based on wavelet transform features and SVM classifiers were suggested. To optimize entropy, these features were input into SVM after being acquired at various wavelet levels. The authors devised a sliding window-based method for detecting complicated traffic with a variety of CTCs.

Deep neural networks have also been used for CTC detection. Darwish et al. [45] suggested a method for developing a CTC detection classifier using deep neural networks. The hierarchical statistics-based approach was used to train this classifier using a collection of statistical characteristics derived from the flow of inter-arrival times. Their findings revealed that deep neural networks outperformed SVM models in terms of accuracy.

The majority of machine learning-based methods in the literature are based on statistical network traffic characteristics, such as CCE values of packet inter-arrival time. Such techniques are very restricted in CTC detection, particularly when the hidden message size is small. Another significant drawback of existing techniques is that they become unstable and unpredictable when the attacker attempts to impersonate genuine communications to avoid detection.

2.3 Detecting attacks based on image processing and machine learning techniques

Several visualization methods for attack analysis have recently been suggested, allowing human analysts to examine the characteristics of assaults such as malware visually. Conti et al. [46] developed a technique for visualizing huge quantities of binary data by transforming it to a grayscale image. Using a number of graphical components, this technique allows you to examine binary byte information from malware samples. Their results may aid researchers in navigating unfamiliar locations. It may also be used to highlight significant data points while ignoring less important ones. Anderson et al. [47] proposed a visualization model to detect malware malicious software samples based on the similarity between malware heatmap images.

Nataraj et al. [48] presented a paradigm for converting binary executable data into grayscale images in bytes. They used the Gist texture feature extraction method to calculate texture features from the images after they were created. Finally, the k-Nearest Neighbor classifier was utilized to identify malware samples by comparing the characteristics of the malware grayscale images. The classifier detected several kinds of malware with an accuracy of 97.18%.

Similarly, Aziz Makandar et al. [49] converted the malware binary code into 2D grayscale images and standardized it into x dimensions before using an SVM classifier to categorize the photos and detect malware. The classifier detected 24 malware families with an accuracy of 92.52%. Han et al. [50] proposed a malware detection method by converting the binary code of malware into color images and then using an image processing method to classify them. Daniel et al. [51] created a bio-inspired parallel implementation using binary 2D images to identify the geometric objects of the homology set.

Image processing technologies have proved effective in evaluating and categorizing massive amounts of data in various assault detection methods. Compared to conventional methods, this methodology has a low computing cost, is easy to implement, and has a high classification accuracy. For these reasons, using image processing methods is a feasible option for identifying CTCs correctly. We are unaware of any use of image processing or machine learning to identify CTCs.

2.4 Deep learning techniques for detecting attacks

Taking into account the current deep learning techniques for attack detection and adhering to the classification of prior work [52] deep learning techniques are classified into three types: unsupervised methods like autoencoders (AEs), supervised methods like convolutional neural network (CNN), and recurrent neural network (RNN), and various hybrid approaches. There are more classification categories, such as Al-Garadi et al. [53] classified deep learning methods depending on their applicability in cybersecurity. Recently, Berman et al. [54] the categorization of deep learning techniques based on attack types and the application of these methods to detect different attacks.

The development of various deep learning algorithms for attack detection techniques may be helpful in various ways. Manually labeled samples provide a wealth of information, allowing supervised learning techniques to be very precise. On the other hand, manual data labeling takes a long time, particularly when dealing with real-world network assaults. As a result, unsupervised learning techniques with no previous knowledge of attacks may outperform supervised learning methods, which is a major benefit. Techniques that include aspects of both conventional and hybrid techniques reduce training needs while maintaining high-performance levels. However, it is rare to use because of its often complicated structure, which requires a considerable amount of processing time. This section briefly discusses using deep learning methods (CNN, RNN, and AE) to detect malware, intrusion, and other threats.

2.4.1 Attack detection based on convolutional neural networks

The usage of Convolutional Neural Networks has aided image identification and categorization (CNN). The CNN weight-sharing network design reduces the complexity and weights of the network model. If the network input is a multi-dimensional image, this advantage is amplified. Traditional machine learning methods require time-consuming feature extraction and data reconstruction, which may be avoided by submitting the image to the network.

CNN is composed of many layers, and it recognizes two-dimensional structures that may be distorted in any manner without losing their original shape. Reduce network-training parameters using one of CNN's three suggested techniques: local receptivity, weight sharing, or pooling. It also uses spatial connections to decrease the number of learning parameters to enhance the backpropagation technique's performance. CNN's most notable feature is its capacity to learn feature hierarchy from large of data.

As a result, CNN has a variety of applications, including network intrusion detection and virus identification. In these applications, CNN is widely utilized. Wang et al. [55] converted traffic data to grayscale images and then used CNN to build meaningful feature extracting model automatically and efficiently from huge quantities of raw network traffic data. The average accuracy of classifiers was found to be 99.41% in experimental findings. Likewise, Vasan et al. [56] presented a malware detection approach, dubbed 'IMCFN,' that uses a CNN to transform traffic data into color images. They used data augmentation methods to improve the IMCFN model performance and deal with unbalanced label datasets. Their model produced the best results, with an accuracy rate of 98.82

Cui, Zhihua, et al. [57] suggested a new technique for improving malware variant identification using a CNN by treating traffic malware data as images. First, the harmful code was converted into grayscale images using this technique. Following that, the images were categorized by a CNN that automatically extracted the malware images' characteristics. Because of the CNN's efficacy and efficiency in detecting malware images, their model's detection time was considerably quicker than other methods. Their approach successfully addressed the issue of data imbalance across various malware families, achieving 94.5% accuracy with a fast detection rate.

Kumar et al. [58] developed a malware detection technique based on a CNN. The

CNN method was utilized in their research to identify unknown or novel types of malware utilizing the picture similarity technique. CNN was studied and evaluated using three different kinds of datasets. CNN achieved a high testing accuracy of 98Similarly, Ni et al. [59] A malware detection approach based on malware visualization and CNN has been presented. To convert comparable malware code to similar hash values, the authors employed a locality-sensitive hashing method. The comparable hash values were then converted into grayscale images that could be used to train the CNN method.

Wang et al. [60] presented a single-dimensional end-to-end CNN traffic categorization method. They can automatically comprehend how nonlinear input traffic is linked due to their approach, which combines feature extraction with a classifier. 1D-accuracy CNN's and recall rate are astonishing due to the model's usage of informative traffic data representations and fine-tuning techniques to enhance its capabilities further.

Yang and Wang developed a solution for the various attack in the wireless network and improved the detection capabilities of hostile infiltration [61]. This network automatically collects sample features from intrusion traffic data and utilizes a gradient descent method to enhance network parameters and bring the model closer. According to the study, this model produced fewer false positives than others.

2.4.2 Attack detection based on recurrent neural networks

The recursive neural network (RNN) is suggested as a neural network used to handle sequence data. Data is moved from the input layer to the output layer through the hidden layer in conventional neural network models, which examine just the impact of the present state of input data without incorporating information from past and future states. These models may perform substantial classification or identification tasks while also taking temporal sequence features into account. RNN is suggested as a kind of neural network structure with a memory function that considers past content states by including time-dependent data. RNN is therefore adept at handling time-series data.

However, there are some issues with RNN architecture design, such as gradient vanishing or gradient explosion, which results in the inability to recall or represent long data of time dependency. As a result, researchers created Long Short-Term Memory (LSTM) and a Gated Recurrent Unit (GRU) with gates design and memory cells that effectively maintain long-term dependence and connections from being lost by transferring essential information flow components.

Early, Staudemeyer [62] presented a model for detecting intrusions based on time series of known malicious activity and network traffic. The authors used LSTM for intrusion detection because of its outstanding ability to represent long-term dependent relationships. They came up with a four-memory block network. Each sub-memory is made up of two cells. The network was capable of balancing detection performance and computational cost. Because LSTM could evaluate and correlate connection in a time-varying way, this study's experimental findings showed that the LSTM model outperformed results.

Jiang et al. [63] presented an efficient attack detection approach in social networks using RNN that includes data pre-processing, feature extraction, and multi-channel detection. LSTM was employed in multi-channel processing to create a classifier used to distinguish attacks from regular traffic in order to save attack characteristics in input traffic data. The LSTM outperforms state-of-the-art techniques in accuracy terms by including a voting mechanism to decide if the incoming data is an attack or not.

Later, Krishnan and Raajan [64] was used RNN to build the framework of an intrusion detection system. During the tests, the suggested detection method could filter out assaults but failed to identify false positives. When compared to the baseline techniques, their suggested method has higher classification accuracy and lower time-consuming measures. Kim et al. [65] Create an LSTM classifier to identify infiltration. The performance of the LSTM model was achieved higher accuracy compared to other detection techniques.

Agarap et al. [66] proposed a GRU binary classification of intrusion detection model by changing the softmax classifier in the output layer with an SVM classifier. The model results revealed that the GRU-SVM model outperformed the GRU-Softmax model in terms of accuracy. Liu et al. [67] implemented an efficient and practical RNN payload classification approach to analyze payloads network traffic and detect attacks. Their model learns feature representations from original payloads without feature engineering and can detect from start to finish. The accuracy of the

RNN technique was 99.98%, which is better than other machine learning approaches. Le et al. [68] built an LSTM intrusion detection model and attempted to find the best optimizer for LSTM optimization. They evaluated six commonly used optimization techniques, Adagrad, Adamax, Adadelta, RMSprop, Nadam, and Adam, and then found that the Nadam optimizer was the most successful.

2.4.3 Attack detection based on autoencoders

To compress data, the Autoencoder (AE) employs a neural network design. It is conceivable that AE will compress the input using feature representation before rebuilding it in the final output. An encoder and a decoder make up autoencoder components. A feature-collecting encoder extract features from raw data, while a feature-recovering decoder reconstructs the data based on what was extracted. The divergence between the encoder's input and the decoder's output will decrease with time. The encoder successfully captures the data's core when the decoder reconstructs the data from the gathered characteristics.

AE is often used for applications like outlier detection and size reduction. Cybersecurity researchers use AE to explain aberrant behaviors in feature space representation, giving unknown threats the benefit of dynamical representation. Yu et al. [69] created a network intrusion detection model by stacking dilated convolutional autoencoders to extract relevant feature descriptors from raw network traffic data (DCAEs). DCAEs utilize unsupervised training to learn the hierarchical structure of features from a limited number of labeled data. By fine-tuning a backpropagation method learned from unlabeled data, they enhanced feature description capabilities. Because it uses raw network traffic data as well as unsupervised pretraining, its model is more adaptable to complex raw data.

Yousefi-Azar et al. [70] utilized AE structure to learn important features for various cybersecurity applications, which comprises of two phases model pretraining and fine-tuning since AE is capable of learning the possible representation of unknown assaults. The pretraining step is intended to find suitable settings for the fine-tuning stage. Stage coverage will be fine-tuned by providing feature descriptions for input data. Experiment findings showed that their feature representation utilizing AE structure with two stages might be utilized in various domains to significantly decrease feature dimensions and produce amazing outcomes compared to prior studies. Similarly, Farahnakian et al. [71] built a classification model to identify aberrant behaviors using a deep-stacked autoencoder structure. Using imbalanced data samples, this model obtained a high accuracy of 94.71%.

Sakurada et al. [72] proposed a nonlinear dimensionality reduction technique using auto-encoders in anomaly detection and compared it to the linear principle component technique. The comparative findings demonstrated that auto-encoders had better accuracy while requiring less complicated calculations. Choi et al. [73] developed an anomaly detection model using several AE topologies. They used a heuristic method to establish a reconstruction error threshold and find that their unsupervised anomaly detection technique beats existing clustering techniques.

To summarize, both supervised and unsupervised deep learning methods have been effectively used in the area of cyber-security. As a result, the combination of supervised and unsupervised deep learning methods is a promising option for identifying CTCs with high accuracy.

Chapter 3

Experimental Design of Binary Covert Timing Channel

This chapter discusses the developing of the binary matching covert timing channel (BCTC) that utilizes normal traffic timing packets to exfiltrate covert data between two parties communicating across a network. Insecure systems, BCTC uses traffic inter-arrival times to transmit covert information. Despite our best efforts, we were unable to locate a dataset that contained both overt and covert time channels. In addition, there was no software available that could generate both types of channels at the same time.

Our strategy was to create new timing channel software that could be utilized covertly as well as publicly. The process units in this channel represent how the traffic inter-arrival times were used to encode and transmit covert data between the sender and receiver.

The channel helps to create a software framework that can generate overt and covert traffic datasets for future study. Moreover, in detecting covert channels, this channel helps determine the packet time delay threshold that consistently differentiates between covert and overt transmission. Defining this threshold value is the first major difficulty in the detection process. Consequently, it may help measure the complexity of covert threats and evaluate the security risks associated with covert communications. Knowing this threshold may save time for future researchers who are researching the detection or mitigation of covert channels in different applications.

This chapter covers the secret channel's design. Following is an examination of various exfiltration trial time delays and how they impact covert channel detection. Finally, the experiment analyzes and reports on the effect of different variables on channel efficiency.

3.1 Covert timing channel design

Our covert channel design follows the literature, such as the Time-Replay [22] and BER [30] channels, by concealing the covert channel traffic inside overt network traffic utilizing packet inter-arrival times. The covert data can be sent Time-Replay and BER channels by replay recorded events using a single threshold value (the median arrival time) to create rules and two-time partitions for encoding binary data in the channel. By delaying the packet matching to the partition, the sender in this channel may transmit the binary symbols. To transmit symbol 0, for example, the sender selects a delaying time from the first time partition that reflects values smaller than the median of inter-arrival times. Similarly to sending 1, the sender selects a delaying time from the second division, representing values fourth of the median.

In our covert timing channel, we use a different strategy by using various threshold values depending on the mean and standard deviation of traffic inter-arrival times that can use for encoding the binary covert data and to determine the threshold that will aid in properly identifying the channel.

Within a security system, assuming two communication parties (the sender and receiver) can communicate and send information only by exchanging the time information of the packets send data from sender to receiver. The channels presented in this chapter conceal hidden information in overt traffic by modifying the transmission time of normal packets, as illustrated in Figure 3.1.

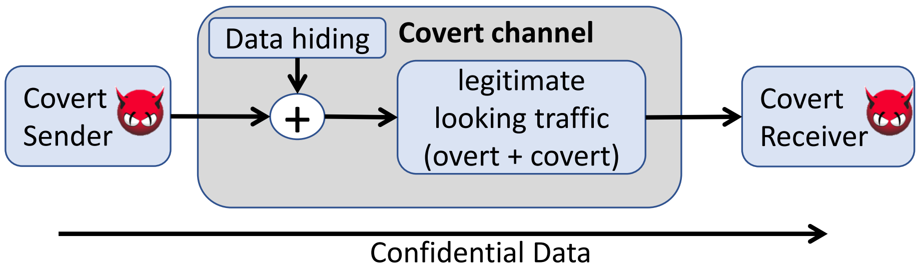


Figure 3.1: Covert communication model.

Adjusting the transmission interval by delaying the sent network packets is critical in avoiding cyber defense detection [20, 30, 31]. Sending covert communications, for example, without considering (or mimicking) overt packet time delays, such as employing time delays longer than overt traffic time delays, distinguishes covert traffic from overt traffic. Several researches in the literature, including [20, 22, 30, 31, 40, 45] have utilized time delays to alter the transmission interval of overt network packets in order to transmit covert data. These researches found that encoding covert messages by delaying overt packets' transmission time using time delays was an effective method of evading discovery. As a result, this dissertation follows the literature by developing a covert channel that utilizes time delays to modify the time of network packets, allowing the sequence of timing intervals to be precisely engineered.

Our covert channel transmits secret data between two parties under assumptions [22] as follows:

- The sender and receiver utilize one covert channel and does not use different channels that aggregate and combine packet traffic to leak covert data.
- The sender can send the data to the receiver at any moment, and the channel protocol enables that.
- The original payloads of network packets are normal without any changes, and therefore do not break any security system rules.
- The data transmitted over the covert channel is binary, but how the binary string is interpreted up to the sender and recipient agreement.
- The covert channel is a unidirectional communication paradigm. The receiver cannot interact with the sender through the covert channel, and the sender has no information about the correct reception of covert data bits. But the direct connection in the channel remains bidirectional, and packets are acknowledged.
- Using error-correcting coding could add additional data bits to the data can help

reduce the error caused by transmission error such as delay or lost some packets in the congested network. For implementation and computational simplicity in this chapter, we use error-correcting codes Hamming codes and the Bose, Chaudhuri, and Hocquenghem (BCH) algorithms.

3.1.1 Channel exfiltration experiments

The BCTC sends covert messages bit by bit from sender to receiver and composes the covert symbol set as zeros and ones. The system's event is using packet arrival time, which is modified and controlled by the sender and noticed by the receiver. The sender creates a series of packets separated by inter-arrival times to deliver consecutive bits to the receiver.

BCTC works as follows: initially, the sender and receiver agree on a time delay and a time range for data transmission. Then, the sender obtains the message symbols sequence as input and converts the message to a binary code sequence. After that, the sender uses a set of delaying rules to determine the time delay for the binary symbols. Finally, send the binary symbol s by delaying the overt packets for the time representing the timing value corresponding to the delay threshold.

Assume a covert sender user intends to leak a covert message to a covert receiver through the BCTC. The covert sender first transforms the message to a binary code sequence with two symbols, zero s_0 and one s_1 . The covert sender then selects a delay time value τ_{s_0} based on the delaying rules of the input sequence and produces a transmission event after idling an overt packet for τ_{s_0} time to transmit s_0. This is the same as transmitting s_1; the sender selects a delay time value τ_{s_1} and idling the overt packet for τ_{s_1} time. The following are the specifics of the sending and receiving procedures in this BCTC topic.

The sender sending routine in the BCTC accepts inputs as the sequence covert binary data and time delays selection criteria and creates a series of packets to transmit covert data within traffic inter-arrival times sequences to the receiver side. To do this, the covert sender employs the time delay τ, which is added to the transmission time of each symbol in the binary coding sequence, to modify the transmission time as follows:

$$T'_t = T_t + \tau + \epsilon \tag{3.1}$$

Where T_t is the packet transmitting time and τ is the extra time delays.

In the leaking experiments, the time delays of binary code were chosen based on a form of *Morse code*. A *short time delay* τ_{s_0} was added to the transmission time T_t of overt packets to send a symbol 0, and *a long time delay* τ_{s_1} was added to the transmission time T_t to encode a symbol 1. A delay value of 0 indicated that no extra delay was added, and no covert data was sent out. In other words, the resulting sequence of n-symbol binary code were represented in a sequence of n packet transmission times $\{T'_{si}, T'_{si+1}, \ldots, T'_{sn}\}$ in the differences, such that:

$$T'_{si} = \begin{cases} T_{si} + \tau_0 + \epsilon & \text{if } s_i = 0 \\ T_{si} + \tau_1 + \epsilon & \text{if } s_i = 1 \end{cases}$$

where ϵ is a delay caused by a range of network conditions such as network congestion, which may add additional delays that influence the time of the network traffic.

On the receiving side and after sending the data from the sender, Wireshark [74] is used to monitor network traffic. Upon observing packets, the receiver calculates the packet inter-arrival time ($IAT = T_{ri} - T_{ri-1}$). It then identifies which partition belongs by comparing the inter-arrival time value with the time range of zeros and ones. Finally, based on the choice partition, the receiver records the symbols s_1 or s_2 to get the original covert message as shown in Figure 3.2.

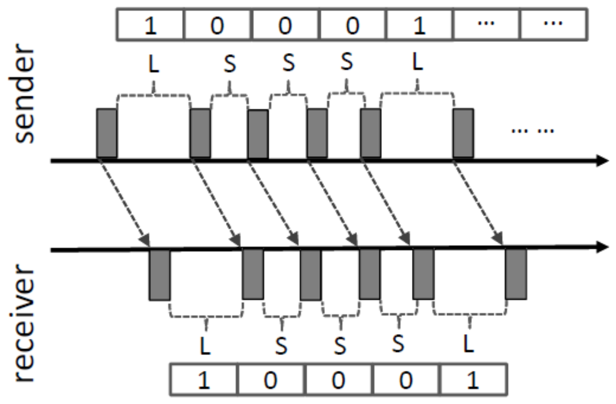


Figure 3.2: An example the covert channel encodes the bit string 10001.

In this channel we assume the sender and receiver should be pre-negotiate and agree on the time delays, time range for partitioning, and encoding method ahead of time. One option to communicate is to hard-code the information into the covert channel framework, so the parties no need to communicate. Another option is to utilize another covert channel such as a covert storage channel to communicate the required information on the fly. In any instance, the receiver receiving procedure just has to know the ruleset of the time range for partition, not the original time message sequence.

In this dissertation, a set of exfiltration experiments were carried out in various network topologies to test the impact of varying time delays on covert channel detection. The time delays of binary encoding that may offer categorizing of short and long traffic inter-arrival times were selected in these tests based on the inter-arrival time behavior of recorded regular traffic transmitted via an overt channel.

To use the behavior of overt traffic, statistical metrics such as t*mean* (μ)*, standard deviation* (σ)*, and* 2 *Sigma* were used to analyze the traffic inter-arrival times of overt traffic within different networks LAN and WAN (details of these networks are provided in Section 3.2). The mean inter-arrival times of overt traffic in these networks were determined in the first phase. For example, the mean inter-arrival equals ≈ 0.0050 s in LAN network 1 and ≈ 0.0025 s in LAN network 2, respectively, and ≈ 0.0664 s in the WAN network.

Then, 2 Sigma control limits were used the (μ) and (σ) to determine the delay time range of the inter-arrival times of overt traffic, which consisted of the lower control limit (L_l), measured as 2 Sigma *below* the μ and the upper control limit (L_u), measured as 2 Sigma *above* the μ, as defined in Equations (3.2) and (3.3). Based on these equations, the time range of inter-arrival times was defined as within the range $[L_l, L_u]$. For example, time range of overt traffic was found to be within two ranges $[0.003109, 0.007274]$ s and $[0.001254, 0.004311]$ s, in the two LAN networks 1 and 2 respectively, and within the $[0.056451, 0.076270]$ s in the WAN network.

$$L_l = \mu - 2 \times \sigma \tag{3.2}$$

$$L_u = \mu + 2 \times \sigma \tag{3.3}$$

As mentioned in our empirical experiments, the covert sender uses the inter-arrival time of overt traffic to determine the time delay used for hiding the covert data. Hence, selection of packet delaying threshold is consider as important factor in the covert channel exfiltration process. Finding this threshold will save the time of future researchers who are studying either the detection or mitigation of covert channels in various applications. For this reason, five cases were examined to determine a threshold time delay. Each case represents a certain value, denoted by λ, that can be used to assign the time delays for covert binary code symbols based on the mean inter-arrival times of overt traffic. In each case *the short time delay is used of a symbol 0* $(\tau_0 = \lambda)$*, and a long time delay is used for a symbol 1* $(\tau_1 = 2 \times \lambda)$, where λ value defined in the equation below:

$$\lambda = c \times \mu \quad where \quad c = \{0.025, 0.50, 1, 2, 3\} \tag{3.4}$$

Where μ is the mean overt traffic inter-arrival times. For example, in the case of applying our exfiltration experiments in LAN network 1 by choosing $\lambda = 0.025 \times \mu$, where the $\mu = 0.0050$ seconds of inter-arrival times of overt traffic in the same network, the time delays are assigned as 0.001250 seconds and 0.002500 seconds for binary covert codes 0 and 1, respectively. Figures 3.3 and 3.4 show the time delays used in our experiments to encode binary codes in both LAN networks and the WAN network based on various values of λ and μ in these networks.

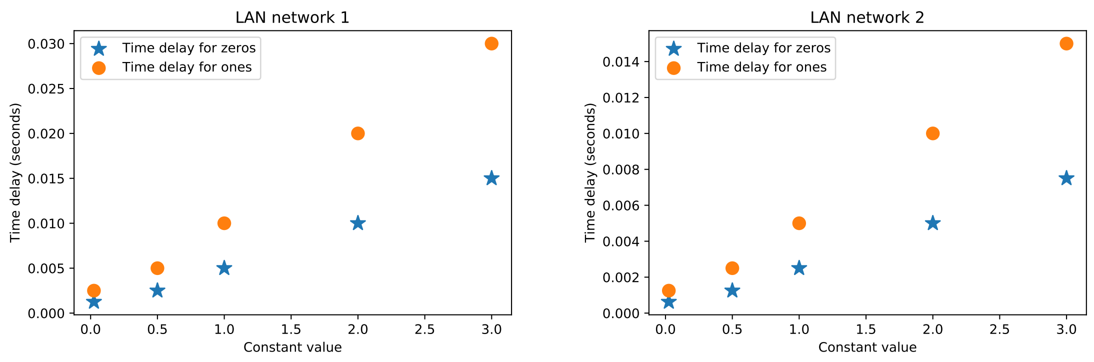


Figure 3.3: Covert binary delay in LAN networks.

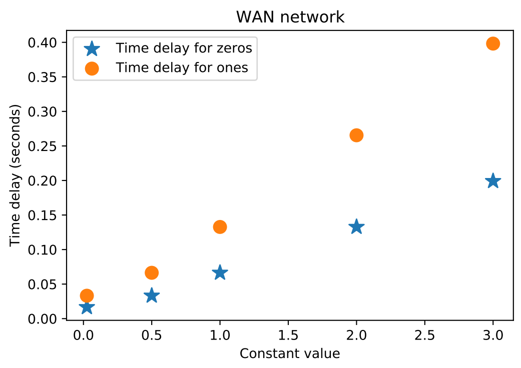


Figure 3.4: Covert binary delay in WAN networks.

After studying the inter-arrival time pattern of overt traffic and establishing the time delays for binary coding, a series of covert data exfiltration tests were carried out in various network settings with varying time delays defined by λ values. For each exfiltration experiment the delay time ranges $[L_l, L_u]$ of binary code (zeros and ones) were defined using Equations (3.2) and (3.3). To investigate the impact of network conditions on packet time, each instance of exfiltration experiments for selecting time delay was repeated N times at various times throughout one day, and the average of the upper and lower control limits of the time ranges for all experiments was determined.

Tables 3.1, 3.2, and 3.3display the time range of covert binary packets in LAN and WAN networks The delays for the binary covert packets are shown in the left-hand columns. The other two columns indicate the time range of these packets' inter-arrival timings, as well as the likelihood that the packets were received within that range based on a particular delay. As an example, consider the first row in Table. 3.1 represents binary packets sent using 0.002500 s and 0.001250 s delays, and 97% of the zero-bit packets were received within a range of $[0.003475, 0.009128]$s, and 96% of one-bit packets were received within a range of $[0.007044, 0.009528]$s. The results are same to the rows in both Tables 3.2 and 3.3.

Table 3.1: Inter-arrival time range in LAN network 1.

$\boldsymbol{\lambda}$	**Binary Code**	**Time Delays (s)**	**Time Range (s)**	**Probabilities**
λ_1	0	0.00125	$[0.003475, 0.007128]$	0.97
	1	0.0025	$[0.007044, 0.009528]$	0.98
λ_2	0	0.0025	$[0.007241, 0.009269]$	0.98
	1	0.005	$[0.007437, 0.012877]$	0.98
λ_3	0	0.005	$[0.007580, 0.012301]$	0.97
	1	0.01	$[0.010052, 0.014773]$	0.98
λ_4	0	0.01	$[0.010077, 0.014489]$	0.98
	1	0.02	$[0.022679, 0.031679]$	0.97
λ_5	0	0.015	$[0.017722, 0.023021]$	0.99
	1	0.03	$[0.032630, 0.036662]$	0.99

Table 3.2: Inter-arrival time range in LAN network 2.

λ	Binary Code	Time Delays (s)	Time Range (s)	Probabilities
λ_1	0	0.000625	$[0.001754, 0.006125]$	0.97
	1	0.00125	$[0.001685, 0.007821]$	0.97
λ_2	0	0.00125	$[0.001795, 0.007915]$	0.96
	1	0.0025	$[0.004315, 0.008203]$	0.98
λ_3	0	0.0025	$[0.004515, 0.008777]$	0.97
	1	0.005	$[0.004995, 0.009020]$	0.98
λ_4	0	0.005	$[0.005985, 0.008864]$	0.98
	1	0.01	$[0.010029, 0.015721]$	0.99
λ_5	0	0.0075	$[0.008437, 0.014877]$	0.99
	1	0.015	$[0.015863, 0.025222]$	0.99

Table 3.3: Inter-arrival time range in WAN network.

λ	Binary code	Time Delays (s)	Time Range (s)	Probabilities
λ_1	0	0.016590	$[0.050876, 0.082136]$	0.97
	1	0.03318	$[0.063427, 0.085147]$	0.97
λ_2	0	0.03318	$[0.060318, 0.082716]$	0.98
	1	0.066361	$[0.078770, 0.099943]$	0.98
λ_3	0	0.066361	$[0.081498, 0.100035]$	0.97
	1	0.132722	$[0.184028, 0.205459]$	0.98
λ_4	0	0.132722	$[0.162647, 0.224923]$	0.98
	1	0.265444	$[0.303688, 0.331161]$	0.98
λ_5	0	0.199083	$[0.233235, 0.269826]$	0.99
	1	0.398166	$[0.421317, 0.496371]$	0.99

3.2 Channel implementation setup

We developed BCTC as sender and receiver (client and server) applications, using Java socket programming and Java software to encode and decode data sent over the channel. The software was designed for and used to send data between two computers (the sender and the receiver) communicating over reliable TCP/IP connections and under normal network conditions in different network configurations: LAN networks and a WAN network. The channel was implemented and tested in two distinct LAN networks (private and public). The two devices were connected to the public LAN network in a research lab in the Computer Science Department at Memorial University of Newfoundland in St. John's, Canada. For the private LAN the identical devices were located in a personal usage network Sin t. John's, Canada. However, in WAN network, the two devices were located in different countries the sender was located Newfoundland in St. John's, Canada, while the receiver was in Virginia, USA. Tables 3.4, 3.5, and 3.6 show the devices' properties that were used in the experiments and the network environments properties that were considered in the experiments as well.

Table 3.4: Devices properties in the LAN networks.

	Sender	**Receiver**
Processor	Intel(R) Core(TM) i7-6500U	Intel(R) Core(TM) i5-4210U
CPU speed	2.50GHz 2.60 GHz	1.70GHz 2.40 GHz
RAM	8.00 GB	6.00 GB
System type	64-bits	64-bits
Adapter type	Ethernet 802.3	Ethernet 802.3

Table 3.5: Devices properties in the WAN network.

	Sender	**Receiver**
Processor	Intel(R) Core(TM) i7-6500U	Intel(R) Core(TM) i7-44790S
CPU speed	2.50GHz 2.60 GHz	3.20GHz 3.20 GHz
RAM	8.00 GB	8.00 GB
System type	64-bits	64-bits
Adapter type	Ethernet 802.3	Ethernet 802.3

In our tests, our packet capturing module uses the Wireshark network analyzer to monitor and record traffic flows between the sender and receiver computers [74].

Table 3.6: LAN networks characteristics.

	Network 1	**Network 2**
Internet speed	52.1 mbps download 15.9 mbps upload	42.8 mbps download 47.1 mbps upload
Latency	55 ms	58 ms
Router type	Home hub 3000	D-link
Number of hops	1	1
Geographical location	Personal use network St John's, NL	Research lab at Memorial University John's, NL

Wireshark captures network traffic and transforms it to a human-readable format. It creates.PCAP files that are used to capture and record network packet data. The PCAP files may then be used to inspect and analyze TCP/IP network packets.

Furthermore, Wireshark can filter network traffic using capture and display filters. Capture filters are used to filter collected traffic, while display filters are used to filter traffic that the receiver agent may see. For example, the receiver agent may filter network protocols or hosts. The receiver agent may begin checking for performance issues after collecting the filtered traffic. Similarly, the receiver agent may filter by source and destination ports to do in-depth analysis on certain network components.

Wireshark captured network traffic on the receiver side of our exfiltration tests using most common five tuples: source IP address, destination IP address, destination port number, source port number, and protocol. These five tuple addresses are used to differentiate across data streams but do not identify covert traffic.

3.3 Covert timing channel analysis

This section examines BCTC's effectiveness in terms of channel accuracy and bit rate, as well as how time delay choices and network condition influence both terms.

3.3.1 Accuracy of distinguishing covert traffic from Legitimate traffic

We assume that the covert channel is devoid of contention noise in order to assess channel effectiveness and give an upper limit on bit accuracy. As a result, the receiver can distinguish between network packets produced by the sender and those intended for other network users. The most significant variables that influence channel accuracy in this situation are clock skew and time delay selection. Clock skew (jitter) in a network is mostly happened based on changing network conditions, which makes increase in packet delay time or reduce latency between sender and receiver. Network conditions have the ability to influence the packet timing transmitting from sender to receiver and reduce channel accuracy [22]. Temporary network congestion, for example, may cause a packet to be delayed longer and alter the covert symbols at the receiver side.

Time delay selection is also essential in deciding how much the time delay threshold impacts channel detection accuracy. Timing intervals linked with various symbols in a sequence should ideally be as distinguish from each other as feasible. For example, the receiver needs to distinguish timing data from the zero time range from those in the one-time range. This is determined by the time delay chosen as well as how specific timing range are determined for each time partitions. The next sections empirically demonstrate the impact of time delay selection and network conditions on channel accuracy.

3.3.1.1 Effect of time delay selection

We initially tested the correctness of our channel by varying the time delays. This point is marked as a threshold, which may be thought of as a barrier separating covert traffic from overt traffic. We investigated five instances of time delays mentioned in Section 3.1.1 to investigate the effect of packet time delays on covert channel detection. We can also identify threshold time delays that assist evaluate the degree of risk of security threats or enhance the quality of channels in transmitting sensitive information by studying these five instances.

The proportion of properly decoding the inter-arrival times for both covert and overt packets based on their time ranges was used to assess the accuracy of differentiating between covert and overt traffic in each instance. The accuracy findings, as shown in Figure 3.5, indicate that accuracy is poor when λ is a quarter of the mean inter-arrival times of overt traffic. Covert traffic is counted in the same way as overt traffic, thus the time range of covert traffic overlaps with that of overt traffic. The sender continues to transmit packets with such inter-arrival times, but they contain no data. When the receiver can observes the changing in inter-arrival times but does not store any binary bits. So the channel with limited rate is more difficult to detect.

However, as time delays increased, such as in the cases of 2 and 3 of λ (which equal half and the same value of the mean inter-arrival times of overt traffic, respectively), the percentage of distinguishing between covert and overt traffic increased as well, allowing for significant differentiation between both between 70-90%. When time delays were included, the accuracy approached 100%, almost twice the mean inter-arrival times of overt traffic. Furthermore, any selected time delay larger than twice the mean inter-arrival times of overt traffic resulted in an accuracy of 100%, showing that covert communication was easily distinguished from overt traffic.

As a result, in terms of receiving covert data clearly to the covert receiver, covert channels with time delays near to or higher than the mean inter-arrival times of overt traffic will be hazardous in malevolent applications or useful in secret applications. Furthermore, we can simulate three kinds of covert attacks with different complexity ranges, from simple to most sophisticated, such as greedy, cautious, and ultra-cautious, depending on the time delay threshold.

3.3.1.2 Effect of network conditions

We also examined the impact of network conditions such as network congestion on the effectiveness of the channel. To do this, we tested our covert channel under normal network as well as a congested network conditions with varying round-trip time (RTT) between communicating parties. Under normal conditions, we observed 100% average bit accuracy with the timing delay set to less double the mean inter-arrival time of overt traffic. But during the network congestion the accuracy rate was

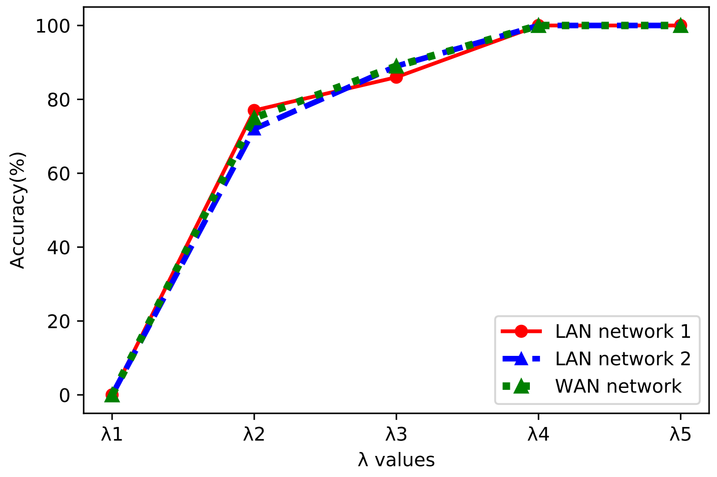


Figure 3.5: Accuracy of distinguishing covert traffic from overt traffic.

decreased. To maintain accuracy during times of congestion, the time delays should be increased.

Therefor, clock skew, has a significant impact on covert timing channel efficiency. For example, may lead the receiver to record packets coming beyond the intended time interval. As a consequence, this it may cause single bit flips. This flipping problem can be solved using error correction coding. Each network packet has a one connection with a one bit of covert data in the covert timing channel, making timing covert channel independent bit error of each other and not affected to the following bit. Because of this advantage, the bit error correction methods is better suited for the timing channel.

One method for dealing with with network condition errors is utilizing error correction codeing to fix single bit mistakes. Error correction coding technique provide extra bits to the message that may be used to identify and rectify certain transmission problems [75]. ECCs have the disadvantage of increasing the amount of bits to be sent. We used Hamming codes and the Bose, Chaudhuri, and Hocquenghem (BCH) encoding method in this research because they are computationally easy to encode and decode and can correct one mistake per codeword.

To test the effect of network conditions, we ran our covert channel between sender and receiver over the WAN network to transmit the binary encoding using one value of λ specified in Section 3.1.1 and obtain the channel accuracy. In Figure 3.6, we show the bit accuracy results in case of using and error correction and without it for all 100 covert inter-arrival time sequences. We found that 92 of 100 sequences yielded more than 90% bit accuracy with no error correction, whereas eight sequences produced less accuracy due to particular sequence characteristics overlapping with typical traffic. With error correction, 98 sequences achieved bit accuracy rates of more than 90%. Error correction resulted in a 6% improvement in bit accuracy on average.

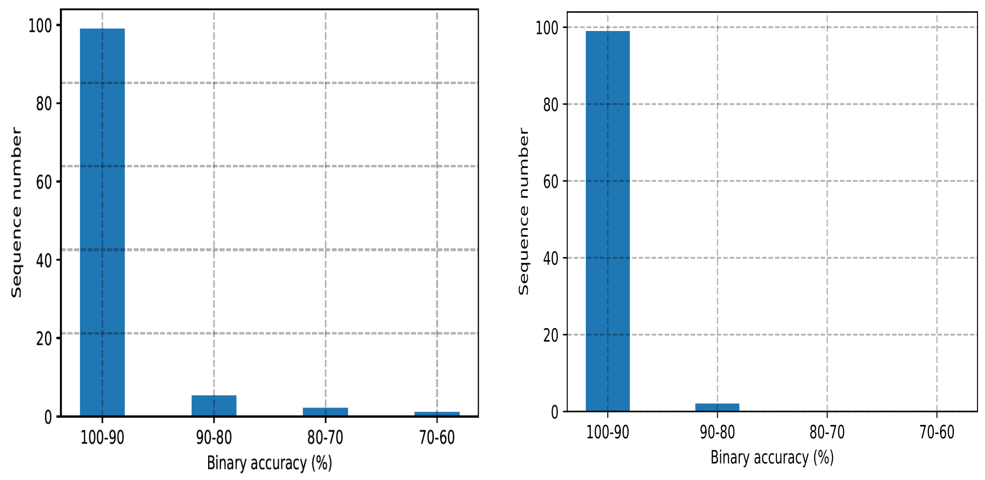


Figure 3.6: Bit accuracy results with no error correction (left) and with error correction (right).

Another method for preventing transmission mistakes is to use no transmission interval between parities. There is no packet transmission between sender and recipient during the no transmission interval period. As a result, the channel can recover the errors that happened before to this time. The length of this interval has already been agreed upon by the parties. The no transmission interval may can be set as a default time interval, or the covert channel can be used to transmit it before the data transmission process begins. However, the sender has no way of knowing the hidden bits were properly received by the recipient. As a result, the sender has the responsibility to monitor the changing in network conditions and choose when to stop transmission.

A rapid shift in the round-trip time between sender and receiver, for example, may be a strong indication that the sender should adopt the no transmission mode. On the receiving side, the receiver does nothing until the start of a bit packet comes. The enter of the silent mode improves channel accuracy while decreasing transmission rate.

Rather than slowing transmission by creating intervals with no transfer, the channel may progressively adjust to changes as network conditions change. In the interval adjusting method, the receiver carefully monitors the arrival time of each packet and compares it to the idealistic scenario such as the expected next packet arrival time based on the current time interval. When the two times are compared, a certain value is calculated, which is the difference between the ideal and real timings and may be indicate as positive or negative value depending on if the packet came late. This value is subsequently added to the interval time, and the clock time is adjusted for the next incoming packet. This method is most helpful when there is an changing in network conditions. It may, however, create problems if the change in network latency exceeds 50% of the interval time, which modifies the time interval incorrect way.

3.3.2 Transmission bit rate

The covert timing channel employed in this research is not symmetric, which means that the transmission delays and bit rates for the binary covert packets vary. In other words, transmitting 1024-bits of zeros takes less time than transmitting 1024-bits of ones. This implies that the communicated bit rate of zeros is greater than the transmitted bit rate of ones.

The height data rate is obtained by utilizing sequences with the shortest inter-arrival times. This rate, however, is limited by 1) effect of clock skew on channel accuracy. With short delay times, a little change in the packets interval times may change the symbol generation such as located it in a different partition. For long intervals, however, the same variation may be insignificant; 2) sequences with high data rates may be abnormal in contrast to the network's typical sequences.

The fact that zeros and ones are intermingled in the communication experiments is significant, since the findings would have been different if the proportion of zeros and ones in the hidden information had been different. To investigate the effect of intermixed zeros and ones inside covert information on transmitted bit rate, several instances were investigated, each with a different proportion of zeros and ones in the hidden information. One of the cases utilized was 50% zeros and 50% ones, indicating that a binary encoding stream with a length of 1024-bits was communicated between the sender and the receiver, with 512 bits of zeros and 512 bits of ones.

This was the case for all other cases, and each situation was evaluated using three metrics: L_l, mean, and L_u for inter-arrival time ranges of binary covert packets, as shown in Figure 4.5. The findings indicated that when short delays were employed in both network setups, the greatest number of bits sent via the covert timing channel was recorded, as shown in Figure 4.5 a. The transmission bit rates dropped as the difference between delays grew, as illustrated in Figure 4.5 b,c in both network topologies.

Furthermore, the number of bits sent based on the L_u value in the range of the inter-arrival times was low in comparison to the number of bits sent based on the L_l value for any transmission delays in both networks. Because of 1 s in configuration, 1 is deemed inadequate time to transmit a significant number of bits with high delays, and the quantity of bits sent is determined by the network's transmission speed. The following method was used to calculate the bit transmission rate values displayed in Figure 4.5:

$$T_{d(4-bits)} = N_0 \times V_0 + N_1 \times V_1 \tag{3.5}$$

where $T_{d(4-bits)}$ is the time duration needed to send a binary data stream with length is 4 bits; N_0 is the number of zeros in the 4-bits stream; V_0 is one of the three values of L_l, mean, or L_u for bit 0, N_1 is the number of ones in the 4-bits stream; and V_1 V_0 is one of the three values of L_l, mean, or L_u for bit 1. In Equation (3.5), a data stream with a size is 4 bits was chosen for simplicity to calculate the transmission bit-rate based on the percentage of zeros and ones in the covert information. For example, in the scenario that introduces 75% zeros and 25% ones at delay $\tau_0 = 0.00125$ s and delay $\tau_1 = 0.0025$ s in network configuration 1, where the minimum values (L_l) in the range of inter-arrival times for zeros and ones are 0.003475 and 0.007044, respectively, the 4-bits stream contained 3 bits of zeros and 1 bit of ones, and the time duration to send these 4 bits was equal to $(3 \times 0.003475) + (1 \times 0.007044) = 0.017469$ s. Then, the number of streams that needed to transmit the 4 intermixed binary encoded symbols based on the time duration of the bits determined in Equation (3.5) was defined in the following:

$$N_{T_d} = \left\lfloor \frac{T_u}{T_{d(4-bits)}} \right\rfloor \tag{3.6}$$

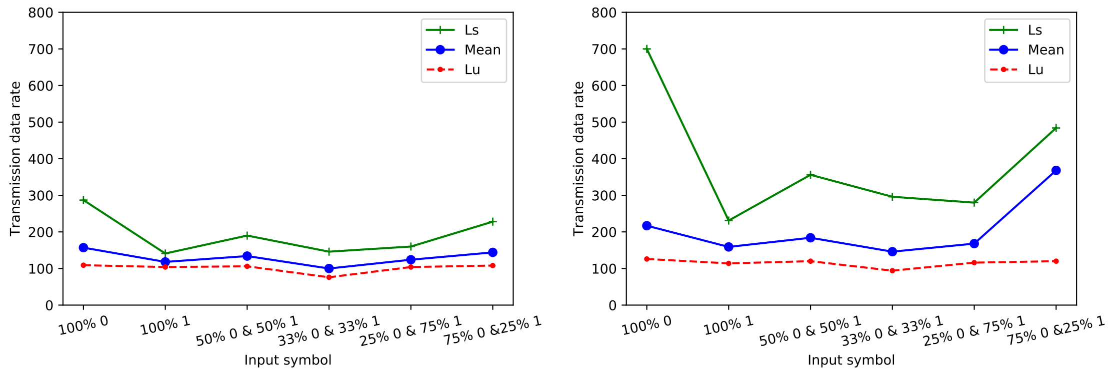
where T_u is the time unit utilized to transmit the hidden information between the sender and recipient. The time unit in this research is 1 second. The number of binary bits transferred per second in all 4-bit streams was then determined using the formulae below:

$$TR_0 = N_0 \times N_{(T_d)} \tag{3.7}$$

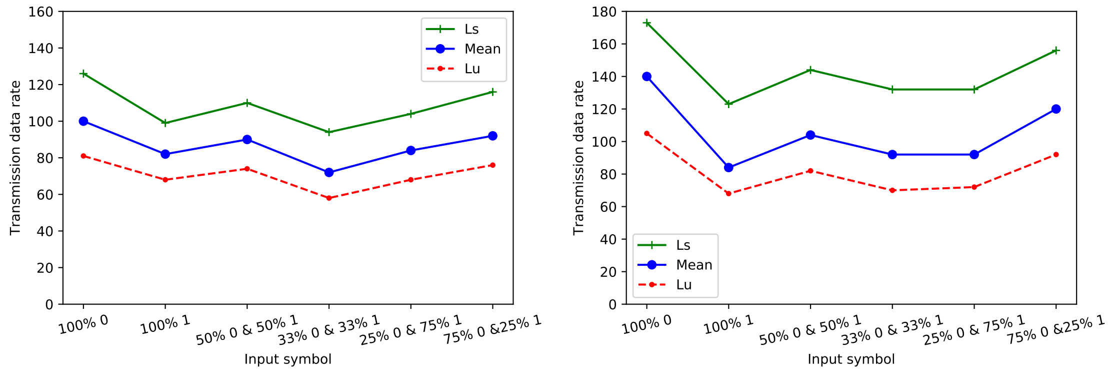
$$TR_1 = N_1 \times N_{(T_d)} \tag{3.8}$$

3.3.3 Experimental environment and evaluation metrics

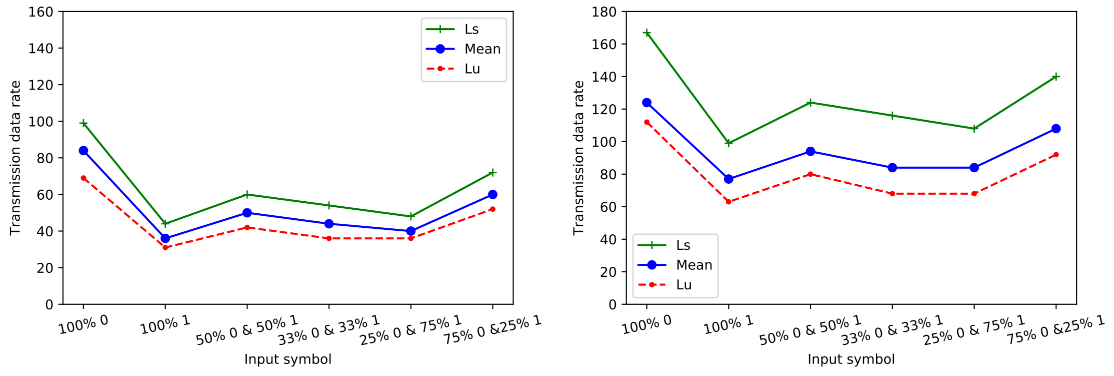
The following hardware and software were utilized in the experimental setting for this dissertation to identify CTC using the suggested detection methods described in the following chapters. The testing environment used a $x86-64$-ThinkStation-P920 server running Linux version $55 - 18.04.1-$Ubuntu; the primary processor unit had



($\mathbf{a}$) Transmission bit rate based on λ_1



($\mathbf{b}$) Transmission bit rate based on λ_3



($\mathbf{c}$) Transmission bit rate based on λ_4

Figure 3.7: Transmitted bit rates of LAN network 1 (left) and LAN network 2 (right).

24 cores and 62.42GB RAM. In addition, several tests are carried out on Google Colab and a personal computer of Intel Core i7-6500U@ 2.50 GHz, 28 GB RAM, and

Windows 10 operating system.

The software is written in Python, a programming language that is extensively used in machine learning and deep learning methods. To make matrix operations easier, we utilized the Numpy and pandas packages. In addition, the TensorFlow and Keras libraries were utilized to perform deep learning on the GPU. The researcher may simply construct the network model using the Keras library, which contains functions such as data preparation, parameter tweaking, and deep learning layers in block form.

The studies took into account various performance metrics, such as recall, which is concerned with the completeness of detection (i.e., the fraction of the total CTCs that were detected). Precision, on the other hand, assesses the quality (or exactness) of identified CTCs. Our assessment seeks to offer a thorough study of different classifiers and accuracy metrics, allowing users to choose the classifier with the most relevant accuracy parameters (stakeholders). As a result, we use the most widely used accuracy measures in network security areas. Following that, we list these metrics and explain each one.

- Accuracy (A): calculates the number of correctly classified instances of both classes (overt and covert) over the total number of instances using the following equation:

$$A = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3.9}$$

- Precision (P): calculates the number of correctly detected class members by the classifier over the total number of correctly and incorrectly detected members using the following equation:

$$P = \frac{TP}{TP + FP} \tag{3.10}$$

- Recall (R): also called Detection Rate (DR) or True Positive Rate (TPR), which calculates the number of class members that correctly detected over the total number of class member samples using the following equation:

$$R = \frac{TP}{TP + FN} \tag{3.11}$$

- F_1 score: provides a score that balances both of precision and recall in one measurement as follows:

$$F_1 = \frac{2 \times (P \times R)}{(P + R)} \tag{3.12}$$

- False Alarm Rate (FAR): also called as false positive rate, shows the percentage of the number of records incorrectly classify using the following equation:

$$FAR = \frac{FP}{FP + TN} \tag{3.13}$$

where True Positive (TP) is the number of segments (images) correctly classified as CTCs. True Negative (TN) is the number of correctly classified segments as non-CTC (overt). False Positive (FP) is the number of segments that are incorrectly classified as CTC. False Negative (FN) is the number of segments incorrectly classified as non-CTC channels while they are CTCs.

3.4 Chapter summary

This chapter demonstrated how to create a covert timing channel by altering overt traffic time and injecting traffic containing covert information into the network traffic flow. The inter-arrival times of overt and covert traffic were analyzed in two different network configurations to investigate the behavior for both configurations and observe how network conditions affected the bite rate transmission of the covert timing channels accuracy of distinguishing covert traffic from overt traffic.

According to our findings, covert traffic does not often show extreme values when the threshold of packet delays employed to conceal the covert data is less than or equal to a fourth of the mean of overt traffic inter-arrival times. As a result, more covert traffic is counted near overt activity, causing the time range of covert traffic to overlap with the time range of overt traffic and making the difference between them difficult. However, when the threshold of packet delays employed is roughly equal to or more than twice the mean inter-arrival times of overt traffic, there is no overlap between the time ranges of covert traffic and the time ranges of overt traffic, making the distinction between them simpler. Based on these findings, it is beneficial to identify certain levels that may assist in distinguishing between covert and overt traffic. This threshold may also be used to create an undetected covert channel that can be utilized for a variety of reasons. This chapter avoids optimizing for any specific channel or networked application in favor of finding characteristics that provide acceptable performance while being extremely resilient under varying conditions.

Chapter 4

CTC Detection and Localization Based on Image Processing Using Machine Learning Techniques

Recently, regarding to the increasing of using covert timing channels, these channels has become a serious threat to network security. As a result, detecting these channels is a important of contemporary information technology infrastructure. Numerous businesses, nations, and government organizations are concentrating their efforts on developing more effective methods for detecting hidden routes. This will act as a critical building element for a decision support system that guards against such vulnerabilities in the IT infrastructure.

Numerous network traffic monitoring and detection techniques have been developed to detect covert timing channel attacks. However, many recent studies have highlighted the need for quick and trustworthy technologies for successfully thwarting such fraudulent traffic. With a fast and reliable tool, it may become possible to immediately identify such channels and take necessary action to suppress and terminate the assaults before they have a chance to send more data across the network.

Thus, this dissertation examines the potential of detecting covert timing channels using an image processing method, which is different from current detection strategies that rely on the static characteristics of traffic to identify such channels. By proposing a novel visualization model, the traffic time is converted to colored images and the covert channel detection issue will convert to an image classification challenge. This approach offers an efficient and rapid methodology for detecting changes in covert attack patterns and pointing that in the image.

The main motivations behind using a visualization model are as follows: I) visualizes packet inter-arrival time data as images, making various of covert assaults clearly visible to the human eye; II) utilizes image processing methods such as color or edge feature extraction algorithms to identify the characteristics of time series that appear as distinct color line patterns (color blocks) on temporal-spatial planes; III) clusters the lines obtained to aid in classifying and labeling the numerous anomalies detected; IV) utilizes image compression techniques. V) shows the efficacy of visual characteristics in detecting and classifying CTC using machine learning classifiers.

This chapter first describes the architecture of the novel proposed SnapCatch CTC detection model, which uses image processing with machine learning algorithms. Then it presents the processes involved in identifying and localizing such channels in traffic flows. Finally, it discusses using various image-based datasets to evaluate the proposed model's efficacy and quantitative performance.

4.1 SnapCatch: Image-based CTC detection model

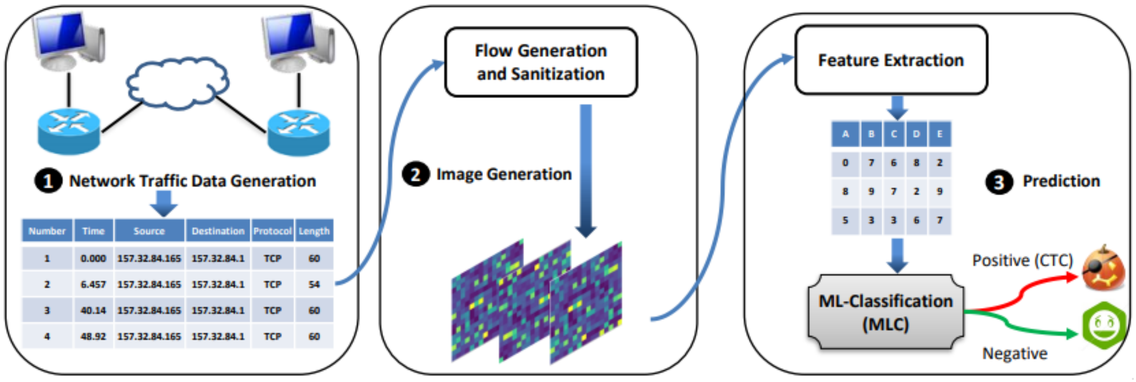
This section provides the suggested CTC detection methodology, as well as measurements taken on real-world traffic traces from two large networks. The suggested model may be divided into three major processes:

1. **Generating traffic datasets using our proposed covert channel (BCTC)**.
2. **Converting packet inter-arrival times into colored-image representations**.
3. **Constructing feature image extraction and machine learning models for CTC detection.**

The main components of our method are shown in Figure 4.1. First, we design and build a software environment that includes two systems (sender and receiver) that communicate over different network topologies (LAN and WAN). Section 3.2 describes the specifics of various system and network topologies. Furthermore, we create a malicious agent that injects (encodes) covert messages into the sender's traffic flows. This agent injects hidden data using three distinct defensive evasion methods, with settings ranging from greedy data exfiltration to cautious and stealthy data exfiltration. These three configurations are found by applying the time delays described in Section 3.3.1.1.

Our packet capturing module then uses Wireshark [74] to monitor and record traffic flows between the source and recipient systems. The recorded traffic flows include both overt (benign) and covert (malicious) traffic. Following that, our method extracts the inter-arrival times of the packets from these flows and stores them in the flow dataset.

The image processing module reads each traffic flow's inter-arrival timings and transforms them into colorful pictures. Then, from the colored pictures, our feature extraction program extracts eight chosen characteristics. Finally, our classification module takes the feature sets as input and builds machine learning classifiers to identify pictures with covert traffic flows. Following that, we will go through the specifics of each module.



Number	Time	Source	Destination	Protocol	Length
1	0.000	157.32.84.165	157.32.84.1	TCP	60
2	6.457	157.32.84.165	157.32.84.1	TCP	54
3	40.14	157.32.84.165	157.32.84.1	TCP	60
4	48.92	157.32.84.165	157.32.84.1	TCP	60

A	B	C	D	E
0	7	6	8	2
8	9	7	2	9
5	3	3	6	7

Figure 4.1: Workflow of CTC detection using image processing and machine learning.

4.2 Raw data generation

In this phase, we create datasets using our suggested channel (BCTC) from Chapter 3. For data creation, the malicious agent has two primary parameters that influence its ability to avoid detection: 1) determining the time delays of its packets; and 2) determining the size of its hidden messages. The covert agent employs packet time delays depending on the time delay threshold specified in Section 3.3.1.1, where this threshold is roughly equal to or more than twice the mean inter-arrival time of overt traffic in the first batch of time delays. As a result, in this section, we developed three distinct time delay configurations to simulate three different kinds of CTC assaults of increasing complexity: (1) greedy, (2) cautious, and (3) ultra-cautious CTC attacks. The assessment section will go through the specifics of each assault.

The size of the covert message is the second parameter for mimicking overt traffic. As previously stated, hostile agents have the ability to transmit stolen data in big or tiny covert communications. Large covert communications create a rapid shift in network activity and are therefore simpler to detect than tiny ones. To investigate the impact of various covert message sizes on the accuracy of CTC detection, we provide the malicious agent (the sender) the option to transmit covert messages in three different sizes: 8, 64, and 128 bits.

Our packet capturing module uses Wireshark [74] to record the communication flows between the covert sender and covert receiver systems in order to create and aggregate the traffic datasets. The collected collection of internet flows is then categorized into overt and covert traffic. Finally, our method extracts the inter-arrival times of these traffic flows and stores them in flow datasets.

4.3 Creating colorful pictures from traffic inter-arrival times

The preceding phase generates datasets comprising the inter-arrival times of the traffic flow recorded during communication between the sender and receiver systems. Our method converts these inter-arrival periods into colorful pictures depending on their values in this phase. This allows us to use common image processing methods to extract more robust picture-based characteristics for subsequent processing. To do this, each sub-flow of inter-arrival times was first put in a 2D 16x16 matrix. Each matrix is filled in row by row and left to right with the inter-arrival timings. The inter-arrival time in each matrix are then normalized to a value between 0 and 255 to create a color picture. Finally, each matrix is rendered as a colored picture by converting each of its normalized (16x16) components to a color pixel. The matplotlib package [76] was used for this purpose, which can generate a picture from a 2D matrix. As illustrated in Figure 4.2, the picture will contain one square for each element of the 2D matrix, and the color of each square is determined by the value of the associated matrix element.

We created two kinds of datasets for our experiments in this chapter. The first kind is for channel detection, and it includes 4,608,000 inter-arrival times for overt and covert traffic based on the three CTC assaults and three covert message sizes. The second kind is for channel localization, which includes $1,534,464$ inter-arrival

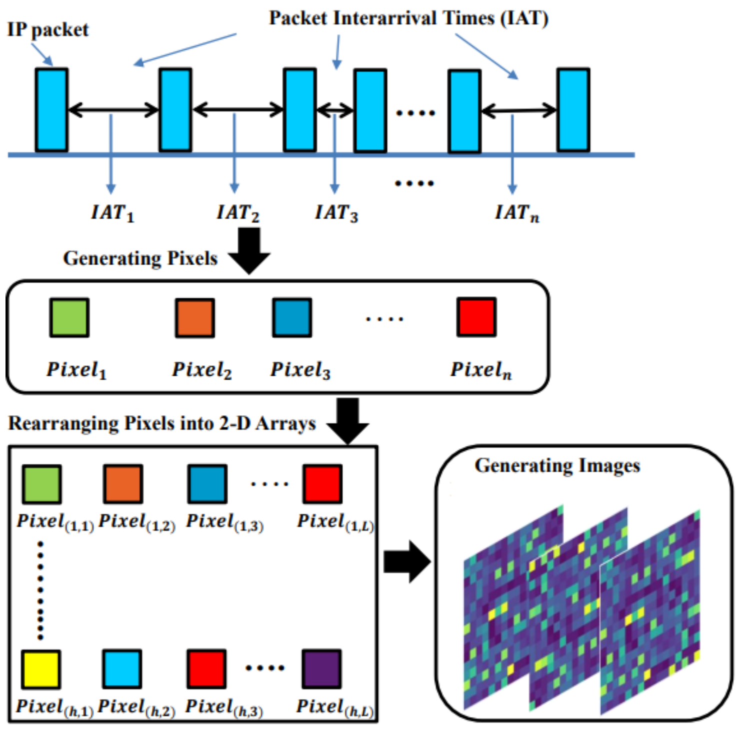


Figure 4.2: Illustration of inter-arrival time images.

time for covert traffic based on three sizes of covert messages injected in three places in the traffic flow: beginning, middle, and finish. Tables illustrate the specifics of each dataset. 4.1 and 4.2, respectively.

Table 4.1: Detection dataset parameters

Covert message size	8, 64, 128 bits
Delay time	2μ, 0.5μ, 0.25μ
The mean IATs of overt traffic (μ)	0.0664 seconds
Sub-flow size	256 inter-arrivals
Number of dataset versions	9
Number of images (all datasets)	18000
Number of covert images (all datasets)	9000
Number of overt images (all datasets)	9000
Number of images per dataset version	2000
Number of covert images per dataset version	1000
Number of overt images per dataset version	1000

Table 4.2: Localization dataset parameters

Covert message size	8, 64, 128 bits
Delay time	2μ
The mean IATs of overt traffic (μ)	0.0664 seconds
Sub-flow size	256 inter-arrivals
Number of dataset versions	3
Number of images (all datasets)	5994
Number of images per dataset version	1998
Location of covert message	Beginning, middle,end

4.4 Feature extraction and image classification

Once the colored pictures are produced, our method extracts eight common characteristics from them. These characteristics are then utilized to train and build a machine learning classifier for detecting covert communications. In this part, we will first explain these characteristics as well as the techniques utilized for extraction. The machine learning methods used to build and verify the CTC detection classifier are then described.

4.4.1 Feature extraction

As previously stated, each colorful picture produced by the preceding phase indicates a traffic sub-flow. Using these pictures as input, our feature extraction program retrieves eight common characteristics from each image. We list and describe each of these characteristics below:

1. **Mean gray value**, which computes the average gray value inside the picture by adding the gray color values of all pixels in the image and dividing the total number of pixels by the number of pixels

2. **Standard deviation**,which computes the standard deviation of the gray values that were used to produce the mean gray value

3. **Mode**,This indicates the image's most commonly occurring gray value Corresponds to the histogram's greatest peak value.

4. **Center of mass**, which computes the brightness-weighted average of all pixels' x and y coordinates in the image These are the first order spatial moments' coordinates.

5. **The integrated density**, which computes the area of one pixel multiplied by the sum of the image's pixel values

6. **Median**,which computes the median value of the image's pixels

7. **Skewness**,which computes the third-order moment about the mean where the distribution's area lies, to the left or right of the mean.

8. **Kurtosis**, a distribution metric used to determine if a distribution is peaked or flat around the mean

We used the ImageJ tool [77] to extract these characteristics from the pictures The collected characteristics were then saved in a CSV file for use in the classifier training procedure. This procedure will be explained further below.

4.4.2 Machine learning model construction

to glean these features from the images

The attributes gathered were then stored in a CSV file for use in the classifier training process.

This method will be discussed in more detail below.

- Support Vector Machine (SVM).
- Decision Tree (DT).
- Naïve Bayes (NB).
- Artificial Neural Networks (ANN).

Using the characteristics collected from each picture, we trained each classification model. The models were then tested using the hold-out validation technique. We divided our dataset into 70% for training and 30% for testing using the hold-out validation technique.

We used four machine learning methods to explore which one performs better in this domain: Support Vector Machine (SVM), Decision Trees (DT), Nave Bayes (NB), and Artificial Neural Networks (ANN) (ANN). The SVM classifier operates on the dataset by creating the optimum hyperplane that splits the best observations (features) into two portions that represent the two classes (overt and covert). Using the DT classifier, on the other hand, has the benefit of allowing a human to evaluate the resultant decision tree in terms of which characteristics are the most correct (i.e., have the lowest Gini impurity value). Furthermore, the most important (determining) characteristics are always utilized at the top of the tree, while the unnecessary features are disregarded (pruned). Because of their capacity to identify underlying connections in data, NB and ANN are highly popular machine learning algorithms that continue to obtain high accuracy metrics in pattern identification.

We utilize the popular data mining program Waika Environment for Knowledge Analysis (WEKA 3.8) [78] to build and verify these models. Following that, we discuss and share our experiments and assessment findings.

4.5 Experimental results and analysis

Our evaluation seeks to quantify the performance of our approach in the following ways: (a) the effectiveness of our approach in detecting CTCs under different defense evasion configurations of cyber-attacks; (b) the ability of our approach to pinpoint the covert part (set of packets) of the traffic sub-flow; and (c) compare and contrast different machine learning classifiers in detecting CTCs based on their accuracy and interpret-ability.

4.5.1 Experimental results of using three types of CTC attacks

To validate the accuracy and efficiency of our proposed approach, we present our experimental results and compare them with three popular baseline CTC detection approaches. These detection approaches are: (1) the regularity test [22], (2) entropy test, and (3) corrected conditional entropy [9, 23].

Furthermore, in order to evaluate and compare the efficacy of our method in identifying different kinds of covert channels, we examine several covert channel configurations ranging from basic to stealthy CTCs that use sophisticated defensive evasion tactics.

As a result, we built the tests using three kinds of CTC assaults based on real-world scenarios: (1) Greedy Covert Timing Channels (GCTC): When transmitting packets, this kind of channel ignores the usual time-delays of overt traffic. As a result, it is considered as the simplest kind of assault to identify. (2) Cautious Covert Timing Channels (CCTC) try to mimic the delay periods of overt communications, making this kind of covert channel more difficult to detect. (3) Ultra-Cautious Covert Timing Channels (UCCTC) are the most sophisticated and difficult to detect kind of covert channel. This kind of channel prevents the malicious sender from deviating from overt traffic at the cost of time and the quality of the stolen information. Following that, we offer additional information about these assaults and provide the outcomes of our approach's assessment for each attack.

4.5.1.1 Greedy Covert Timing Channels (GCTC)

Our preliminary assessment findings demonstrate the effectiveness of our method in identifying Greedy Covert Timing Channels (GCTC). GCTC is the most basic kind of covert channel since it does not attempt to overlap with (or mimic) the time-delays of overt traffic. As a result, when compared to overt traffic, it often creates abnormalities in traffic. Because of this observable irregularity, this kind of channel is simpler to identify.

the packet time-delay setting was adjusted to match the twofold mean inter-arrival time of overt traffic to mimic the GCTC attack. In addition, to verify our approach's efficacy in identifying various sizes of covert messages for this attack, we conducted tests with three distinct covert message sizes: 8, 64, and 128 bits. In each experiment, these covert messages were inserted into a 256-bit traffic flow sent from the sender to the recipient.

As previously stated, we captured the produced traffic using Wireshark and then used our image creation method to transform the traffic flows into colorful pictures. Then, we retrieved the eight characteristics from these pictures and used them to train four different classifiers (SVM, DT, NB, ANN) using 70% of the data and evaluated the classifiers with the remaining 30% of the data for both network configurations.

The detection results for the GCTC attack on the LAN network utilizing three covert message sizes: 8, 64, and 128 bits are shown in 4.3. As demonstrated in the Figure, our method (SnapCatch) obtained the greatest CTC detection accuracy of the GCTC attack, with accuracies of 99.83% , 100% , and 100% for covert message sizes of 8, 64, and 128 bits, respectively, utilizing the DT and NB models. The CCE obtained the second-highest CTC detection using an ANN classifier, with 82.83%, 85.67%, and 90.83% for covert message widths of 8, 64, and 128 bits, respectively. The entropy-based method finished third with accuracy rates of 72.67 %, 76.83 %, and 82.55 %. The regularity-based method, on the other hand, came in fourth (and last) with an accuracy of 71.0 %, 75.50 %, and 79.17 % for covert message sizes of 8, 64, and 128 bits, respectively.

Moreover, Figure 4.4 shows the detection results for GCTC attack inter-arrival times recorded in the WAN network. As shown by the Figure, the highest CTC detection accuracy of the GCTC attack was achieved by the SVM classifier trained by our approach, which achieved 99.83%, 100%, and 100% for the covert message sizes of 8, 64, and 128 bits, respectively. Moreover, Table 4.3 represents more accuracy measures (e.g, F_1 score). The CCE achieved the second-highest CTC detection, which reached 82%, 85.5%, and 89.83% for the covert message sizes of 8, 64, and 128 bits, respectively. The entropy-based approach achieved third place with the accuracy of 72%, 77.33%, and 81.33%. In contrast, the regularity-based approach achieved fourth place with the accuracy of 70.2%, 74.83%, and 78.83% for the covert message sizes of

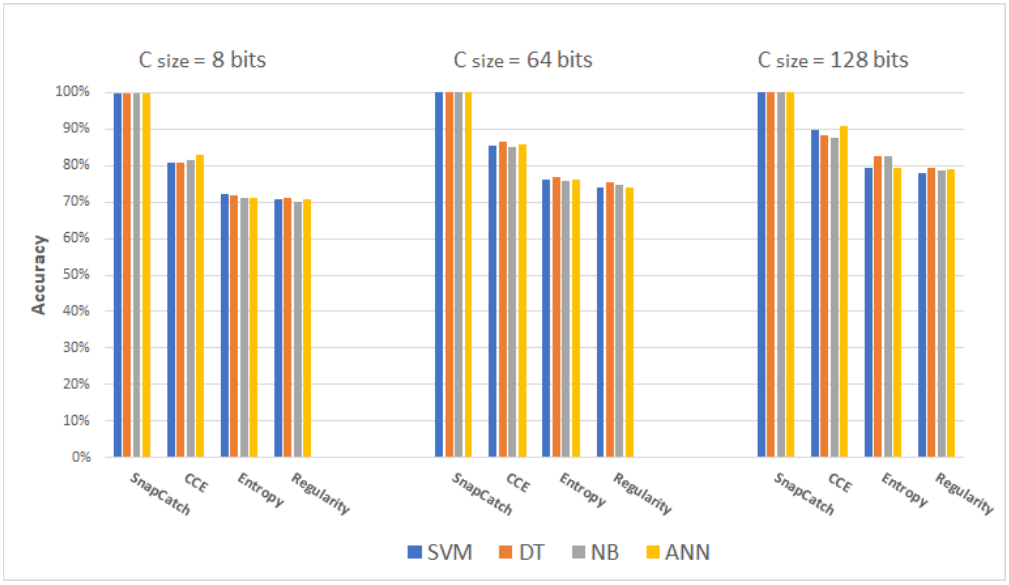


Figure 4.3: The impact of the covert message size on the detection accuracy of GCTC in LAN network.

8, 64, and 128 bits, respectively.

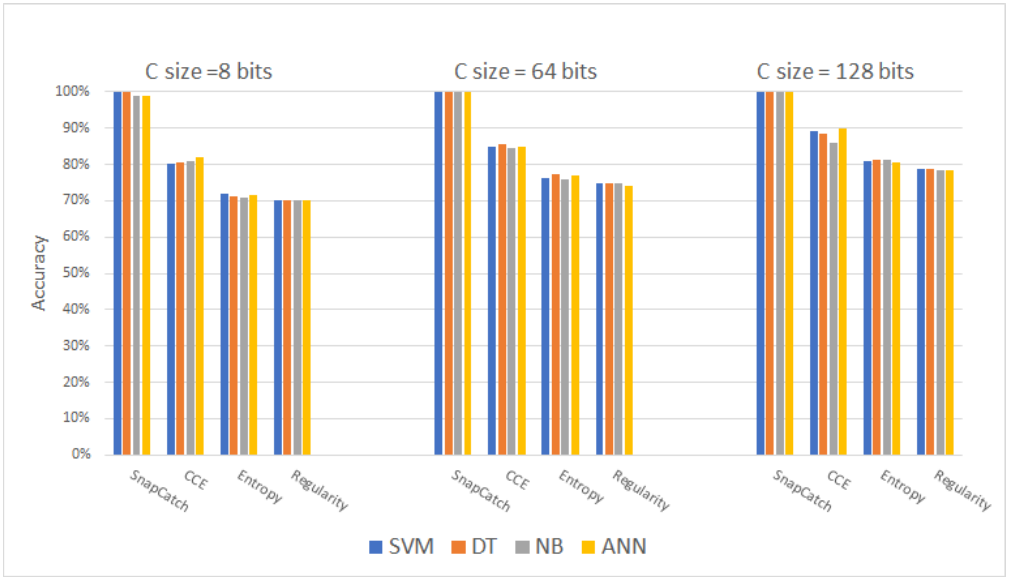


Figure 4.4: The impact of the covert message size on the detection accuracy of GCTC in WAN network.

4.5.1.2 Cautious Covert Timing Channel (CCTC)

Our second set of evaluation results shows our approach's performance in detecting Cautious Covert Timing Channels (CCTC). CCTC is a more advanced cyberattack that seeks to imitate the overt traffic behavior (packet inter-arrival time) to a certain degree. This type of covert channel exhibits near-overt traffic behavior using packet inter-arrival times that are partially similar to overt traffic. Because of its similarity to overt traffic, CCTC is a more challenging type of covert channel to detect.

To simulate the CCTC attack, *the packet delay was set to be equal to half of the mean inter-arrival time of overt traffic*. When combined with the delay enforced by the network, this delay exhibits less abnormality in the traffic, which makes it harder to detect by the security detection measures. We ran the experiments for the CCTC attack using the three covert message sizes: 8, 64, and 128 bits. Figure 4.5 shows the detection results for the CCTC attack in the LAN network using the different covert message sizes. The Figure shows that our approach outperforms the three other methods: CCE, entropy, and regularity. SnapCatch achieved 74%, 92.33%, and 95.33% for the covert message sizes of 8, 64, and 128 bits, respectively. As expected, the accuracy of the detection methods is mostly affected by the cover message size. This is because smaller covert messages (e.g., 8 bits) cause less change to otherwise normal traffic behavior. We found that the detection accuracy of SnapCatch gradually increases when the size of covert messages increases.

The CCE achieved the second-highest CTC detection using the ANN classification, which reached 63.83%, 77.83%, and 82.67% for the covert message sizes of 8, 64, and 128 bits, respectively. The entropy-based approach achieved third place with the accuracy of 60.83%, 71.50%, and 76.83%, where the regularity-based approach achieved fourth (and last) place with the accuracy of the approach 58%, 65.50%, and 71.5%for the covert message sizes of 8, 64, and 128 bits respectively.

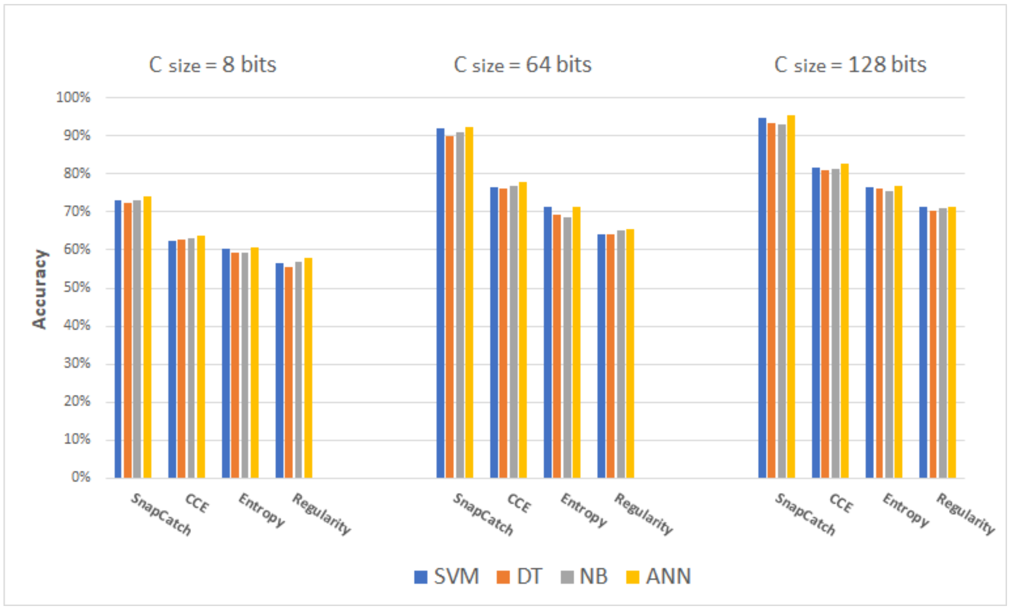


Figure 4.5: The impact of the covert message size on the detection accuracy of CCTC in LAN network.

Similarly, Figure 4.6 depicts the detection of CCTC attacks in WAN network architecture. The SnapCatch technique outperforms the other three approaches, as shown in the Figure. SnapCatch obtained 74.33 %, 91.36%, and 95.83%, respectively, for covert message sizes of 8, 64, and 128 bits. Table 4.4 represents additional accuracy measures such as F_1 score and precision.

CCE obtained the second greatest detection accuracy, with detection accuracies of 63%, 76.83%, and 81.66% for covert message sizes of 8, 64, and 128 bits, respectively.

The third set was obtained by the entropy-based method, which had accuracies of 60.33%, 71.1%, and 77.33%. The regularity-based method, on the other hand, came in fourth place, with detection accuracies of 56.83%, 64.83%, and 72.33%for covert message sizes of 8, 64, and 128 bits, respectively.

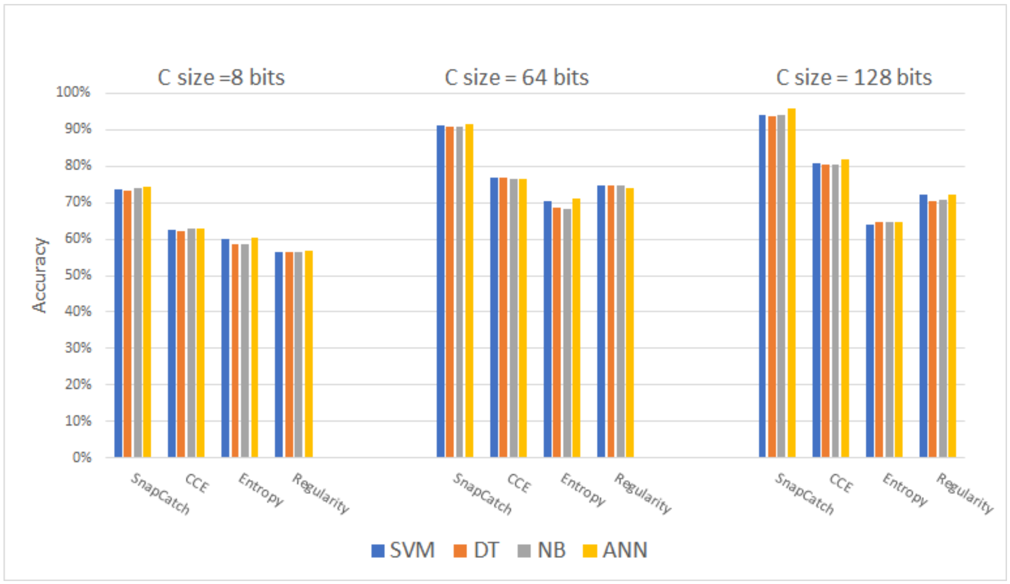


Figure 4.6: The impact of the covert message size on the detection accuracy of CCTC in WAN network.

4.5.1.3 Ultra-Cautious Covert Timing Channels (UCCTC)

Our final set of assessment findings demonstrates the effectiveness of our method for identifying Ultra-Cautious Covert Timing Channels (UCCTC). UCCTC is a cutting-edge cyber-attack that prevents covert channels from displaying characteristics (packet inter-arrival times) that differ from overt traffic. This limitation often compromises the quality of the stolen information by forcing packets to occur within a shorter time period. However, because of the great resemblance to overt traffic behavior produced by non-malicious apps, this kind of cover channel is the most difficult to identify.

the packet delay was set to be equal to the quarter of the mean inter-arrival time of overt traffic to mimic the UCCTC assault. We carried out the UCCTC attack tests with three different covert message sizes: 8, 64, and 128 bits. As demonstrated in Figure 4.7, our method obtained the greatest detection accuracy for the SCTC attack, with detection accuracies of 61.67%, 74.83%t, and 76.83% for covert messages of length 8, 64, and 128 bits. The CCE had the second-highest CTC detection rate, with 57.7%, 63%, and 66%. The entropy method (third place) had accuracy values of 52.83%, 60.33%, and 64.83%, respectively, while the regularity approach (fourth place) had accuracy values of 52.17%, 57.5%, and 63%, respectively.

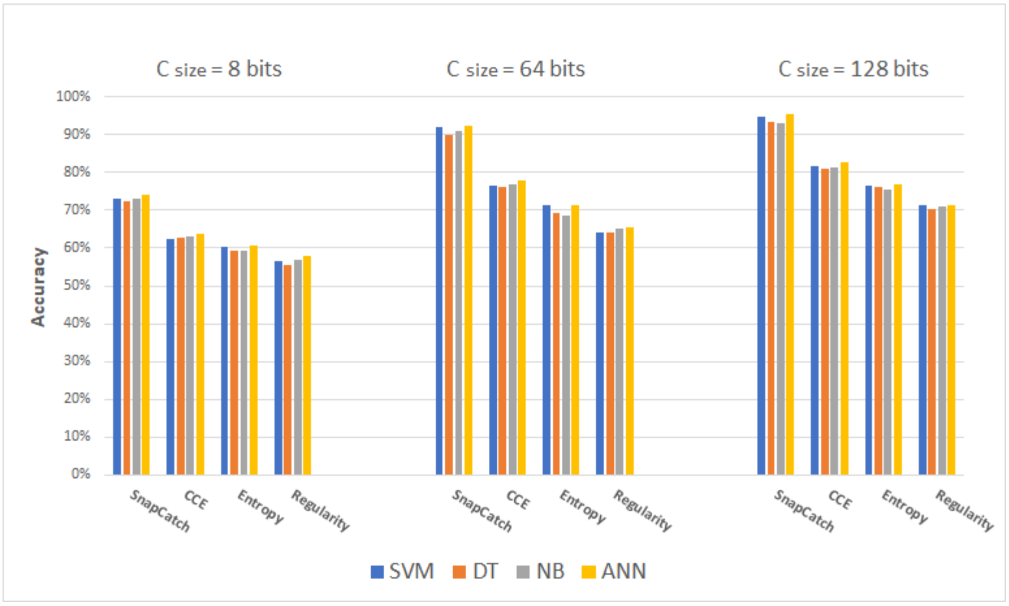


Figure 4.7: The impact of the covert message size on the detection accuracy of UCCTC in LAN networks.

Similarly, the results of UCCTC attack detection in WAN network setup are shown in Figure 4.8 and Table 4.5. The Figure demonstrates that SnapCatch obtained the greatest detection accuracy for the UCCTC attack, with detection accuracies of 61.33 %, 74.5 %, and 76.83 % for covert messages of length 8, 64, and 128 bits. The CCE had the second-highest CTC detection rate, with 58 %, 63 %, and 66.5 %. The entropy method (third place) had accuracy values of 52.83 %, 60.33 %, and 64.83 %, respectively, while the regularity approach (fourth place) had accuracy values of 52.17 %, 57.5 %, and 63 %, respectively.

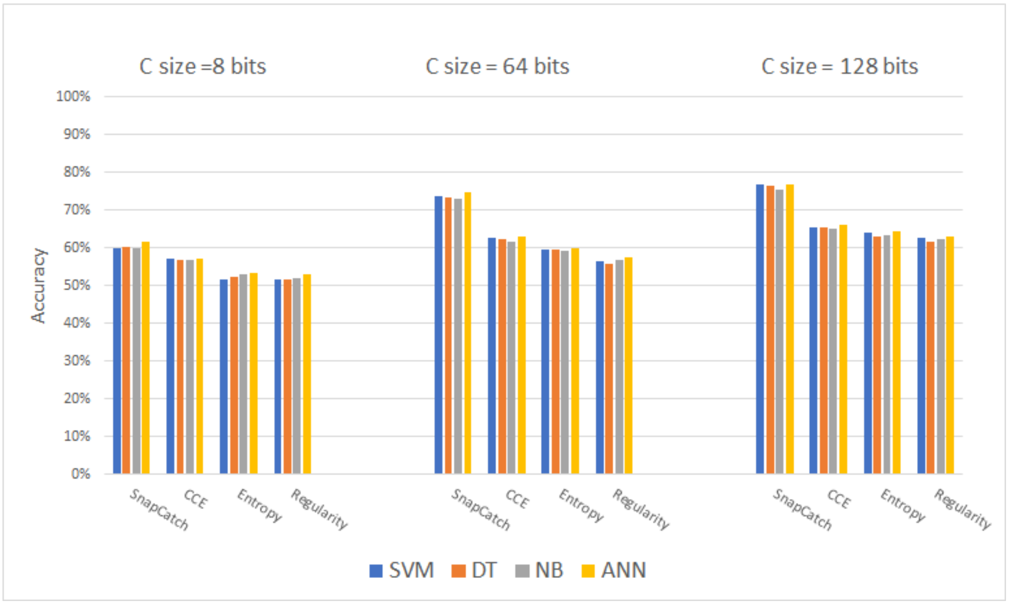


Figure 4.8: The impact of the covert message size on the detection accuracy of UCCTC in WAN network.

Table 4.3: Results of Precision, Recall, and F_1 score of the image-based method for GCTC in WAN network

		Covert traffic				Overt traffic			
Size	Measure	SVM	DT	NV	ANN	SVM	DT	NV	ANN
8 bits	Precision	1.000	1.000	0.997	1.000	1.000	0.997	1.000	0.984
	Recall	1.000	0.997	1.000	0.980	1.000	1.000	0.997	1.000
	F_1 score	1.000	0.998	0.998	0.995	0.9987	0.998	0.998	0.994
64 bits	Precision	0.999	1.000	0.998	1.000	1.000	0.999	1.000	1.000
	Recall	1.000	0.999	1.000	1.000	0.998	0.999	1.000	0.998
	F_1 score	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
128 bits	Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Recall	0.999	1.000	1.000	1.000	1.000	0.999	1.000	1.000
	F_1 score	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 4.4: Results of Precision, Recall, and F_1 score of image-based method for CCTC in WAN network

		Covert traffic				Overt traffic			
Size	Measure	SVM	DT	NB	ANN	SVM	DT	NB	ANN
8 bits	Precision	0.703	0.699	0.820	0.963	0.784	0.782	0.776	0.668
	Recall	0.820	0.820	0.333	0.513	0.653	0.647	0.670	0.980
	F_1 score	0.757	0.755	0.474	0.670	0.713	0.708	0.719	0.795
64 bits	Precision	0.902	0.894	0.896	0.905	0.922	0.921	0.922	0.922
	Recall	0.923	0.923	0.925	0.926	0.901	0.891	0.894	0.904
	F_1 score	0.913	0.908	0.910	0.914	0.911	0.906	0.908	0.913
128 bits	Precision	0.946	0.937	0.937	0.972	0.934	0.937	0.94	0.924
	Recall	00.938	0.936	0.940	0.920	0.947	0.938	0.937	0.973
	F_1 score	0.950	0.935	0.938	0.945	0.940	0.939	0.938	0.948

4.5.2 Detecting the presence of hidden communications in traffic flows

As previously stated, identifying hidden communications is critical in preventing malicious programs from stealing sensitive data from a network. When covert communications are successfully detected, the traffic flow is terminated in order to interrupt

Table 4.5: Results of Precision, Recall, and F_1 score of image-based method for UCCTC in WAN network

		Covert traffic				Overt traffic			
Size	Measure	SVM	DT	NB	ANN	SVM	DT	NB	ANN
8 bits	Precision	0.608	0.550	0.553	0.582	0.983	0.983	0.983	0.685
	Recall	0.560	0.997	0.997	0.807	0.397	0.393	0.193	0.420
	F_1 score	0.585	0.711	0.711	0.676	0.328	0.323 0	.324	0.521
64 bits	Precision	0.653	0.650	0.652	0.738	0.997	0.995	0.996	0.753
	Recall	0.997	0.998	0.996	0.760	0.465	0.458	0.462	0.730
	F_1 score	0.790	0.780	0.789	0.749	0.635	0.628	0.632	0.741
128 bits	Precision	0.797	0.730	0.905	0.987	0.745	0.819	0.922	0.680
	Recall	0.720	0.370	0.923	0.530	0.817	0.840	0.904	0.954
	F_1 score	0.757	0.491	0.914	0.693	0.779	0.831	0.913	0.810

the malicious data exfiltration process. Although deleting traffic flows carrying covert messages is an efficient cyber defense against covert channels, it is also highly disruptive to the QoS of legitimate applications whose overt traffic may comprise the majority of the packets in the discarded flows. Discovering the segments of traffic flows (i.e., sets of packets) carrying the hidden message(s) is a critical goal. It may drop just the harmful portion of traffic flows while allowing the rest to continue through. This accurate identification substantially minimizes overt traffic interruptions caused by non-malicious apps.

This section analyzes and shows the performance of our method in locating hidden messages inside traffic flows (i.e., the sequence of packets). To accomplish this, we devised experiments in which covert messages are inserted into every network flow at one of three points (segments): *beginning, middle, and end.* Then, we evaluated the performance of our method in locating the right traffic flow segment (beginning, middle, or end) containing the hidden message (s).

First, we trained the classifiers with three additional labels: (1) start, (2) middle, and (3) end. Each of these designations corresponds to the position of the hidden message in a particular traffic flow. These labels are created automatically by the malicious agent's settings. For instance, if the malicious agent is configured to inject the covert message at the start of traffic, it will also assign the label tt starting to that traffic flow. It also does the same with the other two labels, tt middle and tt end. After that, the labeled set of covert traffic flow is utilized for training and testing. We trained the classifiers using 70% of the labeled data and tested them with the remaining 30%. The accuracy of identifying the malicious traffic flow segment (beginning, middle, and end) that includes the covert message is shown in Figure 4.9.

Our approach's SVM classifier effectively identified the segments inside traffic flows that included the covert message with accuracies of 97.83%, 100%, and 100% for covert message sizes of 8, 64, and 128 bits, as shown in Figure 4.9. We also performed the CCE, entropy, and regularity to give a baseline for comparison. Interestingly, regularity came in second place with detection accuracy of 40.83%, 49%, and 53.83% for covert message sizes of 8, 64, and 128 bits, respectively. CCE came in third place, with accuracies of 40.17%, 45.55%, and 48.17%. Entropy came in lowest, with accuracies of 33.33% for all sizes of hidden communications.

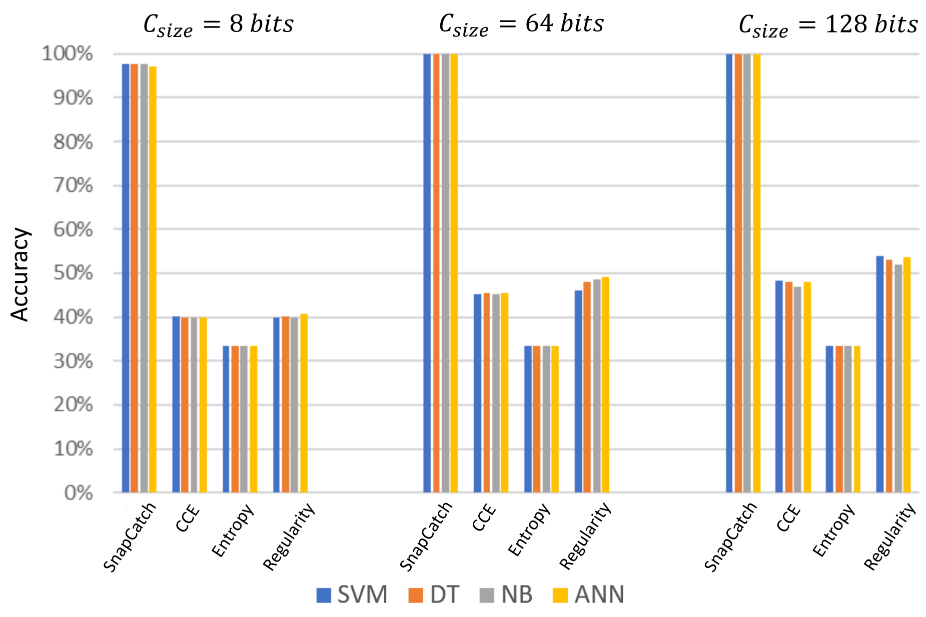


Figure 4.9: The accuracy of pinpointing the location of covert messages within a traffic flow using different message sizes.

4.6 Discussion

According to our findings, SnapCatch takes a significant step toward accurate and automatic covert channel identification. This is important not only for identifying covert channels, but also for precisely finding covert messages inside traffic flows, allowing us to disrupt these messages without causing substantial QoS degradation, as shown by our measurement research. With the fast rise in cyber assaults that install and use covert channels to exfiltrate sensitive information, the suggested method identifies CTCs rapidly. The suggested method, on the other hand, is still general and has to be fine-tuned to conform to the organization's security purpose and limitations. In this part, we will go through the tweaking that is needed to attain the *best* results using SnapCatch.

Selecting the *best* machine learning classifier. SnapCatch's assessment reveals that it achieves excellent accuracy and coverage. However, our method still produces false positives (overt traffic that is incorrectly identified) and false negatives (malicious traffic that is incorrectly missed/allowed). We tested with several machine learning classifiers and measured their accuracy and recall, as shown in Table 4.6. Depending on the stakeholder defense strategy, several variables may be considered while choosing the most suitable classifier. Choosing the ANN classifier, for example, yields the highest accuracy score (98.7%t) for identifying a UCCTC attack with a covert message size of 128 bits. This classifier is biased toward correctly identifying only covert channels, at the expense of missing about 47% of the covert channels (false negatives). Choosing the NV classifier, on the other hand, results in the highest recall value (92.3%) for the identical assault. This classifier emphasizes identifying the most covert channels at the expense of erroneously detecting 9.5% of overt traffic (false positives).

Table 4.6: Measures comparison of SnapCatch classifiers for UCCTC attack (covert message size = 128 bits).

classifier	Precision	Recall
ANN	98.7%	53%
NB	90.5%	92.3%
SVM	79.7%	72%
DT	73%	37%

Balancing QoS and security. According to our findings, SnapCatch achieves better accuracies not just in identifying CTCs, but also in segmenting traffic flows and locating the right traffic segment containing hidden communications. SnapCatch beats the second-best method, regularity, by about 57This implies that SnapCatch identifies a traffic flow segment containing the hidden message, allowing for the exact dumping of that traffic segment rather than the whole flow. This implies that the loss of QoS caused by deleting whole traffic flows containing hidden messages may be minimized by up to 66 % when adopting our method. As a result, we think SnapCatch is the best candidate tool to combine usability and security, with an average accuracy of 99.2% in identifying the right traffic flow segment out of three potential segments carrying hidden message(s).

4.7 Chapter summary

The SnapCatch model is presented in this chapter as a new method for automating and accurately detecting CTCs. SnapCatch is a stealth traffic detection system that specializes in image processing and machine learning methods. First, the system transforms traffic inter-arrival times into colored pictures using a unique method that collects actual network traffic characteristics and displays them in colored images. SnapCatch trains different machine learning classifiers to identify covert channels quickly by extracting robust and accurate features from colored pictures, based on a configurable defensive strategy that prioritizes (or balances) correctness and completeness.

Furthermore, we offer a method for locating covert messages (i.e., a collection of packets) inside a communication flow, allowing us to remove just the segment of the traffic flow containing the covert message rather than the whole flow. SnapCatch surpasses the corrected conditional entropy, entropy, and regularity methods in our assessment. Furthermore, our approach has the lowest performance loss while identifying tiny (e.g., 8 bit) covert communications and ultra-cautious covert channels (UCCTC), the most sophisticated kind of covert cyber assault. SnapCatch considerably outperforms baseline methods in identifying segments inside traffic flows carrying covert messages, resulting in a considerable reduction in service quality caused by deleting covert traffic flows. Finally, we provide a variety of scenarios and use cases for customizing SnapCatch to execute a defensive strategy that is appropriate for the tool's users' resources and security goals.

Chapter 5

CTC Detection and Localization Using Convolution Neural Network

The increasing using of the internet has opened up a new channel for covert channel hackers, advising businesses and organizations to stay ahead of the detecting threat environment. As a consequence, the SnapCatch detection model is being considered as a possible defensive measure for network security. Based on SnapCatch's outstanding accuracy in practice, as shown in Chapter 4, we hypothesized that image recognition and classification techniques may help enhance the performance and accuracy of covert timing detection and localization.

The SnapCatch detection method, on the other hand, is restricted by the lengthy time required to extract image features using a large-scale dataset. Furthermore, the quality of retrieved features has a substantial impact on the overall performance of classifiers that meet certain requirements or assumptions. As a result, the challenge is devising a CTC detection model to use as a technique for quickly and automatically extracting features. It may be able to quickly detect such channels and take appropriate action, such as covert mitigating defensive measures to interrupt the concealed communication flow, by automatically extracting image characteristics.

Deep learning, as opposed to machine learning, removes the requirement for human feature extraction. For example, we might instantly input images to a deep learning system, which would identify and forecast the item. The deep learning model surpasses the traditional machine learning method in this respect. A convolutional neural network (CNN) is one of the most popular deep learning applications in computer vision. CNN can automatically extract image characteristics, avoiding the drawbacks associated with manually derived unsuitable features.

In comparison to other fully connected neural network approaches, CNN depends significantly on local relation and weight sharing techniques to reduce the number of network parameters and computation needed during training the model. Furthermore, CNN utilizes the original image as input, with the convolution layer extracting various image features. As a result, no new features are required. Each convolution layer produces a set of features that correspond to a higher level feature extracted by a certain filter. For these reasons, CNN is being utilized to combine traffic visualization with new covert timing channel identification and localization techniques.

The CNN model is a strong competitor to improve the SnapCatch model's performance in terms of training and testing model speed and accuracy by automating image comparison that visually identifies certain hidden patterns inside an image. The covert timing channel detection research flow will be expanded to a deep learning technique as a consequence of the proposed CNN model. Deep learning abstracts data at the highest level possible using a complex architecture or a mixture of non-linear transformations.

The use of shallow convolutional neural network architecture for image classification in various applications has lately drawn the attention of the scientific community. To reduce model parameters for classification tasks, Lee et al. [79] proposed a shallow CNN with a logarithmic filter. Similarly, Li et al. [80] created a shallow CNN model to recognize images with a limited dataset size. Their shallow CNN model achieved results comparable to traditional machine learning. Lei et al. [81] recently proposed a shallow CNN with batch normalize technique to improve image classification accuracy. Lei et al. model's is composed of four layers. The model results show that shallow CNN may get excellent classification results when compared to another deep CNN models with lower parameters. As a consequence, since we have a limited dataset size, we motivated this study by developing a CTC detection and classification model using the same shallow CNN model with batch normalization technique. We will utilize various variables, including as input image sizes and different boosted parameters, to explore the viability of shallow CNN techniques for CTC detection and localization.

This chapter describes the design and implementation of a covert timing channel detection model based on CNN with batch normalization. It learns the hierarchical feature representation automatically, then determines the optimum model architecture that provides high-performance outcomes in channel identification by evaluating model hyper-parameter effects and determining optimal parameter values for the proposed model. Furthermore, it detects and localizes hidden channels in traffic flows, and lastly, it employs a variety of image-based datasets to evaluate the model's effectiveness and quantitative performance.

5.1 Image classification and CTC detection using a CNN architecture

Deep CNN and shallow CNN structures were utilized to extract image characteristics and categorize photos in the relevant research that employed the CNN method to identify network threats. The CNN method employs weight sharing and pooling techniques to substantially decrease computation and parameters, allowing the training depth model to be implemented. Many deep convolutional neural networks are being created in response to the fast growth in computer capacity. Deep CNNs have shown satisfying results in computer vision tasks, as demonstrated by AlexNet [82] and VGGNet [83]. These deep algorithms, on the other hand, contain complicated network topologies and many parameters such as kernel size and number of layers. VGG16 [83], for example, has 16 layers and over million parameters, requiring a significant amount of time and memory.

Many layers require training time. As a result, it does not always make sense to utilize deep networks with millions of parameters, particularly in basic problems with less complicated data patterns. According to the literature, tackling difficult pattern recognition issues seems to be feasible only with the use of specialized hardware or optimization techniques such as GPU computations. Such options do not satisfy the network traffic dynamics, which is dynamic, quickly changing, and diverse, due to the long training time of deep networks. However, the use of shallow CNN is under-utilized, and research on such networks has lately gained interest. Shallow networks may be more advantageous for some devices with restricted access. Furthermore, it is critical to have a quick, light technique for working with large data and making decisions in a timely manner.

For image classification and CTC detection, this dissertation presents a new shallow CNN with batch normalization. The shallow CNN network has fewer layers and smaller convolution kernels. The fundamental concept behind our proposed approach is that the quantity of CTC data is very restricted, and that small data sets make deep networks difficult to converge, thus increasing the danger of over-fitting. As a result, a shallow CNN may be effectively trained with few data. Shallow networks would be more suited for analyzing our small and constrained datasets. Furthermore, by developing the shallow detection model, we may explore if certain well-known CTC image classification issues can be addressed using just shallow CNN, which can be trained in a short amount of time and without the need of special hardware or complex computations.

To speed the convergence of the proposed shallow CNN model and enhance the ability of generalization for classification accuracy, the batch normalization technique is introduced after each convolutional layer of the network. The presence of an internal covariate shift phenomena in training conventional deep neural networks [84] is the primary reason for adopting batch normalization. The data distribution in each of intermediate layer would be different through the updating the model training parameter than before the parameters were changed. Because of this behavior, the network is continuously forced to adapt to changing data distributions, making training very challenging. The batch normalization technique [85] is introduced to cope with parameters that vary during CNN training [86] and enhances network convergence and generalization capabilities.

The parts that follow first show the CNN model design before delving into the model's characteristics and evaluating its performance.

5.1.1 CNN based CTC detection

This section describes our proposed detection method and the procedures required to identify CTC using image processing and the CNN methodology. The proposed approach is split into two main stages: **1) transforming packet inter-arrival times into colored image representations; and 2) extracting features from colored images and classifying them using CNN.**

The two main components of our method are shown in 5.1. First, the packet inter-arrival times were converted to colored image representations using the same image processing module described in Section 4.3. The image processing module converts the inter-arrival timings of each traffic flow into a 2D matrix that is then interpreted into a colorful image. As stated in Section 5.2.1, we created a variety of image-based datasets in this chapter.

The second stage in our proposed method is to categorize the colored images and identify CTC using a shallow CNN. When the input is multidimensional, a CNN offers even more benefits. The colorful images may be thought of as the network's input in our job. In contrast to conventional recognition algorithms, the CNN technique is not hindered by difficult feature extraction and reconstruction. CNN is a multilayer perceptron that is intended for recognizing 2D forms and is resistant to image distortion (e.g., scale, translation, and rotation).

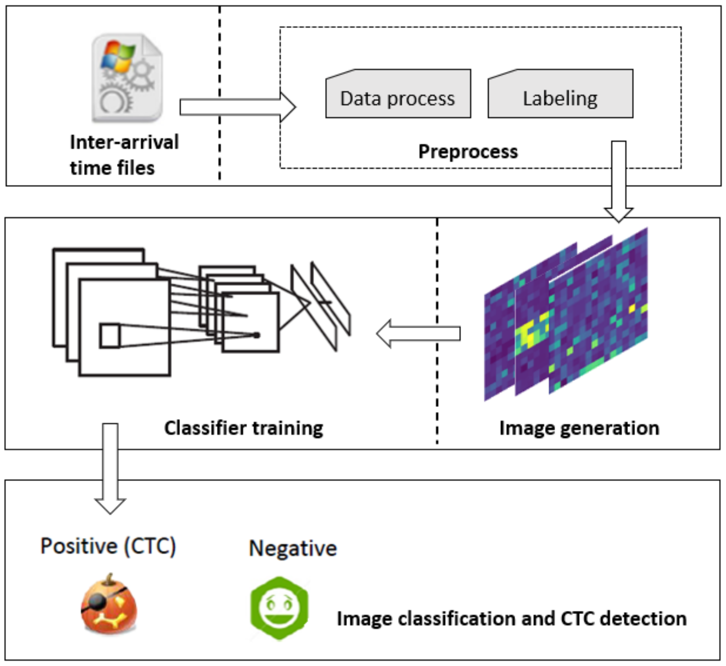


Figure 5.1: CNN framework.

The CNN model is composed of many components, as explained in Algorithm 1 and shown in Figure 5.2. The first layer is the input layer, which is responsible for bringing the training image data into the neural network. Following that are the convolution and subsampling layers. The first layers enhance data properties and decrease noise. The latter may decrease data processing while preserving and storing valuable information. Then, a fully connected layer converts two-dimensional features into one-dimensional features to meet the classifier requirements. Finally, the classifier detects and categorizes the images with two labels covert and overt based on their properties. The specifics of each layer, as well as the iterative formula for each layer, are as follows.

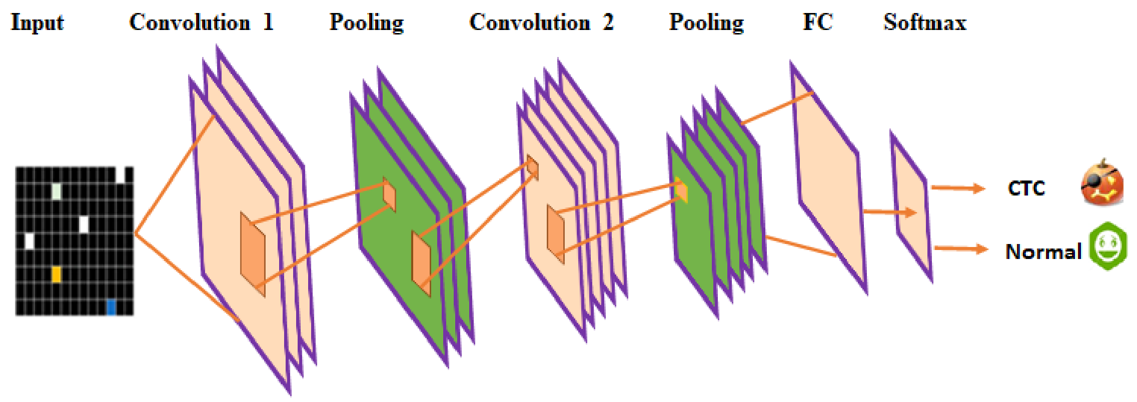


Figure 5.2: Architecture structure for CNN model.

Algorithm 1: CNN Based CTC Detection Algorithm

```
Input: Training data and parameters
Output: Class label
Create classification model
  Generate the first convolutional layer, followed by batch normalization
  technique
  Add max pooling layer
  Generate the second convolutional layer, followed by batch normalization
  technique
  Add max pooling layer
; Add a dense layer
Model training
for $I_{th}$ epoch iterate
    for $J_{th}$ mini-batch iterate
        Extract feature representation using Eq. 5.1
        Minimize the cost function
        Update model parameters using Adam optimizer algorithm
    end for
end for
    Output the result according softmax classifier
Model testing
Test the model using the test dataset
```

The input layer is the first layer, and it is responsible for bringing the training images into the neural network. The model then extracts data features using 32 filters with 3 $\times$ 3 size. Batch normalization is used in CNN to normalize each feature map produced by convolution layers. To minimize the likelihood of gradient vanishing, the activation function's input value considers within the sensitive input. The convolutional layer is followed by BN and ReLU to enhance the model's accuracy and nonlinear expression capabilities. Then, to decrease data size and computational complexity, max pool with size 2x2 is employed.

To extract deep features, the next convolutions are used as (3 $\times$ 3) 64 filters. Following the second convolutional layer, BN technique are employed to enhance the accuracy and nonlinearity of the CNN model. The technique used by the second 2 $\times$ 2 max-pooling layers lowers dimensional of data and computational complexity. The feature is then converted into a 1-D feature that meets the classifier requirements via the fully connected layer. ReLU is employed with the fully connected layer to improve the CNN model's nonlinear capabilities. Dropout is then used to minimize overfitting and enhance the model's generalization capabilities. Finally, the softmax classifier classify images into two categories covert and overt based on their probability. Softmax needs minimal calculations and memory, which saves precious time resources. The full iterative formula for each layer is provided below.

1. *Convolution layers*: is one of the most essential layers of CNN because it may decrease the amount of image parameters while maintaining the key characteristics, known as invariance, such as translation, scale, and rotation invariance. This method successfully avoids overfitting and improves the model's generalization capabilities. Various convolution kernels extract various data characteristics. The proposed CNN model has convolutional layer with 32 and 64 filters

of size 3 × 3, as shown by the following equation:

$$x_j^l = f\left(\sum_{i \in M_j} x_j^{l-1} * k_{ij}^l + b_j^l\right) \tag{5.1}$$

where f is the activation function, M_j is the set of input mappings, k_{ij} is the convolution kernel, and b_j^l. The error cost function partial derivative to respect bias and convolution kernel as given below:

$$\frac{\partial E}{\partial b_j} = \sum_{x,y} (\delta_j^l)_{x,y} \tag{5.2}$$

$$\frac{\partial E}{\partial k_{ij}^l} = \sum_{x,y} (\delta_j^l)_{x,y} (p_i^{l-1})_{x,y} \tag{5.3}$$

where p_i is the patch for each convolution layer and k_{ij} is value can be obtained of position in the input feature map.

2. *Pooling layer:* In general, its convolution kernels are the patch's maximum or mean and are not affected by backward propagation. It improve training time speed by decreasing the size of the feature map, preventing overfitting, and minimizing data redundancy. The proposed CNN model employs two 2 x 2 max-pooling layers to decrease data dimension and computational complexity while maintaining the extraction's important characteristics unchanged.

3. *Fully connected layers:* Each neuron node of this layer is connected to each neuron in the upper layer. The CNN model use a fully connected layer with 512 neurons.

4. *Softmax classifier:* to accomplish binary or multi categorization, the softmax output layer is frequently employed with convolutional neural networks, which is defined as follows:

$$softmax(y)_i = \frac{e^{y_i}}{\sum_{i=1}^{n} e^{y_i}} \tag{5.4}$$

where n is the number nodes in the output layer, which corresponds to the number of classes in the particular classification problem, and y_i denotes the output.

5. *Batch Normalization (BN):* BN strategy is utilized in neural network improve the classification results by increasing the stability of network and speed up the training process. In neural network the gradient can be zero or infinite value when the depth of the network is increase, which causes some problems such as vanishing gradient and exploding gradient. To avoid these problems, batch normalization is adopted with shallow CNN to normalize the feature representations. The BN is defined as follows:

$$\hat{x}^k = \frac{x^k - E(x^k)}{\sqrt{Varx^k + \epsilon}} \tag{5.5}$$

where $E(x^k)$ and $Var(x^k)$ considered as the mean and the variance of features, and ϵ is a constant positive value used to increases the model numerical stability.

The statistical characteristics of the current mini-batch are used by BN to normalize the intermediate representations. Each activation is converted to a Gaussian distribution with a mean of 0 and a variance of 1. In a network with normal parameterization, the scale of each layer varies dramatically. As a result, the optimum learning rates vary across layers.

As a consequence, training networks with conventional parameterization takes a long time. BN, on the other hand, improves the resilience of the training parameter scale by keeping the scale of each layer constant. In comparison to standard parameterization, BN results in quicker convergence and improved generalization capabilities. During the training phases, the batches of data entered one at a time and then the mean and variance of each mini-batch were utilized in the BN transformation; this method increase the speeds of the training process. In the test step, the trained global mean and variance with a one data sample were acquired through the transformation of BN to stabilize the test findings.

6. *Dropout:* when there are less input data and too many training parameters in deep learning models, it may lead to overfitting issues, which manifest as high training accuracy and poor testing accuracy. To prevent overfitting and speed network training, our CNN model employs the dropout method, which involves randomly discarding a given probability on the fully connected layer.

5.1.2 CNN model initialization

The weight initialization technique of the neural network has a significant effect on the model's convergence speed and performance. We utilize random parameter initialization technique to investigate the impact of utilizing a limited number of layers. The parameters have a gaussian distribution with a mean of 0 and a variance of 1, thus the parameter values were properly initialized to small random integer value close to 0. The output value approaches 0 rapidly as the number of convolutional layers increase using the parameter initialization given in Figure 5.3. The gradient becomes smaller as the number of layers increase, making it difficult to update the model parameters. However, the proposed approach with small number of convolutional layers is unaffected by this.

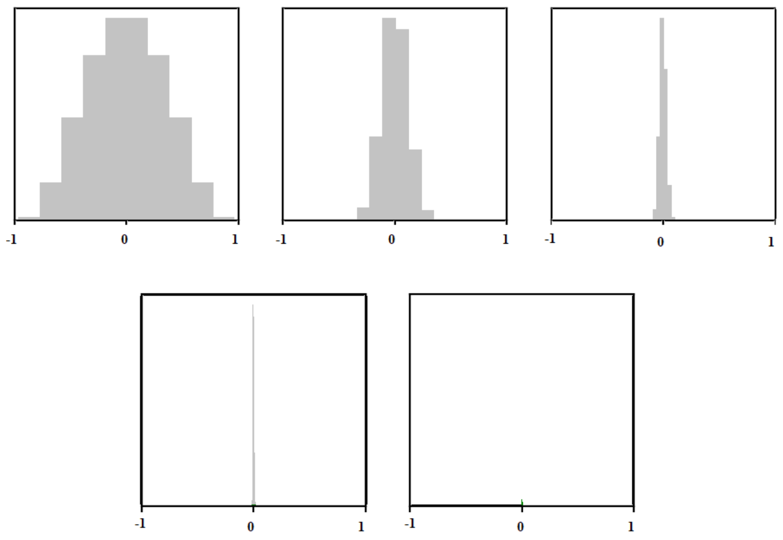


Figure 5.3: Model parameter with weight random initialization.

5.2 Experimental setup

5.2.1 Experimental datasets and data augmentations

As stated in Section 4.2, this chapter focuses on detecting one of the most sophisticated covert timing channel assaults known as Cautious Covert Timing Channels (CCTC). CCTC is a more sophisticated cyberattack that attempts to mimic overt traffic behavior (packet inter-arrival time) to some extent. This kind of covert channel behaves similarly, with packet inter-arrival intervals comparable to overt traffic. The packet delay was adjusted to half of the typical inter-arrival time of overt traffic to mimic the CCTC assault. When coupled with the network's latency, this delay causes less irregularity in the traffic, making it more difficult to detect by security detection methods. We tested the CCTC attack utilizing several traffic flow lengths and one size of the hidden message in the traffic flow.

In the preprocessing phase, the collected collection of communication flows between the two communicating systems includes both overt and covert traffic. The inter-arrival periods for these traffic flows are then retrieved and saved in the flow dataset. Inter-arrival times are classified into four flow lengths: 256, 784, 1024, and 4096 packets. The covert message was injected into the traffic flow 64 bits at a time in each flow length. Finally, each sub-flow of inter-arrival times was put into a 2D matrix for image generation. Then, each matrix is viewed as a colorful image by transforming each of its normalized values to a color pixel. We created various traffic images for our model based on the four lengths of traffic flow: 256=(16×16), 1024=(32×32), 4069=(64×64), and 16384=(128×128).

We created two kinds of datasets to test the performance of the proposed CNN detection model. The first kind is for channel detection and includes 16000 overt and covert images, which are 4000 images for each size. The second kind is channel localization, which includes 6000 images created based on inserted covert data with 64 bits in three places in the traffic flow: the beginning, middle, and finish. The specifics of each dataset are given in Tables 5.1 and 5.2.

Deep learning methods often need a significant amount of data to train the model without overfitting problem. However, if the sample size is inadequate, the data augmentation method is the best way to expand the number of data samples. A suitable data augmentation technique may successfully prevent overfitting issues and enhance the model's resilience. In general, the transformation of the original image data such changing the position of pixels while maintaining the features is utilized to create new data. Techniques for data augmentation include flipping, zooming, rotating, flipping, zooming, shifting, scaling, and color modification. Typically, several data augmentations are required. Table 5.3 shows the image augmentation parameters that are utilized to enhance the quality of data samples in this chapter.

Table 5.1: Detection datasets

Covert message size	64 bits
Delay time	0.5μ
The mean IATs of overt traffic (μ)	0.0664 seconds
Number of dataset versions (each version for one image size)	4
Number of images (all datasets)	16000
Number of all images per dataset version	4000
Number of overt images per dataset version	2000
Number of covert images per dataset version	2000

Table 5.2: Localization datasets

Covert message size	64 bits
Delay time	$.5\mu$
The mean IATs of overt traffic (μ)	0.0664 seconds
Image size	32×32
Number of images (all datasets)	6000
Number of images per dataset version	2000
Location of covert message	Beginning, Middle, End

Table 5.3: Setting of data augmentation.

Methods	Setting	Methods	Setting
rotation-range	30	shear-range	0.1
vertical-flip and horizontal-flip	true	zoom-range	$[0.8, 1.2]$
height-shift	0.1	width-shift	0.1

5.2.2 Model hyperparameters

The CNN model includes a number of hyper-parameters that affect the network topology such as, the number of filters and type of optimizer. The model's performance may vary greatly depending on the choice of hyper-parameters used. For example, the number of layers in the CNN model was determined via numerous trials. Because the convolution layers were found to be optimum, and because it is critical to use tiny kernels for convolutional layers, the kernel size of convolution layers was set at 3x3.

To find the optimum classification model, we evaluated various model hyper-parameters. The CNN model's hyper-parameters are given in Table 5.4. We modified and updated the model hyper-parameters in the search space before doing a grid search to get the optimum value. For example, when the number of training epochs approaches 15, the model test results do not improve continuously, and the training time becomes more longer. Another critical hyper-parameter that controlled whether the goal function converged to a local minimum was the learning rate. A suitable learning rate may cause the objective function to converge to a local minimum in a reasonable amount of time. In addition, we adjusted the learning rate decay at 0.001. As the number of iterations grew, the learning rate progressively dropped, speeding up model training in the early stages.

Table 5.4: CNN model hyperparameters.

Model hyperparameter	**Search space**	**Choose value**
Epoch	10-30	15
Dropout probability	0.1–0.9	0.5
Batch numbers	32-512	128
Learning rate	0.0001–0.1	0.001
Optimization algorithm	SGD, Adam, RMSprop	Adam
Loss function	-	Cross Entropy

5.3 Experimental results and analysis

We assess the proposed CNN performance and efficiency with batch normalization by running a series of tests on image datasets, including binary-category detection (overt and covert) and three-category localization (beginning, middle, end). More precisely, the experiments in this chapter seek to:

a. Compare the outcomes of our proposed method for identifying CTCs based on utilizing the batch normalization technique.

b. Test the efficacy of various model hyper-parameters.

c. Determine the impacts of various traffic image sizes.

d. Compare the results of shallow CNN to the results of existing machine learning approaches for identifying CTCs.

e. Contrast the results of shallow CNN with those of existing deep CNN methods for identifying CTCs.

We utilized 5-fold cross-validation to assess the CNN model. The dataset was broken down into 10 sub-samples. In each running experiment, nine samples were used as the training set, and one sample only was utilized as the validation set. The training and testing process of model were repeated 10 times, and then the average of the findings was used as the final assessment value.

5.3.1 Effects of using batch normalization technique on the CNN model performance

We provide two variation instances of CNN with various assumptions for utilizing batch normalization, as shown below, and compare them to the CNN model that utilizes batch normalization after each convolution layer to demonstrate how batch normalization may speed network training and enhance model performance:

case 1: Remove just the batch normalization approach after the first convolutional layer, leaving all other layers and model parameters without changing.

case 2:Delete all batch normalization strategy after the first and second convolutional layers, while all remaining layers and model parameters stay the same.

The accuracy results of CNN, CNN-case 1, and CNN-case 2 are given in Table 5.5. According to the table findings, the CNN model with BN after each convolutional layer has the greatest accuracy of 96.75%, which is 0.3 % better than the CNN model based on case 1 and 0.55 % higher than the CNN model based on case 2. As a consequence, we can infer that our classification accuracy obtains the best classification result when combining CNN with BN, proving the validity and efficiency of the proposed model.

Furthermore, Figure 5.4 depicts the training accuracy of the proposed CNN model and the other two architectures based on cases 1 and 2 across 200 epochs. The experimental findings obtained with BN show significantly quicker convergence than the results obtained with the standard parameterization without BN. The accuracy achieved with the standard parameterization is poor and unreliable. The BN method

Table 5.5: Classification accuracy of proposed model.

Model	Accuracy (%)
CNN	96.75
CNN-case1	96.45
CNN-case2	96.20

is used to normalize the parameters of each layer such that they follow the conventional Gaussian distribution. As a result, parameter updating is more stable than conventional parameterization. This indicates that the CNN with BN can learn data characteristics more effectively and quickly. The BN method may accelerate network convergence while also improving model accuracy.

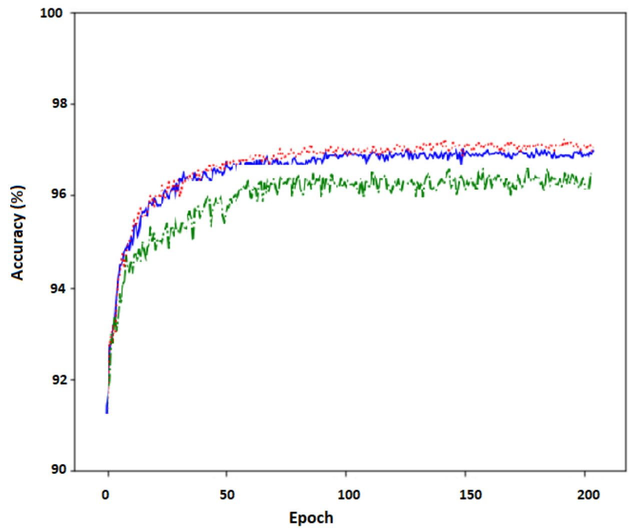


Figure 5.4: Classification results of three CNN model cases.

Table 5.6 provides additional accuracy measures: precision, recall, and F_1 for each class in our dataset (covert/overt). The accuracy for the covert and overt classes is determined to be 95.8 % and 96.1%, respectively, based on the findings. Furthermore, the recall rates for the two groups are 95.5 % and 96%, respectively. High F_1 values are also present. Because of the high accuracy and recall values, the proposed CNN detection model identifies the CTC at a very high rate and is very likely correct.

Table 5.6: Evaluation results for two classes.

Traffic type	Precision (%)	Recall (%)	F_1 (%)
Covert	95.80	95.50	95.65
Overt	96.10	96.00	96.05
Average	95.95	95.75	95.85

5.3.2 Hyperparameters effect on CNN model performance

This section includes various experimental comparisons depending on model hyperparameters such as kind of activation function, the number of layers, and learning rate value, and represents how thees parameters affect the model performance. Table 5.7 compares the performance of the CNN model with various numbers of convolutional layers and sigmoid and the ReLU functions. Because we discovered that the model might enhance performance, we set the unit number of completely connected layers to the same as its input units. Also the cross-entropy loss is the related cost function.

The experiment findings indicate that increasing the number of convolutional layers with changing the functions have no effect on the performance of the CNN model. However, the impact on the model's run time. Furthermore, the findings indicate that the ReLU activation function saves more time than the sigmoid function.

The run time is unaffected by the convolutional layers using the ReLU function. In terms of overall performance, the ReLU is a more useful function than the sigmoid.

Table 5.7: Accuracy results based on various convolutional layers and activation functions.

Conv layer	Activation function	Model accuracy	Running time(s)
1	Sigmoid	95.10	5.50
	ReLU	95.51	4.00
2	Sigmoid	96.60	7.22
	ReLU	96.75	5.03
3	Sigmoid	96.20	9.50
	ReLU	96.55	6.20

Table 5.8 also represents the evaluation results of using various activation functions in the fully connected layer and a pooling layer. The experiment findings show that setting the ReLU function with fully connected layers will decrease the run time of the model. The results also indicate that adding the max pool layer with sigmoid function increase running time of model compared to using the max pool layer with ReLU function. Thus, it is useful adding max pool in the model when the sigmoid function is applied in the model.

Table 5.8: Performance results based on using a max-pooling and fully connected layers with various functions

Conv layer	Activation function	Accuracy (%)	Time(s)
Max-pooling	Sigmoid	96.60	7.22
	ReLU	96.75	5.03
Fully connected	Sigmoid	96.60	7.51
	Tanh	96.65	6.50
	ReLU	96.75	5.03

Moreover, we assess the effect of using the learning rate set between 0.0001 to 0.1 on model detection rate and false alarm rate. As mentioned, the detection rate represents the proportion of covert traffic samples that detect correctly, while the FAR represents the proposition of the overt traffic data that are classified incorrectly. Figure 5.5 shows the increase of learning rate is affect on the DR and the FAR, and we get the best DR when we set the learning rate at 0.001. Based on this value of learning rate, the model can classify the overt and covert samples correctly.

5.3.3 Image size effect on CNN model classification

In CNN proposed model, image size plays an essential role that affects model classification results. The small size of the image could not help extract enough information, which causes losing the attack properties. At the same time, using a large image can increase the computational time without any improvement in the accuracy of classi-

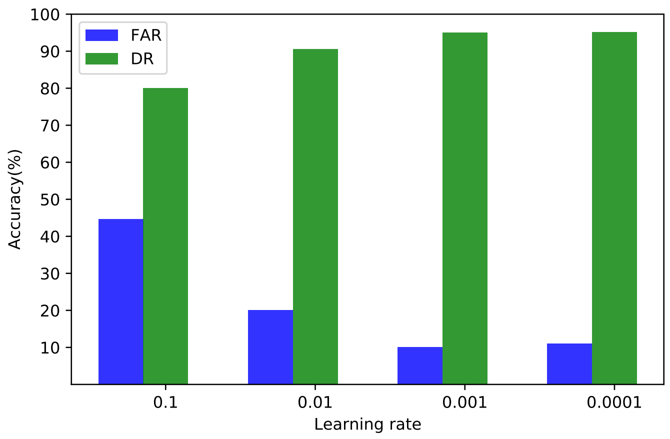


Figure 5.5: Performance test with increasing the learning rate.

fication.

In this dissertation, we conducted a variety of experiments based on using different image sizes: (16 $\times$ 16), (32 $\times$ 32), (64 $\times$ 64), and (128 $\times$128) as discussed in Section 5.2.1. We used fixed image size as required for CNN model input. Figure 5.6 show the accuracy results of 2-classes classification using different image sizes. Table 5.9 also shows various performance metrics of these experiments. The results show the model performance is increase when the image become bigger. However, a large size of image needs more running time. We noted that when the image size was greater than 32 $\times$ 32, the model performance did not positively correlate with the image size, and the training time was increase rapidly. Therefore, the CNN model with input image size is 32 $\times$ 32 pixels achieve the best classification results.

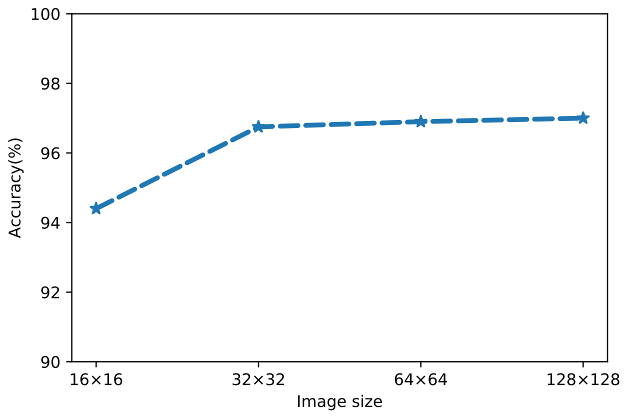


Figure 5.6: Accuracy results of using different image sizes.

Table 5.9: Image sizes effect on model performance measures.

Size of image	Precision	Recall	F_1	Time(s)
16 × 16	94.46	92.00	92.56	4.00
32 × 32	95.95	95.75	95.85	5.03
64 × 64	95.93	95.22	95.84	12.00
128× 128	95.44	95.00	95.22	27.00

5.3.4 Effects of image reshaping on CNN model classification

This work also studies the effect of image shape on model performance. We used the inter-arrival times of traffic to generate a non-square image with different dimensions. We choose the image size 16×16, and then we reshape it to three non-square image shapes include 2×128, 4×64, and 8×32. Table 5.10 shows the performances of non-square image shape were similar to a square image shape, which means width and height of the image did not affect the pattern of covert traffic in the image and performance of the classification model. Thus, a shallow convolution network with a small size of the filter, one stride one linear layer at the end, would be worked well; it is very easy for a filter to detect the horizontal split of pattern in the covert images.

Table 5.10: Model classification results using different image shapes.

Size of image	Accuracy
64 × 4	94.451
32 × 8	94.455
128 × 2	94.438
16 × 16	94.46

5.3.5 A comparison of the CNN model performance with other conventional machine learning methods

We compared our model to five existing CTC detection models that utilized standard image feature extraction techniques to verify the effectiveness and efficiency of our suggested methodology. These techniques collected characteristics from traffic pictures before using machine learning algorithms to detect and categorize CTCs. We chose a 32 x 32 image size for this comparison. The results of our method are shown in Figure 5.7 and Table 5.11 when compared to the results of the five other CTC detection models: Random Forest (RF), Multilayer Perceptron (MLP), Support Vector Machine (SVM), Decision Tree (DT), and Naive Bayes (NV). We can observe that our model has improved in terms of accuracy and detection speed. The other methods performed slightly worse since their performance is dependent on particular characteristics.

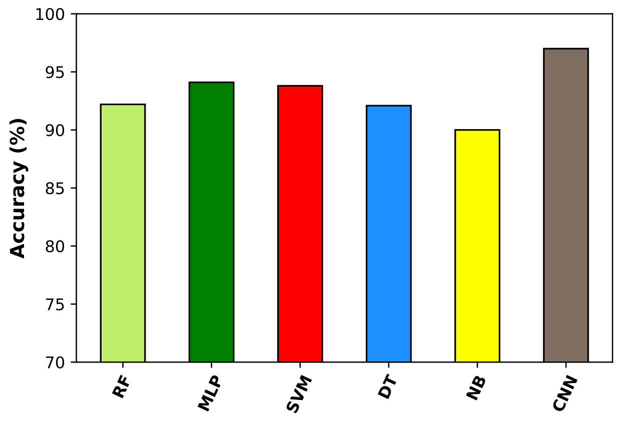


Figure 5.7: The comparison of accuracy results.

Table 5.11: Classification results of shallow CNN model with traditional machine learning methods.

Classifier model	Precision	Recall	Train time(s)	Test time(s)	Error rate
RF	92.11	90.00	63.30	51.39	7.75
MLP	93.01	93.00	36.02	25.01	5.89
SVM	92.00	89.00	73.96	59.20	6.05
DT	91.80	88.99	66.25	53.12	7.91
NB	90.00	90.88	25.10	20.02	9.78
CNN	**95.95**	**95.75**	**5.03**	**3.50**	**3.25**

5.3.6 A comparison of the shallow CNN model performance with existing deep CNN methods

Table 5.12 compares the proposed model with VGG16, AlexNet, and ResNet, which contain a high number of convolution and fully linked layers, to verify the benefit of the proposed shallow CNN method over deep CNN alternatives. Convolutional layers are denoted by Conv in the final column of the table, while fully linked layers are denoted by Fc (including output layer). The table results indicate that the suggested model achieves an accuracy of 96.75%, which is higher than deep CNN models with multiple convolutional layers and filters. ResNet, for example, consists of 419 filters with 5 x 5 convolutions in first layer. The shallow CNN method employs two 3 x 3 convolutions with 32 and 64 filters for the first and second convolutional layers, respectively. With additional convolution layers, the suggested method outperforms AlexNet by 6.74%, achieving a test accuracy of 90.01%. It is also 9.73% and 8.75% more accurate than the VGG and ResNet models, with accuracy of 87.02% and 88%, respectively.

In addition, Table 5.12 displays the comparative findings in terms of training time on the datasets. In most instances, the suggested shallow CNN model outperforms all of the previously stated deep CNN techniques in terms of computing cost, network structure, and training time.

Table 5.12: Classification results of shallow CNN model with other deep approaches.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F_1(%)	Time(s)	Con./Fc
AlexNet	90.01	89.02	86.00	87.40	25	5/3
VGG16	87.02	86	85.50	85.75	42	13/3
ResNet	88.00	85.5	87.00	86.24	35	$\geqslant$ 7/1
Proposed CNN	**96.75**	**95.95**	**95.75**	**95.85**	**5.03**	2/2

5.3.7 Localization of CTC in traffic flows

One critical goal is to identify the segments of traffic flows (i.e., sets of packets) that include the hidden message (s). It offers the ability to warn and enables just the harmful portion of traffic flows to pass through while allowing the remainder of the traffic flow to pass through. This accurate identification substantially minimizes overt traffic interruptions caused by non-malicious apps.

This section describes our approach's performance in locating hidden messages inside traffic flows. We devised an experiment in which hidden messages are inserted into all communication flows at one of three points: beginning, middle, and end. Then we create traffic images based on the three section locations. Each of these designations corresponds to the position of the hidden message in a particular traffic flow. For example, if the malicious agent is configured to inject the hidden message at the start of traffic, it will also give the label at the start of that traffic image.

It also does the same for the other two labels, middle and end. This labeled collection of covert traffic images is then input into the CNN model, which is used to evaluate the performance of our method in locating the right traffic flow segment that included the hidden message(s) using an image-based model. The accuracy of pointing the malicious network flow segment that includes the hidden message using different methods is shown in table 5.13.

As demonstrated in Table 5.13, our method identified the segments inside traffic flows that carried the hidden message with 94.01% accuracy. We also ran the SVM, AlexNet, Decision Tree, and NaiveBayes to give a baseline for comparison. Surprisingly, the AlexNet came in second with a detection accuracy of 92.01%. SVM came in third place with an accuracy of 91.05%. NaiveBayes finished last with an accuracy of 89.50%. Furthermore, when compared to previous models, the proposed model has a lower positive rate and false-negative rate.

Table 5.13: Localization results of shallow CNN model with other approaches.

Classifier model	Accuracy	FP	FN
SVM	91.05	5.53	12.98
DT	90.54	5.13	14.39
NV	89.51	5.22	17.22
AlexNet	92.01	4.29	12.29
CNN	94.01	3.17	6.72

5.4 Chapter summary

For image classification, this dissertation employs a batch normalization technique in combination a shallow CNN model with a batch normalization method to accelerate convergence time and improve image classification accuracy. The CNN model features a four-layer basic structure with decreased time complexity.

The proposed shallow network is defined by its few parameters, resistance to overfitting, and improved generalization. Based on the generated and labeled training and test sets, the suggested design outperforms SVM, ResNet, VGG, and other models for certain covert traffic classification tasks, demonstrating the effectiveness and feasibility of this shallow neural network architecture. As a consequence, the particular problems associated with real-world CTC classification are addressed. Classification related activities in many applications, such as the smart visual Internet of Things, now have an alternate solution thanks to the new model. It also reduces deep neural network complexity while maintaining their strength.

Chapter 6

CTC Detection Based on Sequential Time Series Using Recurrent Neural Network and 1D Convolution Neural Network

Natural language translation and time series classification are just a few of the many applications that have benefited from the usage of recurrent neural networks (RNN) and one-dimensional convolutional neural network (1D-CNN). They are widely used because to their capacity to extract high-level characteristics from sequential raw data without requiring human intervention and to achieve a high success rate. RNN and 1D-CNN extract high-level semantic relations based on adjacent regions correlations.

A RNN model examines the relationship between current and previous states to identify the order in which events happened. 1D-CNN shines when dealing with time series data because it can be used to extract features from the data by translating it to the time direction. 1D-CNN algorithms that use convolutional methods with a focus on adjacent inputs have the potential to find novel pattern combinations.

In this dissertation, RNN, 1D-CNN, and their network combinations are utilized to construct time series covert timing channel detection models. These models, which use deep learning methods, are capable of detecting certain sequences that may aid in distinguishing between covert and normal communications. Working directly with raw data and learning feature representations from the time series data removes the requirement for feature engineering.

By providing covert timing channel detectors that can evaluate packet timing without the need for extra ore-processing methods, we are able to provide realistic and rapid network monitoring tools that can immediately identify covert assaults. Because of their quick reaction times, these detectors may be utilized in real-time applications. This chapter starts with an overview of the RNN model, 1D-CNN, and the networks it generates. Then evaluate the model's efficiency and quantitative value, an array of sequentially-based datasets.

6.1 CTC detection using a sequential time series

6.1.1 CTC detection with RNN sequential data

A low-rate covert channel attack is difficult to detect because it seems to be genuine network data from the victim's end. Meanwhile, a series of careful CTC assaults on systems must be launched over time. It would not be harmful to the network resources if this were not the case. This highlights the significance of historical information in covert channel detection. Because of the lack historical pattern in the learning model, the single-packet detection technique cannot enhance performance.

In general, the nodes within the same layer are not connected in the most type of neural networks. This structure is not appropriate for processing sequential data in order to identify a historical trend in the learning model. As a result, neural networks with inner layer connections, such as the recurrent neural network (RNN) [87], are introduced. RNN fully exploits the context of sequential data and provides a scalable model for variety problems involving time series data [88]. By storing past states, RNNs learn feature representations from input data. RNN may be used to identify CTC attacks using sequential traffic inter-arrival times for these reasons.

We propose a detection method based on RNN approaches to detect CTC attacks based on the benefits of RNN. Our detection method employs a series of traffic inter-arrival timings to understand the nuanced distinction between attack traffic and normal traffic. To detect CTC attack, historical data is put into the RNN model. It aids in identifying recurring patterns that indicate CTC attacks and locating them in a long-term traffic sequence.

Traditional RNN structure are not appropriate for categorizing lengthy sequence data in reality due to integrate only limited prior [89] and are difficult to train correctly [90]. To address the limitations of conventional RNNs, RNN variations such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks have been developed. Error reduction in GRUs is accomplished via distance weighting. LSTMs enhance the learning impact by remember and forget particular information. We examined different kinds of RNNs experimentally, and our ultimate decision was the LSTM type, as demonstrated in Section 6.1.5.

The RNN detection model contains of an input layer, two LSTM layers with 128 hidden states, a full connection layer and an output layer as shown in Figure 6.1. The number of hidden states represents the length of the and the number of LSTM layers represent the depth of the neural network. In our study the traffic inter-arrival times are the input data, where these input data then fed into the LSTM units.

Each LSTM unit calculates the current state of input using the previous states and then send the results to the next LSTM unit. Finally, the softmax classifier receives the state results from last LSTM unit and calculate the probability for predicted result. The model hyper-parameters such as dropout rate and batch sizes are described in Table 6.1. The formulas of LSTM layers are discussed as follow:

In the recurrent neural network, the recursive function in the layer accepts one vector of the input sequence at time t and the previous hidden state h_{t-1} and returns

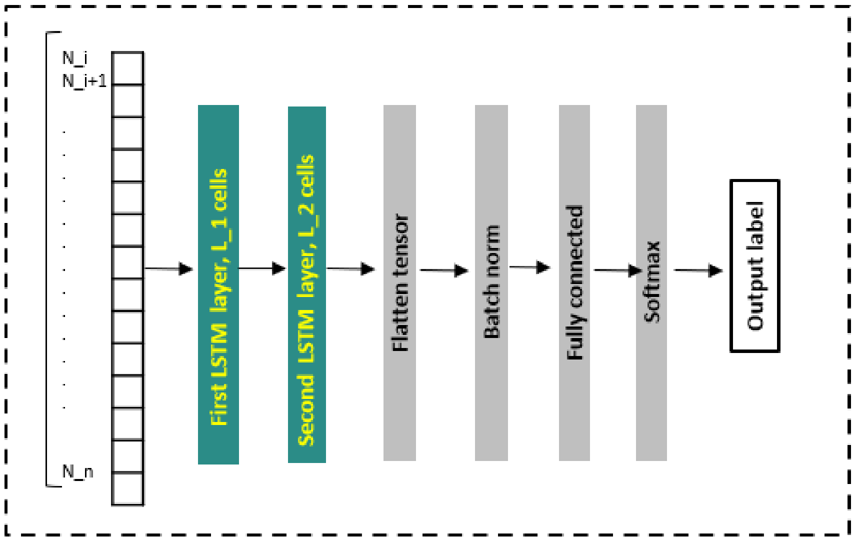


Figure 6.1: Overview of the proposed LSTM sequential data based CTC detection model.

the new layer state:

$$h_t = f(v_t, h_{t-1}) \tag{6.1}$$

The first hidden state is commonly initialized as an zeros vector to avoid vanishing gradient. The LSTM unit is made up of four sub-units: an input gate, a forget gate, an output gate, and a candidate memory cell. They are calculated using the following formulas [87]:

$$\begin{aligned} i_t &= \sigma(w_i[h_{t-1} + v_t] + b_i) \\ o_t &= \sigma(w_o[h_{t-1} + v_t] + b_o) \\ f_t &= \sigma(w_f[h_{t-1} + v_t] + b_f) \end{aligned} \tag{6.2}$$

where i, f, o represent input, forget, and output layer gates, respectively; h denotes hidden state; v_t denotes input at current state; σ denotes sigmoid activation function;

w_x weight for respective gate(x); b_x biases for respective gate(x). The LSTM unit's activation is calculated as:

$$\check{c} = \tanh(w_c[h_{t-1}, v_t] + b_c) \tag{6.3}$$

$$c_t = f_t * c_{t-1} + i_t * \check{c} \tag{6.4}$$

$$h_t = o_t * \tanh(c_t) \tag{6.5}$$

The recurrent network's output sequence is $(h_1, h_2, dots, h_L)$. Typically, the final output vector h_L is always immediately translated to two binary classification outputs.

Table 6.1: LSTM model hyperparameters.

Hyperparameters	**Value**
Number of layers	2
Activation function	tanh
Number of neuron	128
Dropout probability	0.5
Number of neuron of connected layers	128,1
Activation function of connected layers	Relu, sigmoid

6.1.2 CTC detection using 1D-CNN sequential data

A convolution neural network is divided into two sections: feature extraction and classification. The feature extraction component is automatically in charge of obtaining useful characteristics from raw data (time series). The portion of classification, on the other hand, is utilized to correctly categorize data by using the extracted characteristics. In summary, these two parts collaborate to accomplish the primary job of this model.

Convolution layers and pool sampling layers are used in the feature extraction process. Convolution layers may be thought of as a filter that enhances original data characteristics while reducing noise. Convolution is conducted between the top layer's feature vectors and the current layer's convolution kernel. Finally, the activation function returns the outcomes of convolution computations. The feature map x_j is computed for each convolutional layer output as follows:

$$x_j = f((\sigma x_j * W_{ij} + b_j) \tag{6.6}$$

where x_j is the vector corresponding to the j^{th} convolution kernel, B_j is the bias coefficient, and f is a nonlinear activation function.

To decrease the dimension of data while preserving valuable information, the pool employs the concept of local correlation (down) sampling layer. At the same time, pooling technology is used to preserve the characteristics of displacement, invariance, and scaling. The pool sampling layer has the role of extracting more features (average or maximum); in the meanwhile, the spatial resolution between hidden layers is diminishing, and its formula is as follows:

$$x_j = f(\beta_j down(x) + b_j) \tag{6.7}$$

Where $down()$ denotes the downsampling function and β_j denotes the weighting coefficient

In addition to the convolution and sampling layers, there are still input and output layers. Before training the network, the network model is established, followed by splitting the time-series as input data and specifying the target's output vector.

The CNN was first intended to handle two-dimensional data. We must, however, deal with one-dimensional data. As a result, we needed to change the structure of the CNN model and utilize the 1D-CNN structure. The improved CNN model for CTC detection and classification is illustrated in Figure 6.2 and explained in Algorithm 2. It has three convolution layers, three max-pooling layers, a complete connection layer, and an output layer. The $softmax$ classifier obtains the output label at the conclusion of 1D-CNN CTC detection. The softmax layer computes the loss between predicted and true values during the training phase. Based on the loss numbers, the network weights should be adjusted. During the testing phase, the softmax layer computes the probability of the class label categories (over covert).

The input of each neuron in the 1D-CNN model is linked to the output of the preceding layer, which is utilized to extract local characteristics. The input is a traffic-inter arrival time segment with n sample points.

The weights are calculated by the neural network using the chain rule. So, in case of increasing the depth of network, the gradient becomes close to being zero or infinite value, which causes the vanishing and exploding gradient issues. To normalize the intermediate representations in this issue, we use Batch normalized (BN).

Algorithm 2: 1D CNN-Based CTC Detection

```
Input: Training dataset and model parameters
Output: Predicted classes
for $I_{th}$ epoch iterate
    for $J_{th}$ mini-batch iterate
        Reshape input data as $1 \times n$
        Compute convolution results
        Compute max-pool results
        Compute the result through the batch norm
        Compute the result through dropout
        Punish the result through connected layer
        Compute the training error
        Update the model weight and bias
        Output classification results using $softmax$
    end for
end for
```

The 1D-CNN detection model includes a number of hyper-parameters such as the number of filters and type of optimizer. The model's performance may also vary significantly depending on the hyper-parameters used. The grid-search technique was used to identify hyper-parameters optimal for the model based on the datasets. The grid search technique attempts every conceivable combination of hyper-parameters to find the optimal hyper-parameter for a dataset. Table 6.2 describes the model

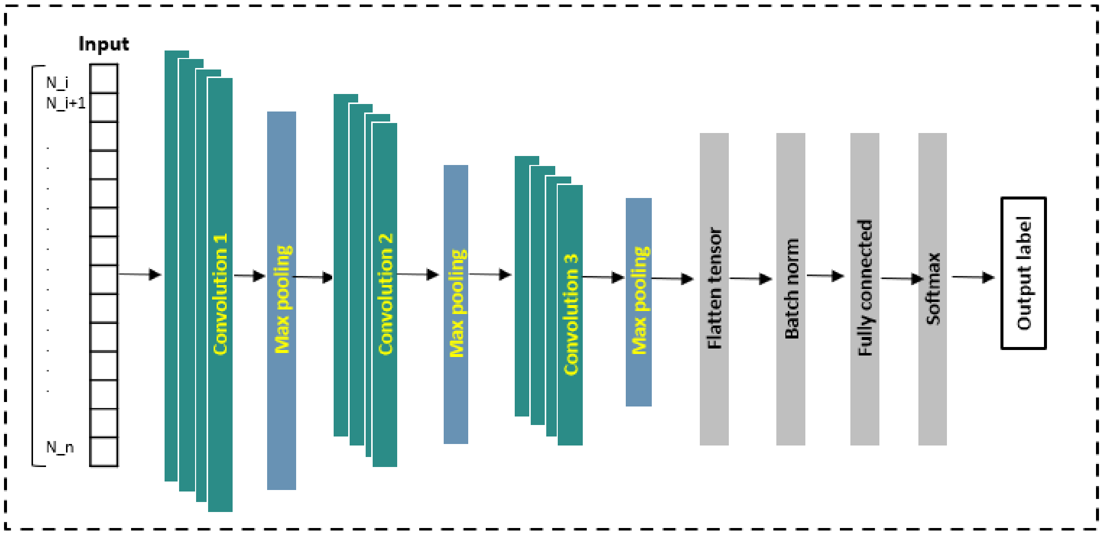


Figure 6.2: Overview of the proposed 1D-CNN sequential data based CTC detection model.

hyper-parameters utilized in this research.

Table 6.2: 1D-CNN model hyperparameters.

Hyperparameter	**Search spaces**	**Choose value**
Number of filters	16, 32, 64, 128	32
Filter size	3-11	5
Batch sizes	32–128	32
Dropout probability	0.1–0.9	0.5
Learning rate	0.0001–0.01	0.001
Optimization algorithm	RMSProp, Adam, Nesterov	Adam
Loss function	-	Cross Entropy

6.1.3 CTC detection based combination network

As previously stated, 1D-CNN and pooling procedures may extract abstract and local higher-level features. It is beneficial in terms of gaining a better knowledge of how traffic feature sequences are linked to one another. Simultaneously, the LSTM includes memory cells that may address the temporal issue by learning the temporal structure of sequential traffic input and subsequently attaining high-level data abstraction. LSTM concentrated on temporal correlations of traffic evolution and ignored spatial correlations from a network standpoint. Where spatial traffic characteristics may assist in detecting CTC at a distinct place in network traffic.

This section describes a network detection model that combines 1D-CNN and LSTM in one model to identify CTC more accurately based on the performance of extracting spatial and temporal traffic characteristics. The concept of merging 1D-CNN with LSTM for classification has been studied before, however prior studies were in other study areas, not for CTC detection. When two distinct algorithms are integrated, the stacking order of the algorithms plays a major role in generating the optimum performance of CTC detection. As a result, this research examines the impact of layer ordering in network combination on the performance of the CTC detection model.

The combination network was chosen following a series of tests with the number and kind of layers that were installed. First, LSTM layers are applied on top of the convolutional layers (CNN-LSTM). The input of traffic inter-arrival times was fed into the CNN model first, and the output of the CNN model was supplied into the

LSTM layer. The outputs of the LSTM layers are then concatenated and sent into the output layer, which classifies the input data into one of two traffic classifications (over/covert). This method is justified by the fact that there are dependencies between various portions of traffic inter-arrival times in an input sequence file. On top of the CNN layers, recurrent layers will assist summarize the whole input file content into features that will be sent to the output layer.

Figure 6.3 depicts an overview of the CNN-LSTM model and how the LSTM is layered on top of CNN. It is made up of an input layer, a convolution layer, a max-pooling layer, two LSTM layers, a connected layer, and an output layer. The $softmax$ classifier at the conclusion of the CTC detection model determines the output label. The feature filtering will be conducted before the temporal modeling in this combo model. The sequential input data is processed by CNN layers first for feature reduction, and then the smaller feature dimensions are fed into LSTM layers for data sequence learning.

The CNN-LSTM architecture may contain a variety of structures based on a number of factors such as the number of convolution layers, kernels, LSTM layers, and LSTM units. These parameters may influence learning performance of model due to extracting additional features from data, and speedup the training process. For example, our data that input the CNN-LSTM model is in the form of sequence data containing spatial-temporal information through a sliding window method, a kernel of size five was utilized to minimize information loss.

The CNN-LSTM model parameters are given in Table 6.3. The CNN-LSTM input is a time vector of length n (depending on various data lengths), flow the convolution

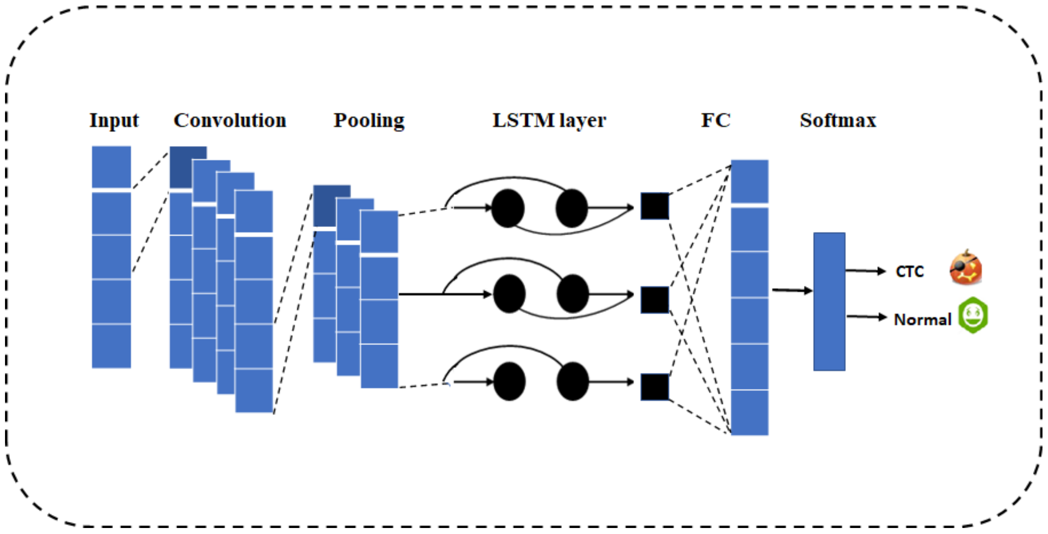


Figure 6.3: Combination of CNN and LSTM.

and pooling layers, then the LSTM layer. In the proposed model, we utilized tanh as the model's activation function.

Table 6.3: The CNN-LSTM model structure.

Layer	Filter number	Size of kernel	Stride number
Convolution layer	32	5	1
Activation function(tanh)	-	-	-
Batch Normalize(32)	-	-	-
Max pooling	-	4	1
Convolution layer	32	5	1
Activation function(tanh)	-	-	-
Batch Normalize(32)	-	-	-
Max pooling	-	4	1
LSTM layer(64)	-	-	-
Activation function(tanh)	-	-	-
Dropout probability (0.4)	-	-	-
Fully connected layer (128)	-	-	-
Activation function (tanh)	-	-	-
Dropout probability (0.4)	-	-	-
Softmax	-	-	-

As illustrated in Figure 6.4, the second design of a combination network employs CNN layers on top of LSTM layers (LSTM-CNN). The CNN layer outputs are then concatenated and sent to the output layer, which classifies the input data into traffic classifications (over/covert). The architecture of the LSTM-CNN detection model is shown in table 6.4. To learn temporal characteristics from sequential input data, LSTM is used as the first two layers. The architecture may be thought of as a deep through time step with LSTM memory cells producing output features. LSTM layers are used in conjunction with memory cells to recall all feature inputs. The LSTM output is then transferred to the CNN layers, which reduces feature variance. The architecture used is a one-dimensional convolution layer with 5x1 feature filters distributed throughout space. The convoluted output is then subjected to a 4x1 size max-pooling operation.

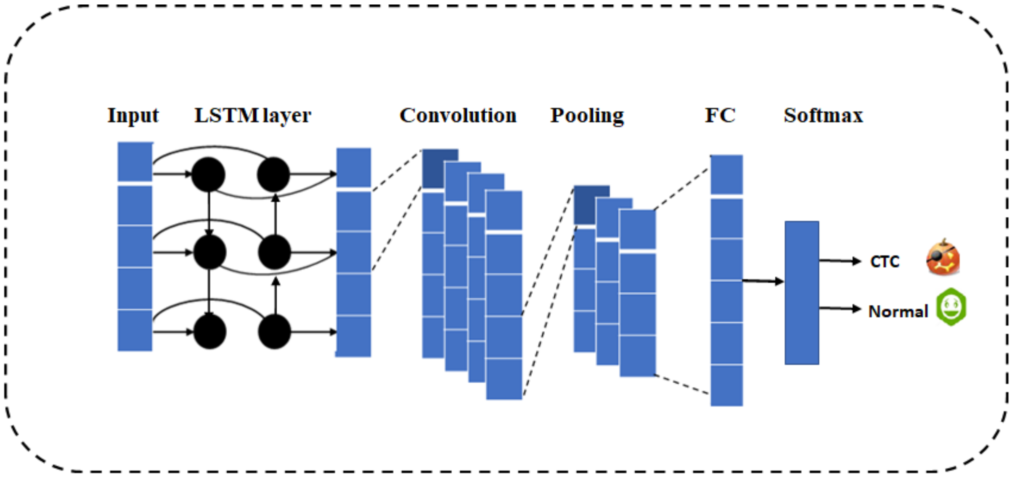


Figure 6.4: Combination of LSTM and CNN.

Table 6.4: The LSTM-CNN model structure.

Layer	Filter number	Size of kernel	Stride number
LSTM layer with 64 units	-	-	-
Dropout probability	-	-	-
LSTM layer with 64 units	-	-	-
Dropout probability	-	-	-
Convolution layer	32	5	1
Batch Normalize(32)	-	-	-
Max pooling	-	4	1
Flatten layer	-	-	-
Fully connected layer (128)	-	-	-
Dropout probability(0.5)	-	-	-
Softmax	-	-	-

6.1.4 Experimental dataset

This chapter uses actual data gathered via the proposed channel BCTC, which was developed and described in Chapter 3. This chapter focuses on identifying Cautious Covert Timing Channels, which are among the most sophisticated covert channels (CCTC). CCTC is a more sophisticated hack that attempts to mimic overt traffic behavior to some extent. This kind of covert channel behaves similarly to overt traffic, with packet inter-arrival intervals that are comparable to overt traffic. Because of its resemblance to overt traffic, CCTC is a more difficult kind of covert channel to identify and is difficult to detect by security detection methods.

In this chapter, we utilized the collected network flows for covert and overt traffic to generate datasets. The covert message was injected into the overt flows at a size equal to 50% of the flow size. Then, with a well balanced frequency distribution, these flows are grouped into two separate designated overt and covert flows. The dataset includes five different flow length sizes (32, 64, 128, 256, and 512). There are 4000 flows in each length scenario, 2000 with covert data and 2000 with overt data. The end result is a time series of feature vectors for each flow.

6.1.5 Recurrent network analysis

This section begins by comparing several kinds of RNNs. The effect of sequence length, hidden state number, and hidden layer number on model performance is then discussed.

To choose the most efficient RNN type capable of detecting CTC, we compare three kinds of RNN: standard RNN, GRU, and LSTM. The architecture of each model is shown in table 6.5. The assessment measures for these three kinds are presented in table 6.6. In terms of accuracy, LSTM networks and GRU networks clearly outperform standard RNNs. Furthermore, LSTM networks had the greatest performance among these models, with the maximum accuracy of 96.2%. Because they can retain more information than conventional RNNs, LSTM networks address the long-term dependence issue. As a result, the LSTM method was selected as the final type for the CTC detection issue in this dissertation.

To investigate the impact of the number of hidden states and layers on the performance of the LSTM model, Figure 6.5 demonstrates that the model performs best when the number of hidden states is 128 with two hidden layers. In general, the model's performance improves as the network's number of hidden layers and states are increase. When these two parameters are excessively high, the LSTM model

Table 6.5: Specifications of different RNN models.

Model	LSTM	GRU	Simple RNN
Number of layers	2	3	3
Activation function	tanh	tanh	tanh
Number of neuron	64	64	64
Dropout probability	0.5	0.5	0.5
Number of neuron of connected layers	128	128	128
Connected and function	Relu	Relu	Relu

Table 6.6: RNN types using sequence length 128 bits.

RNN type	Accuracy(%)	Precision (%)	Recall (%)	F_1 (%)
Standard RNN	91.02	89.01	88.00	88.50
GRU	93.51	91.67	91.51	91.62
LSTM	96.2	94.77	95.41	95.09

may exhibit overfitting. As a consequence of the testing findings, we set the model parameters to acceptable levels.

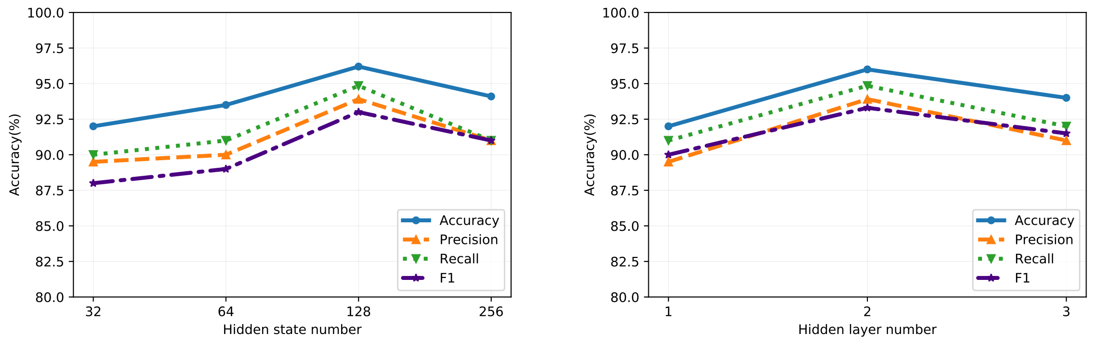


Figure 6.5: LSTM model performance using various hidden state (left) and hidden layer (right).

6.1.6 Convolutional network analysis

To investigate the impact of 1D-CNN hyperparameters on model performance, this part first examines the effect of convolution filter number and breadth, followed by the effect of activation functions. Figure 6.6 (left) shows the effects of applying various filter numbers of 16, 32, 64, and 128. The same Figure 6.6 (right) shows the results of employing various filter widths of 3, 5, 7, 9, and 11. The figure findings show that as the number of filters rises, so does the performance. With 32 filters, the greatest performance is obtained. When the breadth of the filter is more than 5, the performance gradually improves.

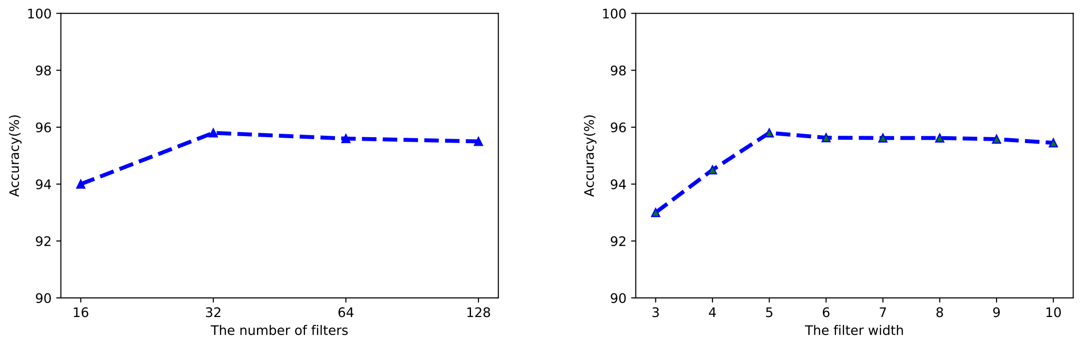


Figure 6.6: Performance of 1D-CNN with (left) different filter number and (right) different filter width.

We performed many tests to compare the results of our 1D-CNN model with the results of three activation functions: Sigmoid, ReLU, and tanh to show the benefits of utilizing 1D-CNN over these activation functions. Using the balanced data, we trained and assessed our best model. Table 6.7 contains the evaluation results. The findings indicate that 1D-CNN performs better in convolutional layers when using the ReLU activation function. We may, for example, obtain the maximum accuracy of 95.78%, precision of 92.67%, recall of 93.01 %, and F1-score of 91.82%. The lowest accuracy was 90.8%, the precision was 91.01%, the recall was 88.00%, and the F1-score was 89.47 %.

Table 6.7: 1D-CNN model performance based on using different functions.

Function	Accuracy	Precision	Recall	F_1
Relu	95.78	92.67	93.01	91.82
Sigmoid	91.02	91.50	90.41	90.94
tanh	90.80	91.01	88.00	89.47

6.1.7 Combination network analysis

To determine which sequence-based detection model performs best in the CTC detection issue, we evaluate the performance of the 1D-CNN, LSTM, and combination networks (CNN-LSTM) and (LSTM-CNN).

The best designs for these models that were utilized in comparison trials are represented by sections 6.1.2, 6.1.1, and 6.1.3. In addition, we compare these deep learning detection models to various conventional machine learning detection models:Naive Bayes (NB), Linear Support Vector Machine (SVM), multi-layer perceptron (MLP), K-Nearest Neighbor (KNN), and Decision Tree (DT) are examples of conventional techniques.

The following are the parameters of the usual methods: The constant parameter was set to 0.01 for NB. The kernel was set to linear for SVM. The loss was squared hinge, and the halting criterion tolerance was set to $1e - 4$. We utilized one hidden layer with 50 units for MLP. The neighbor parameter was set at 3 for KNN. The minimum number of samples in a leaf in DT was set to three.

We use 5-fold cross-validation to evaluate the model's performance. The procedure of cross-validation is also repeated five times. Then the five results were averaged to provide a single estimate.

The accuracy outcomes of models are shown in 6.7. The findings show that deep learning approaches surpass all traditional methods. Furthermore, it was discovered that the combined LSTM-CNN approach outperformed both neural networks and traditional machine learning techniques.

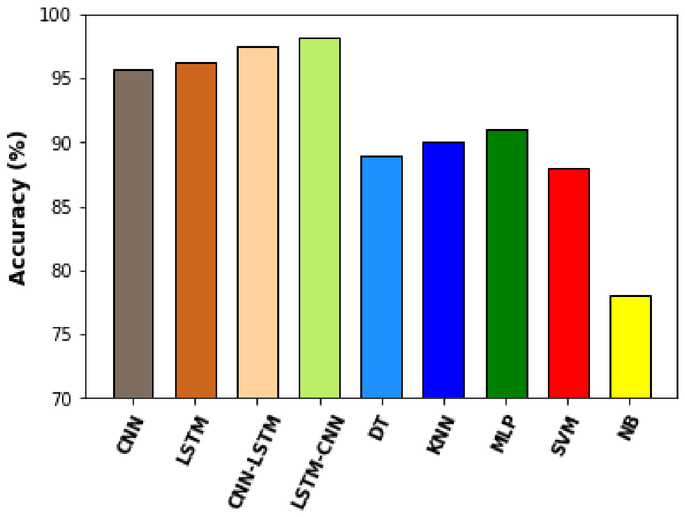


Figure 6.7: Comparison of models performance.

Although LSTM and 1D-CNN were multi-layer architectures at their core, particular topologies emphasized them. When applied to temporal sequences, LSTM outperforms 1D-CNN and other machine learning methods, accurately detecting 96.2% of CTC detection, compared to 95.73% for 1D-CNN. The main difference between CNN and LSTM is the convolutional and pooling layers. It combines input data with the learnt function, in which the LSTM analyzes and links all of the information to produce an output.

The CNN-LSTM model has the second highest CTC detection accuracy 97.01%. In the first layer, the CNN algorithm picks the best characteristics before sending them to the LSTM algorithm to learn the extracted features. It was, however, not as successful as identifying CTCs using the LSTM-CNN model. The LSTM-CNN classifier has the highest accuracy of 97.5% outperforming generic CNN, LSTM, and traditional machine learning methods. This is because the LSTM takes the best feature in the first layer before passing it to CNN to be learned.

Furthermore, Table 6.8 displays additional performance metrics for deep learning and machine learning methods. We also verified that combining the CNN and LSTM models to represent spatial and temporal data yielded superior results. The LSTM-CNN performed the best, with a maximum accuracy value of 98% and recall is 96.8%.

Table 6.8: Performance measures comparison.

Model	Precision (%)	Recall (%)	F_1 (%)
CNN	95.00	90.00	92.43
LSTM	96.10	93.05	94.55
CNN-LSTM	97.00	96.20	96.64
LSTM-CNN	97.5	96.80	97.15
DT	89.00	85.00	86.95
KNN	89.00	79.99	84.25
MLP	90.07	89.00	90.8
SVM	88.01	86.00	87.00
NB	76.70	74.00	75.32

The loss and accuracy per experimental epoch of the LSTM-CNN, 1D-CNN, and LSTM models are shown in Figure 6.8. The greatest accuracy and lowest loss are achieved by LSTM-CNN detection model.

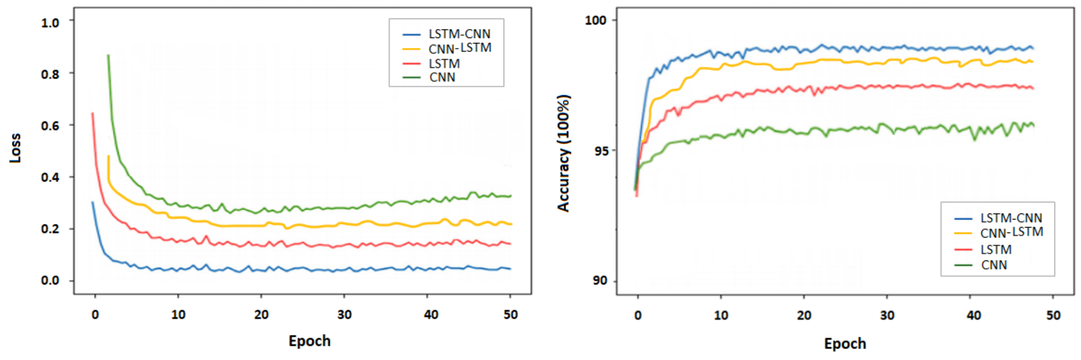


Figure 6.8: Model loss (left) and accuracy (right) per epoch.

The results supported the study's goal of integrating neural network methods to efficiently manage and detect spatial and temporal problems.The order in which the layers were stacked, on the other hand, had a substantial impact on model recognition performance.By stacking the LSTM layer before the CNN layer, more information was gathered from all inputs before the input was sent to the CNN for extraction. If the layer is started using CNN, the sequence information in the inputs can be lost, and the LSTM function is not fully used.

In this part, we also look at how the dropout method affects the small-scale traffic dataset in light of the over-fitting problem. The Figure 6.9 shows the results of different dropout rates computed using LSTM, 1D-CNN, and combination networks. The models using the dropout method, as illustrated in the figure, are capable of inhibiting over-fitting and attaining better performance. A greater rate translates to better performance. Because of its varying dropout rates, LSTM shows the greatest variation of the three methods. In contrast, the CNN and LSTM-CNN methods are unaffected by change in dropout rates. Furthermore, when the rate reaches 0.5, performance gradually increases.

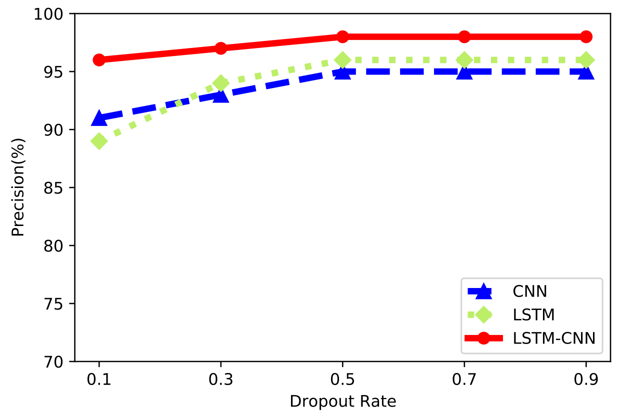


Figure 6.9: Model performance using various dropout rate values.

Moreover, we evaluate the effect of sequence length on detector results. The sequence length parameter defines the largest sequence that LSTM model can evaluate.

Generally, the sequence's length should be similar to the raw data length. Figure 6.10 demonstrates that when the sequence length is 64, the models function best. Additionally, as the length reaches 64 inter-arrival times, performance progressively declines.

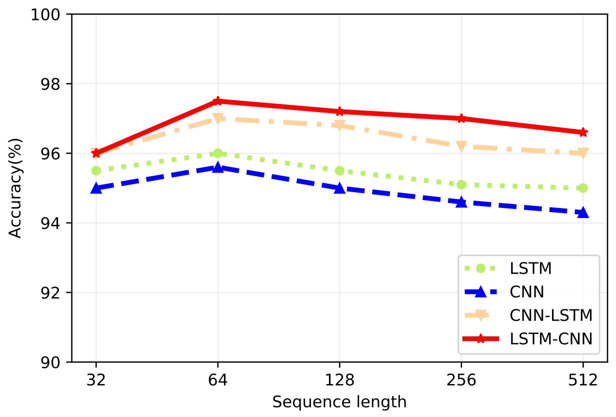


Figure 6.10: Performance sequence based models with different sequence lengths.

6.2 Chapter summary

The proposed sequential-based detection model using LSTM, 1D-CNN, and combination networks was presented in this chapter. Additionally, investigate the effect of model hyperparameters on model detection performance. This chapter demonstrates that the LSTM-CNN model outperforms the CNN, LSTM, and CNN-LSTM models in detecting CTC. This shows that the hybrid neural network approach was more accurate and suitable for recognizing sequential input. This is an intriguing study direction since it will broaden the combination network and assess its effect on CTC identification.

Chapter 7

CTC Detection and Localization Using Autoencoders

As networking technology has improved, the Internet has become more widely applicable. Covert channel threats are becoming more proficient at using the advantages of Internet openness to create covert channel techniques at an alarming pace, even as its global reach expands.

Covert timing channels have grown in breadth and severity, making it more difficult for internet businesses and corporations to remain ahead of hackers. As a result, one of the most promising defensive cybersecurity solutions is the hidden timing channel detection system. Network security has lately attracted a lot of attention from researchers. Even though covert timing channel detection systems have evolved to a high degree of sophistication, today's high network traffic volume and complex network architecture test their detection capabilities. Because of this, it is critical to developing a new method for enhancing the covert timing channel detection based on massive amounts of network data. Technological progress recently has increased the ability to represent features with a large volume of data and has started a revolution in the design of effective attack detection models to achieve high performance levels and protect networks from cyberattacks in the changing threat environments.

The majority of research has been focused on supervised classification tasks, which require varied and suitable labeled data for model training, although these methods have produced amazing results in covert channel detection, as shown in earlier chapters. The process of adding labels to a significant amount of network traffic is taking long time and prone to mistake in the actual network environment. In light of the aforementioned challenges, the covert timing channel detection model using unsupervised deep learning methods is receiving increasing amounts of interest in the scientific community. Without tagged data, these methods may identify covert attack activity in network traffic without any prior knowledge of covert attack methodology.

Various autoencoder models used for attack detection have been thoroughly evaluated in the literature and shown to have various advantages and generalization abilities. As a result, many unsupervised deep learning methods, including autoencoder techniques, have been intensively investigated in the attack detection area. It enables the adoption of data-driven methods for sophisticated feature representation and also demonstrates the ability to reconstruct data.

Choi et al. [73], for example, examined the use of several autoencoder models in the development of intrusion detection frameworks. They used normal and abnormal data to train the autoencoder models and adjusted the reconstruction error threshold to enhance attack accuracy detection. Similarly, Aygun and Yavuz [91] evaluated several autoencoder models with a stochastic approach for reconstruction error threshold in order to develop intrusion detection systems. Nasser et al. [92] created a number of autoencoder models to capture the resilient characteristics of their dataset that may be used to refine and enhance the Intrusion classifier system. Vaiyapuri and Binbusayyis [93] have presented an experimental evaluation of various autoencoder models in order to build an unsupervised intrusion detection system. The authors compared autoencoder models as one-class classifiers in this research to offer actual experiments on how these models may be utilized to build a comprehensive Intrusion detection framework for all types of assaults.

motivated by the work of Vaiyapuri and Binbusayyis [93], and Choi et al. [73] this dissertation fills a knowledge gap by providing the first comprehensive experimental evaluation of CTC detection performance between different autoencoder models in order to develop an unsupervised detection framework for detecting and localizing CTCs. However, since no actual study has been done to assess and contrast autoencoder models as classifiers for unsupervised covert time detection, it is still unknown how they may be utilized as classifiers for unsupervised covert time detection.

This research compares different autoencoder models to identify CTC by training the models on inter-arrival time of normal traffic to determine the reconstruction error threshold and enhance model detection accuracy. Stacked autoencoder (SAE), Sparse Stacked AE (SSAE), Contractive AE (ContAE), Denoise AE (DAE), and Convolutional AE (CAE) will be utilized in this comparison. Choose these techniques because their competitive performance has been proven in the relevant literature, and they are now the most often used domain for attack detection [94–97]. The AE models, when coupled with two types of input data (sequential and image), provide a potentially strong basis for building an effective covert channel detection model, minimizing the need for diverse and adequate concealed network traffic behavior. This research also demonstrates how the AEs models are used to capture robust feature representation for CTC traffic, which may improve the ability of the AE models to be used as multi-class classifiers to point to the location of hidden data in network traffic.

7.1 Autoencoder and reconstruction error for CTC detection

Autoencoder is a one of neural network methods that can compress the input data into lower-dimensional representations and then uses them at its output layer to recreate the original input data [98]. An autoencoder architecture made up by three component: encoder, hidden layer (code), and decoder, as shown in Figure 7.1. The encoder compresses the input data and maps them to hidden layers to create the latent feature representation using an activation function as given below:

$$C = f(WI + b) \tag{7.1}$$

After that, the decoder recreates the input data from the feature representations

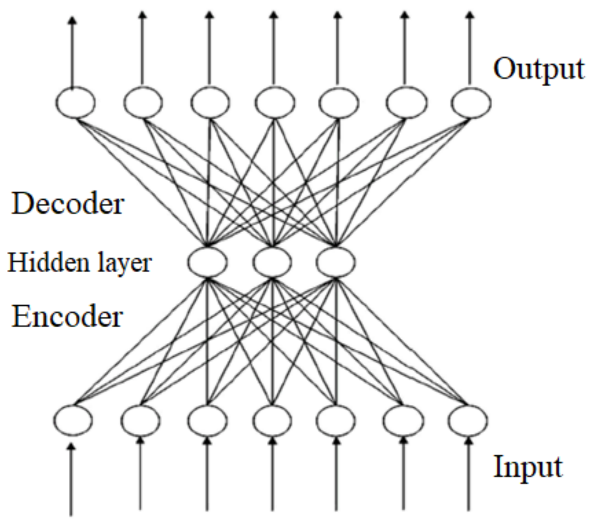


Figure 7.1: General structure of autoencoder.

only using an activation function, as illustrated below:

$$O = f(W'C + b') \tag{7.2}$$

During the training process, the autoencoder learns the parameters of both encoder and decoder networks to minimize the reconstruction error (RE) using the following cost function:

$$J = min\frac{1}{2N}\|I - O\|^2 + \frac{\lambda}{2}(\|W\|^2 + \|W'\|^2) \tag{7.3}$$

The cost function of the autoencoder consists of two parts; the first part represents RE between the input and reconstructing data using data samples. The second part represents the regularization with weight parameter λ that use to help the network avoid overfitting and restrain the weight parameter.

Recently, Vaiyapuri et al. [93], and Sakurada et al. [72] utilized the reconstruction ability of autoencoder to devise approaches for anomaly and malicious detection. The idea behind these approaches is that an autoencoder trained only using overt traffic data will have no ability to reconstruct malicious data samples that it has not seen trained before, and resulting in a high reconstruction error. Therefore, the reconstruction error can be utilized as an indicator for malicious detection and can be used in one-class classification approaches. Autoencoders are commonly used in various domains, such as for anomaly and intrusion detection.

However, covert channel attacks can display different behavior in network traffic, which is hard to collect them in real networking configurations. For this, the application of using autoencoder as one class classifier in the covert timing channel detection plays a major role for modeling the overt network traffic to detect covert channels.

In this dissertation, we aim to assess using different autoencoders as one class classifiers with reconstruction error for CTC detection. Toward this purpose, Algorithm 3 and Figure 7.2 illustrate how the autoencoder will be used to detect any kind of covert channel attacks. The autoencoder detector used a threshold of reconstruction error to identify CTC from overt traffic data and was calculated during the autoencoder training phase by calculating the average of RE total training samples.

Algorithm 3: CTC Detection Autoencoder-Based

```
Input: Test, training sets
Output: Predicted classes
Initialize: The neural network parameters
 Model training
 for $I_{th}$ epoch iterate
     for $J_{th}$ mini-batch iterate
         Extract feature representation C using Eq. 7.1
         Recreate input data O using Eq. 7.2
         Minimize the cost function using Eq. 7.3
         Update model parameters
     end for
 end for
Compute reconstruction error average $\alpha$ threshold using training dataset
 Model testing
 for $i_{th}$ instance in the test dataset
     Calculate reconstruction error (RE) loss
     If RE more than $\alpha$ then data sample is CTC
                      else is overt traffic
 end for
```

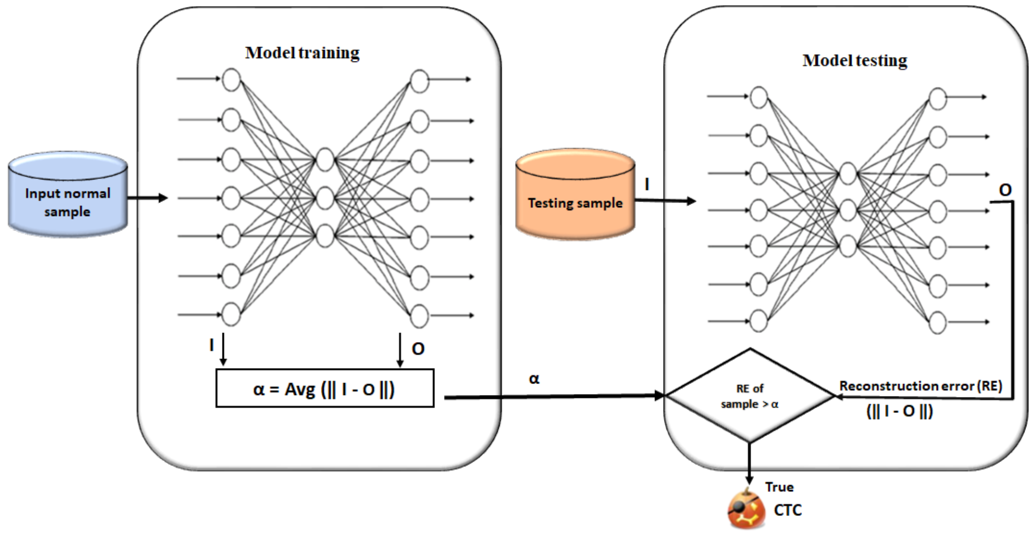


Figure 7.2: Reconstruction error based for CTC classification

7.2 Autoencoder models

This section introducing a brief overview of basic autoencoder network architectures that will be used in this study.

7.2.1 Stack Autoencoder (SAE)

Stacked autoencoder is a common autoencoder architecture constructed by stacking multiple autoencoders such that where the output of the first autoencoder is fed into the next autoencoder as shown in Figure 7.3. Because training all autoencoders at the same time is time-consuming, SAE instead uses greedy layer-wise training the autoencoders in forwarding order.

As a result, training layers in SAE to gradually gain more critical information.

Following the completion of the models training, the encoding layers are concatenated, then followed by the decoding layer for all autoencoder models. As a consequence, SAE is composed of several layers that gather deep representative information from inputs and utilize it to enhance their reconstruction capabilities.

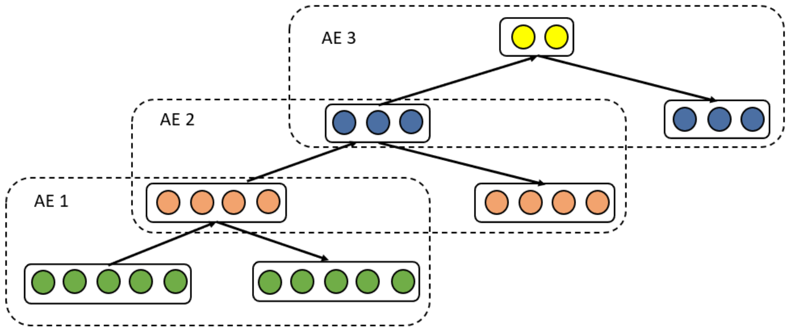


Figure 7.3: Stacked autoencoder structure.

7.2.2 Sparse Stacked Autoencoder (SSAE)

The sparse autoencoder is one of an autoencoder architecture that regularize the autoencoder by introduces a sparse constrain to the basic autoencoder principle to learn and extract sparse feature representations from input data, as shown in Figure 7.4. In particular, the sparse penalty term would be added to the loss function of autoencoder as represented in Eq. 7.4, makes only a proposition of nodes become active with nonzero values in hidden layers and represents each input data as a group of nodes to discover essential sparse features [99].

$$J_{SSAE} = J(\theta) + \alpha \sum_{j=1}^{s} KL(\rho \| \hat{\rho}) \tag{7.4}$$

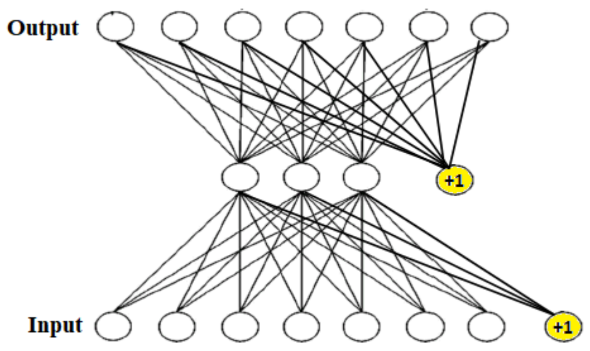


Figure 7.4: Sparse stacked autoencoder structure.

7.2.3 Denoising Autoencoder (DAE)

To learn more general data features from noisy input data and avoiding overfitting problems, DAE was developed [100]. Unlike other autoencder models, the encoding method in DAE is highly effective and efficient in extracting more intelligent features that may negate the impact of corruption for improved data construction, as shown in Figure 7.5. DAE can improve feature extraction, when it capture and prevent statistical relationships between input data and performs consistently, even when the input data is contaminated by noise.

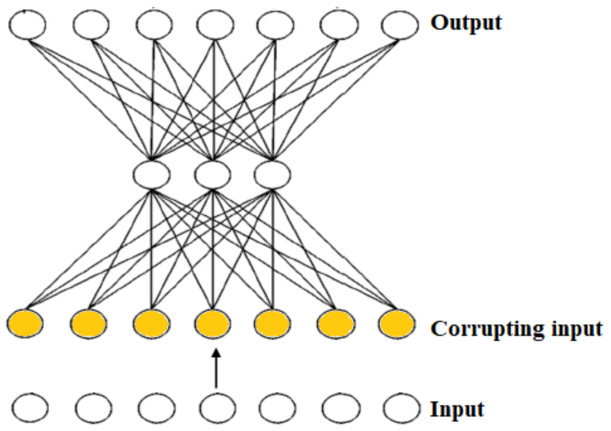


Figure 7.5: Denoising autoencoder structure.

7.2.4 Convolutional Autoencoder (CAE)

CAE [101] is autoencoder architecture that accomplishes excellent feature representation by utilizing the advantages of CNN. CAE uses stochastic gradient descent to learn non-trivial features and discovers effective initializations for CNNs that avoid the many unique local minima of highly objective functions that arise in various deep learning situations. CAE may find localized characteristics that repeat themselves throughout the input and discover strong feature representation by considering the connections between the more suitable features to remove unnecessary features.

In the CAE the weight is sharing among the inputs and guarantees that the feature spatial localization o data is preserved. As a consequence, there are fewer parameters to memorize. Rather than being fully connected layers, CAE is coevolutionary layers as illustrated in Figure 7.6. Taking inspiration by Chen et al. [102] and [92], this research creates 1D and 2D CAE with the premise that utilizing 1D and 2D CAE would allow for greater efficiency with sequential and image network traffic data.

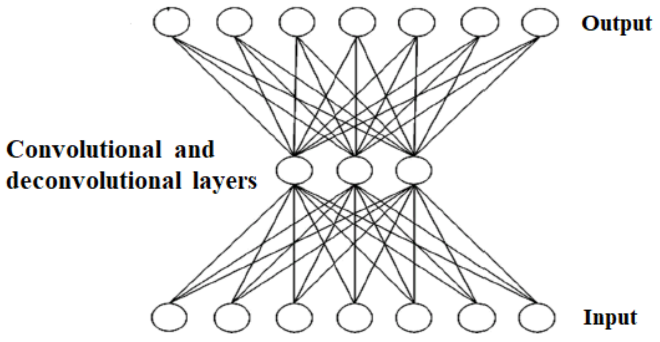


Figure 7.6: Convolutional autoencoder structure.

7.2.5 Contractive Autoencoder (ContAE)

Contractive autoencoders is as a suitable neural network follows DAE that have a strong ability of learning robustness features against the effect of perturbation noise input data [103] . This resilience to minor changes is accomplished by applying a penalty term for encoder activation with regard to all the hidden representation partial derivatives of input data based on the Frobenius norm of the Jacobian matrix, which is expressed as follows,

$$\|J_f(i)\|_F^2 = \sum\nolimits_{xy} \left(\frac{\partial h_y(i)}{\partial i_x} \right)^2 \tag{7.5}$$

The term penalizing refers to the fact that the learned characteristics are not impacted by minor modifications in inputs since they are locally invariant in nature.

With a penalizing term, the cost function of ContAE looks like this:

$$J_{ContAE(\theta)} = J_{AE}(\theta) + \|j_f(i)\|_F^2 \tag{7.6}$$

The penalty terms help in learning and obtain lower dimensional feature space that are better representation of local directions of data variation. Although DAE and ContAE have the same goal of improving the robustness of basic autoencoder, the approach they take is different. DAE, for example, accomplishes stochastic by contaminating the input data with random noise values. As shown in Figure7.7, the ContAE may be calculated analytically by balancing the reconstruction error against the contractual penalty.

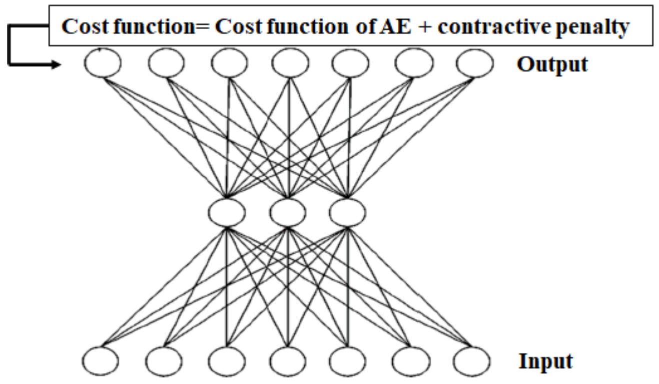


Figure 7.7: Contractive autoencoder structure.

7.3 Experimental setup for autoencoder model design and comparison

This section explains the experimental and framework used for comparative assessment and evaluate all autoencoder models utilizing sequence-based and image-based input data for CTC detection.

7.3.1 Autoencoder network structure and parameters

A network architecture that produced acceptable results with all of the selected autoencoder models is determined via a series of exploratory tests to perform a fair and meaningful comparison. In the determination procedure, the reconstruction error was utilized as an indicator for having optimum performing network design, as shown in Table 7.1.

The model network design is made up of two hidden layers with dimensions of 64 and 48, respectively, and a bottleneck layer with a size of 32. Furthermore, tanh is utilized as an function over all layers for sequence based data set. SSAE is created from the network design by adding sparse penalty on layers to get the unique statistical feature are retrieved from the input data. In the preprocessing phase, 11% Gaussian noise is added to input data to corrupted data in DAE. The DAE's performance degrades when noise level rises. This is because when the noise levels increase, critical information is lost, making reconstruction difficult and classification performance poor. The ContAE is created by adding the penalty value to the autoencoder cost function.

Table 7.1: Autoencoder structure parameters.

Parameter	Value
Input dimension for sequence	(64,1)
Input dimension for image	(23x32)
Hidden layer	2
Nodes of first and second layer	64 and 48
Gaussian noise corruption factor in	11%
Function	Sigmoid and $Tanh$
Sparsity penalty	.0001
Bottleneck sparsity penalty	.001
Node in bottleneck layer	32
Contractive penalty	.0001

In autoencoder models, the hidden and bottleneck layers in the autoencoder architecture are changed with convolutional layers to create the CAE model, as shown in Table 7.2. By removing duplicate and unnecessary characteristics, the pyramid design lowers the number of trainable parameters and allows learning the important aspects from input network traffic. A max-pooling layer with size of two is used to reduce the dimensional of data.

Figure 7.8 depicts and discusses the architecture of the implemented 2D-CAE. As shown in the figure, the input data is a 32x32 colored image. CAE arrangement of input as a 2D matrix and discover the localized relationships between these data features. CAE is distinct from other algorithms in maintaining the spatial locality by sharing the model weights across all input locations. CAE is more sensitive to feature transitive relationships and aid in learning high-level relationships of global features that can be ignore by other classifiers.

Table 7.2: 1D-CAE parameters for sequential data.

Layer	Input	Output
Input data	(64,1)	(64,1)
1D-Conv	(64,1)	(64,8)
Batch Normalization	(64,8)	(64,8)
1D-Max pooling	(64,8)	(64,8)
1D-Conv	(32,8)	(32,8)
Batch Normalization	(64,8)	(64,8)
Up-sampling1D	(64,8)	(64,8)
Conv1D	(64,8)	(64,8)
Batch Normalization	(64,8)	(64,8)

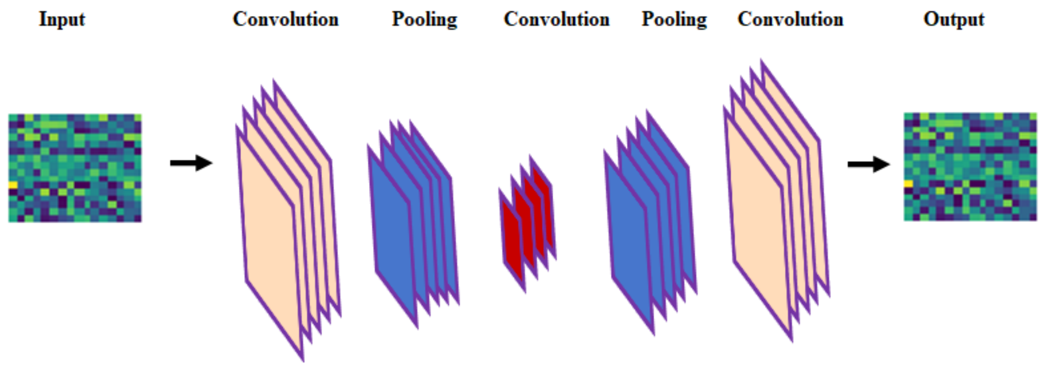


Figure 7.8: 2D CAE for image input data.

Finally, to assess and compare the model's performance, Figure 7.9 depicts the experimental frameworks developed for this comparative research for two types of input data: sequence-based and image-based. The frameworks are comprised of three stage: 1) preprocessing the data; 2) train all autoencoder models using overt network traffic data. As an assessment procedure, to avoid overfitting, all models use a 5-fold cross-validation method; 3) after the training the models, testing the models using test dataset, and then the model performance is compared using a set of metrics.

All of the autoencoder models are trained and optimize their cost function using

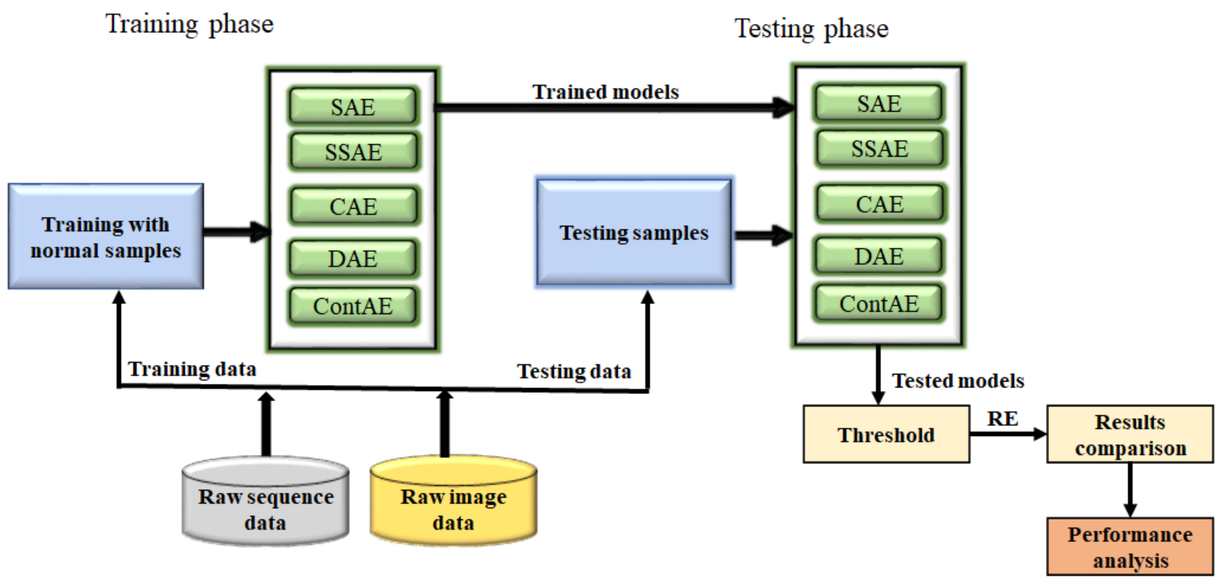


Figure 7.9: Comparison experiment framework.

Adam optimizer due to its flexibility and computing efficiency [104]. Furthermore, to provide a fair comparison of all suggested AE models, it is trained for 14 epochs, 0.001 learning rate, and 128 batch-size.

7.3.2 Experimental datasets

We utilized a dataset recently generated via our covert channel, which is a mix of overt and covert network traffic flow. This dataset contains one kind of covert attack: Cautious Covert Timing Channels (CCTC). CCTC is a more sophisticated hack that attempts to mimic overt traffic behavior to some extent. This kind of covert channel behaves similarly to overt traffic, with packet inter-arrival times that are comparable to overt traffic.

We created two kinds of datasets in our work: sequence-based and image-based. The Cautious Covert Timing Channels (CCTC) attack is included in this dataset. We utilized network flows derived from our covert channel suggested in Chapter 3 for sequence data-based.

This research took into account a flow of 64 inter-arrival time. In each flow, the inter-arrival times of packets were extracted and labeled as covert and overt. Finally, we constructed our dataset using these inter-arrival time sequences. The collection contains 8000 flows, 3000 of which include covert data and 5000 of which contain overt data. In addition, we generated a second version of the sequence-based dataset for localization purposes, based on 64 bits of covert messages inserted at three points in the traffic flow: the beginning, middle, and finish. This version comprises 6000 flows, each with a series of 64 inter-arrival time values.

In the second form of our dataset, we utilized inter-arrival time records shaped as 32x32 colorful images for the image-based dataset. We generated two versions of the image-based dataset. The first version is for channel detection, and it includes 6000 for both overt and covert communication. The second kind is for channel localization, which includes 6000 for covert traffic, which was gathered based on 64 bits of covert messages inserted in three places in the traffic flow: beginning, middle, and finish. The specifics of each dataset are given in Tables 7.3 and 7.4.

Table 7.3: Detection dataset parameters

Covert message size	64 bits
Delay time	0.5μ
The mean IATs of overt traffic (μ)	0.0664 seconds
Sub-flow size	1024 inter-arrivals
Number of images (all datasets)	8000
Number of covert images (all datasets)	3000
Number of overt images (all datasets)	5000
Image size	32x32

Table 7.4: Localization dataset parameters

Covert message size	64 bits
Delay time	$.5\mu$
The mean IATs of overt traffic (μ)	0.0664 seconds
Sub-flow size	1024 inter-arrivals
Image size	32×32
Number of images (all datasets)	6000
Number of images per dataset version	2000
Location of covert message	Beginning, middle,end

7.4 Experimental results and analysis

This section analyzes and contrasts the efficacy and efficiency of each autoencoder model, taking three viewpoints into account. To begin, examine the models convergence ability to represent the design choice of the contrasted autoencoders and show its generalization capacity to identify CTC assaults. In the second stage, look at the ability of the models in CTC detection to offer any insights into the practical applicability of various models. Finally, investigate the performance of the proposed AE models for unbalanced classification to show the AE models' stability in the face of imbalanced datasets.

7.4.1 Network convergence of autoencoder models

For comparing all autoencoder models, the training and testing data from each dataset (image-based and sequential-based) are input into all of models and 5-fold cross-validation is used to evaluate the generalization and converge on two datasets, sequence-based and image-based, separately. This performance analysis is carried out from two perspectives: reconstruction error ability and model learning behavior, as shown below.

7.4.1.1 Reconstruction error ability

To validate each autoencoder model quantitative of reconstruction error ability, the average of reconstruction error provided by each model throughout the training phase is calculated for both the sequence-based and image-based datasets, and the results are presented in Table 7.5. On both datasets, the results show that all of the models have slightly superior ability of reconstruction than the DAE model.

CAE had the best results on both datasets comparing of all other models. The explanation may be related to the network configuration used by CAE finds the most essential feature representations that enhance its reconstruction data ability; this model showed steady and high ability of model converges and generalization throughout the training phase.

During training phase, SAE and SSAE setups showed consistent convergence and generalization capabilities. Furthermore, the training procedure of the ContAE model learns more robust features for data reconstruction and outperforms the SAE and SSAE models in both datasets. The contractive penalty in cost function of ContAE model increase the guarantees that the more generic feature for data reconstructions are captured.

Table 7.5: Average reconstruction error foe autoencoder models.

Reconstruction error	Sequence-based	Image-based
CAE	0.0011	0.0005
SAE	0.0022	0.0014
SSAE	0.0024	0.0016
ContAE	0.0021	0.0013
DAE	0.003	0.0011

7.4.1.2 Model learning behavior

Figures 7.10 and 7.11 show the loss of training and validation data for all models using both sequence-based and image-based datasets for analyzing the learning behavior of models. It is clear from observing the figures the training loss of all models show a fast decrease during the first epochs and quickly convergence. All models show high learning behavior rate, avoiding overfitting problem, and improve generalization as shown by the validation loss curves. According to these results, the optimal network configurations utilized all models are correct, and they can reliably detect CTCs in the validation (testing) set.

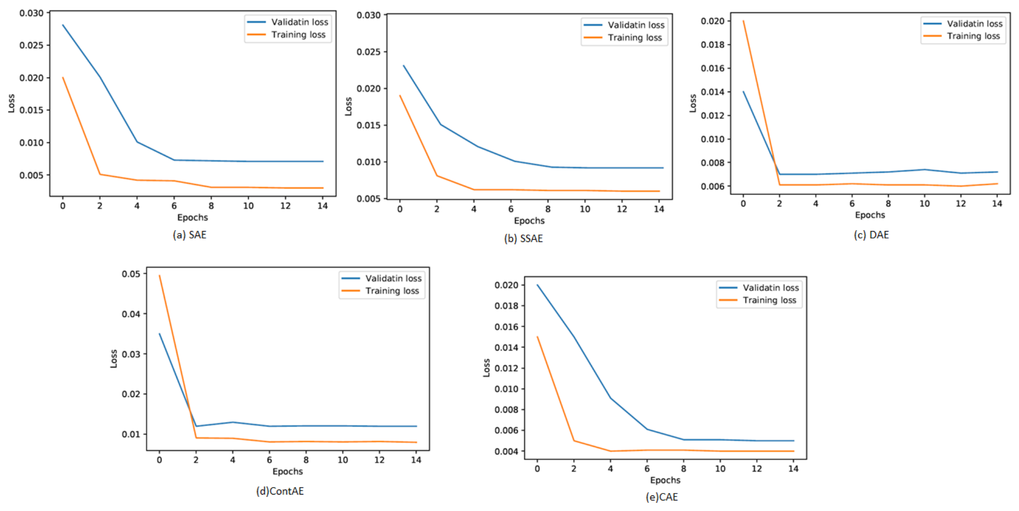


Figure 7.10: Model loss using sequence-based datasets.

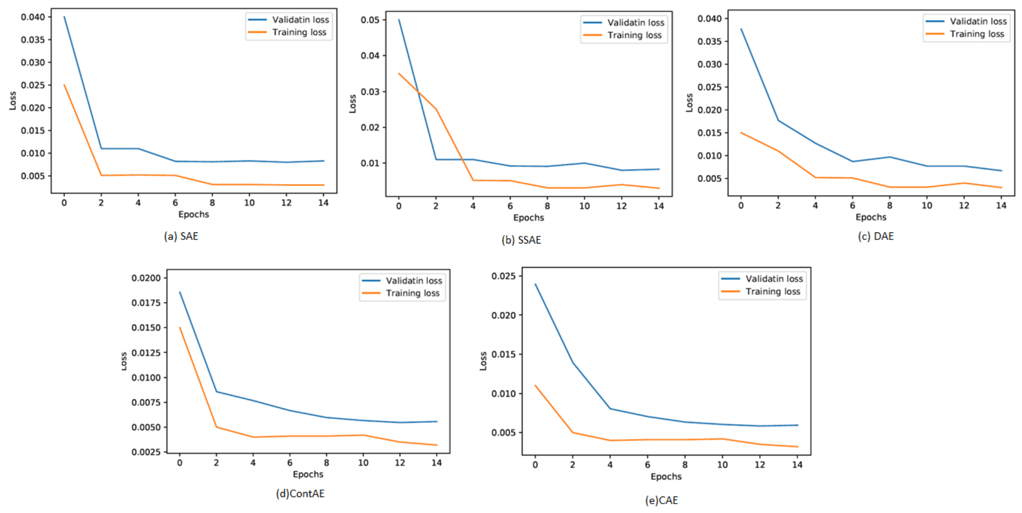


Figure 7.11: Model loss using image-based datasets.

7.4.2 Model performance based on CTC detection

The performance of models in detecting CTCs is compared in this section based on two perspectives: quantitative analysis and ROC curve analysis. In this comparison the average of reconstruction error is utilized by models to classify network traffic data as either covert or overt using both datasets.

7.4.2.1 Model detection performance analysis

To compare the model performance based on detecting CTC, the performance measures are calculated for all models utilizing both datasets and the result reported in Table 7.6. According to the findings, the CAE model outperforms all of the evaluated measures. On two datasets, the CAE model achieves the greatest accuracy as well as the highest average performance in recall and FAR. There is further evidence of DAE's uneven performance across datasets. On the image-based dataset, for example, the DAE models perform relatively better. However, it performs the worst on the sequence-based dataset. On the contrary, we can see that SAE and SSAE models perform consistently well on average across both datasets.

Table 7.6: Model detection performance

	Sequence dataset			Image dataset		
AE models	Accuracy	Recall	FAR	Accuracy	Recall	FAR
SAE	89.61	87.26	17.04	91.23	88.13	14.62
SSAE	89.86	88.34	18.81	91.36	86.02	15.50
CAE	91.72	90.00	13.04	93.14	91.23	12.20
ContAE	90.45	80.37	13.00	91.07	90.42	14.00
DAE	88.50	85.98	15.13	90.92	88.34	13.25

7.4.2.2 ROC model analysis

Receiver Operating Characteristic (ROC) curve is considered as an essential measure for displaying an attack detection model's detection performance. It also allows for the comparison of various detection models in terms of relative significance for both recall and precision measures, which are considered as important criteria for practical idle models. As a consequence, the ROC curve for each autoencoder model was built using the average reconstruction shown by each model throughout the training process. Figure 7.12 shows the models' ROC curves based on the two datasets: sequence-based and image-based. It is clear from both datasets that all of the models have ROC curves close the upper-left corner and indicate to the high classification ability of models.

The curves of ROC for both datasets (image and sequence based) show an intriguing that all ROC curves on image-based datasets are closer to the upper-left corner than on sequence-based datasets. This result may show that the way models describe traffic time samples in datasets affects their performance. As a result, our original debate that visualization traffic time as images is adequate and may enhance detection model performance is confirmed. It also highlights the possibility of improving the detection performance of models in network settings by using more overt traffic samples.

The area under of ROC, reveals models' generalization capacity to detect new attacks, as shown in Figure 7.12 for both datasets. With an AUC value of 92.0% and 90.1%, the CAE model consistently outperforms on images dataset and sequence-based dataset, respectively. It is also worth noting that the denoising training method allowed model to get more robust feature representations for classification and outperform both of SAE and SSAE on the image-based dataset, with AUC values of 90.9%. However, when utilizing sequence-based input data, DAE performs worse, with AUC values of 82.0%.

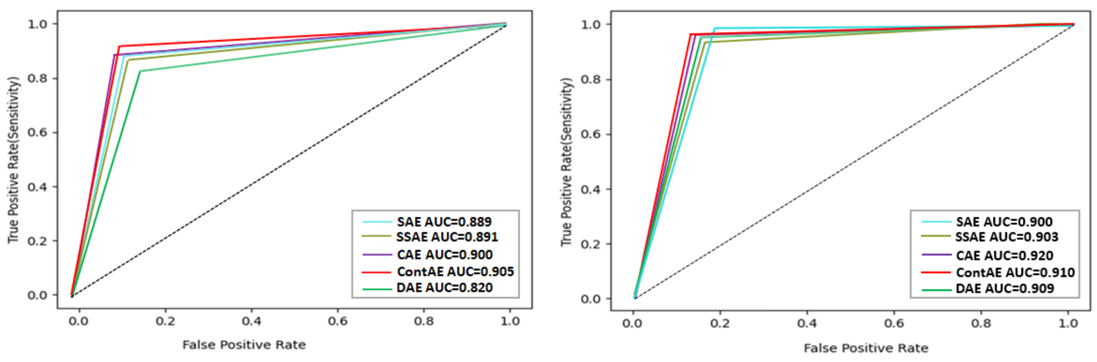


Figure 7.12: Model ROC curves using sequence-based dataset (left) and an image-based dataset (right).

7.4.3 Model performance using imbalanced classification

Regarding covert channel behavior that utilizes normal traffic to leak covert data, in real application the traffic data may be unbalanced with a limit number of covert attack and a high number of normal traffic. The existence of such conditions necessitates a comparison of the model performance for unbalanced binary classification.

The measures were used to evaluate detection performance previously such as ROC may don't have ability to represent the model performance under using unbalanced training data. For example, ROC curve measure is presentable measures for evaluating the performance of CTC attack detection. However, using this measure with imbalance data may mislead by providing an optimistic result for overt traffic samples while failing to account for model accuracy.

As a result, the experimental comparison on imbalanced data analysis uses the most commonly used metrics such as precision-recall (PR) curve and F_1-score to offer a comparison of autoencoder models with unbalanced traffic datasets as bellow.

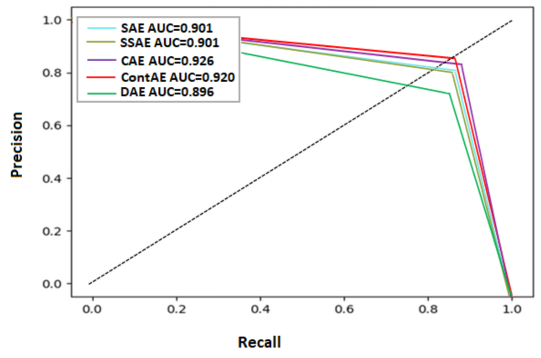
- **Precision and recall curve analysis**

A precision-recall (PR) curve depicts the relative trade-off between precision and recall. A PR curve is recommended in recent research for two reasons: it allows for the calculation of the percentage of correctness when a dataset is being evaluated for binary classification. It also evaluates various classifiers to find the one that optimizes CTC detection precision while maintaining adequate recall. To demonstrate its benefits, Figure 7.13 compares PR curves for models on sequence-based and image-based datasets.

The PR curves for a perfect discrimination CTC detection model pass through a location in the upper-right corner. When examining these images from this perspective, it is possible to see that the PR curves of all the models on both sequence-based and image-based datasets pass near the upper-right corner. Looking closely at the curves on the sequence-based dataset, it can be seen that all of the models, except for DAE, are near the upper-right corner.

When compared to other models on the image-based dataset, DAE exhibits exceptions with penalized performance. Furthermore, it is worth noting that the generated PR curve findings accord with the ROC curve results. Based on this observation, we can conclude that all of the autoencoder models that can be trained only with overt traffic data are reliable detection performance against using imbalanced datasets, implying the proposed autoencoder models can be used as a better detection solution as a one class classifier for building CTC detection models to improve detection ability.

The resulting area under the PR curve in Figure 7.13 shows that all of the models are similarly efficient in precision and recall terms on both datasets. In contrast to other models, CAE obtains the best results on both datasets of (0.926, 0.934). On the other hand, it can be observed that DAE outperforms on an image-based dataset but under-performs on a sequence-based dataset, giving an AUPR value of 0.919. Overall, the AUPR findings show that all of the investigated AE variants are extremely effective at improving CTC detection performance against highly unbalanced datasets.



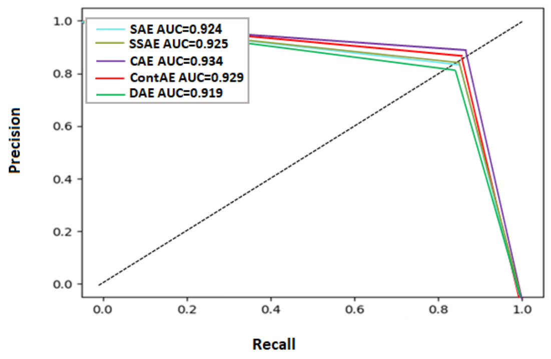


Figure 7.13: Model PR curves using sequence-based dataset (left) and image-based dataset (right).

- **F_1 score analysis**

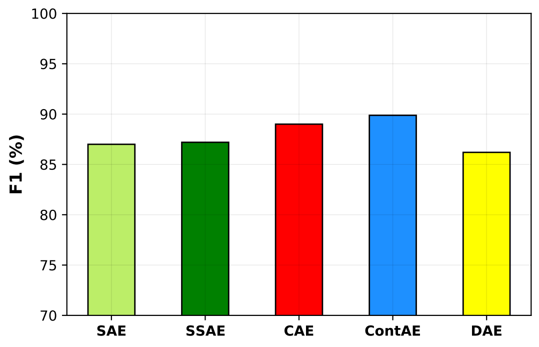
The F_1 score is an important performance measure for highlighting the detection model's effectiveness in terms of recall and accuracy using a factor that regulates their respective importance [105]. According to the most current research, the F_1 score may assist other metrics by preferring the accurate categorization of traffic attack samples while avoiding the misclassification of overt traffic flow data.

To get a better understanding of the discriminating ability of the models, the F_1 is examined, and the findings are shown in Figure 7.14. The resulting $F1$ complements the previously calculated metrics and supports the importance of all models in attaining improved CTC detection performance.

It is worth noting that, despite DAE having a lower F_1 of 87% when compared to other autoencoder models on the sequence-based dataset, it still has a higher score of 90.69 % on an image-based dataset. The three investigated models, SAE, SSAE, and ContAE, similarly show greater F_1 two dataset gains. On image input data, all of models provide the best F_1.

7.5 Autoencoder for pinpointing CTC

As previously mentioned, identifying hidden communications is crucial if you want to prevent malicious software from stealing your sensitive network data. When a covert message is discovered, the transmission is halted to prevent the malicious data from being exfiltrated. When defending against covert channels, deleting traffic



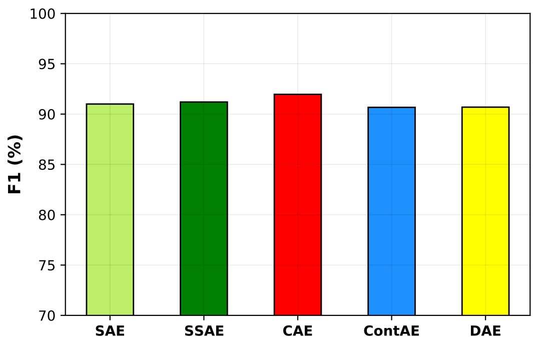


Figure 7.14: Model F_1 using sequence-based dataset (left) and image-based dataset (right).

flows containing covert communications is an effective cyber defense; nevertheless, it disrupts QoS for overt traffic from valid applications, which may include the majority of packets in the discarded flows.

It's critical to locate the secret message's hiding place. If only harmful traffic is being dropped, the remainder may go through. Non-malicious apps no longer cause as many interruptions thanks to our fine-grained identification.

To test how well autoencoder models work in locating hidden messages inside traffic flows, we ran various tests separately. This was accomplished by designing an experiment in which secret messages were inserted into all communication flows in one of three points: the beginning, middle, and end. After that, we looked at the performance of the models to determine the right traffic flow segment containing the hidden messages, using two datasets: one based on sequence and the other on images. In these experiments, the models use as multi-class classifiers. The location of covert data can be classified by extracting feature information from the hidden layers of models and fed them to the classifier for CTC localization.

Table 7.7 presents the results of our tests, which show the average accuracy in directing the covert traffic for each of the three classes. We can see from the findings that models do a good job of locating hidden traffic on sequence datasets. Sequential data is a challenge for autoencoders to detect patterns in. Image-based values are more accurate, although models increase in performance, like CAE.

As a result, utilizing images to locate covert traffic would be more accurate and efficient than employing autoencoder models with time sequences. Since the network architecture used by all models was not optimal for detecting covert traffic locations, a more complicated network model was required to capture and extract essential dataset features, such as the time sequence dataset, which contains features with complex relationships.

Table 7.7: Model accuracy for CTC localization.

	Accuracy	
Model	Sequence-based	Image-based
SAE	84.14	88.54
SSAE	85.32	87.23
CAE	88.45	91.14
ContAE	84.2	88.02
DAE	83.00	86.06

7.6 Chapter summary

To propose unsupervised CTC detection systems utilizing two kinds of data sequentially and image-based, this chapter offers an overview of the autoencoder models used, including SAE, DAE, SSAE, ContAE, and CAE. Each model's ability to accurately classify binary data is assessed using a variety of metrics, including the precision-recall curve and the F_1 value.

The overview of the findings of the study of these measures is given to shedding light on the detection potential of various autoencoders. For instance, the CAE model's behavior was consistent across datasets with varying characteristics, and it provided the greatest detection performance. The DAE model performed well on the image-based dataset, but it was unable to provide consistent results on the sequential-based one. Since even a little amount of noise may damage the practical information needed for the reconstruction of the data, this discrepancy in DAE variant performance across datasets is possible. Because of this, DAE is not the superior option for a dataset of this kind In comparison to the SAE and SSAE models could not offer similar results; nevertheless, across different datasets, these models demonstrated an average dependable performance across all performance metrics. Image-based data with autoencoder CTC detection and localization models are more accurate than sequential time series with the same models utilizing image-based data would be. Consequently, this chapter may be used to kick-start further work on improving unsupervised AE detection models with high and reliable capacities to detect CTC assaults.

Chapter 8

Conclusion and Future Work

The last chapter reviews the dissertation's contributions and suggests various possible expansions to our work.

8.1 Conclusions

Finding and using common resources that are unlikely can be used as a communication channel is a crucial step in developing a covert timing channel. This dissertation showed the covert attacks use inter-arrival times for leaking data is considered a real threat to network security. For example, a covert channel limited rate is represented as complex channels indistinguishable from normal channels.

To support this assertion, we asserted that after a shared resource has been detected, covert traffic can often be distinguished from normal traffic based on altering event timing to encode the covert information. This work described the design and building of a new covert channel (BCTC) that conceals channel activity by using a carefully planned sequence of timing intervals. BCTC employs different delay times to represent several types of covert channel attacks, such as greedy and cautious. They utilized particular threshold values to map packet time of binary symbols to different inter-arrival times.

According to our results, a motivated user may alter the time of the message encoder in covert timing channels to construct a channel that may go unnoticed by distribution analysis. Furthermore, it is possible to intelligently alter packet timings to remove links between timing values and symbols and choose the best time to hide covert packets. Using this method, the concealed timing channels would go unnoticed.

In the second part of the dissertation, we investigated detection approaches based on image processing, machine learning, and deep learning methodologies using two types of datasets (sequential-based and image-based) to detect covert timing channels and warn of their existence so that covert mitigation defense mechanisms might be started.

In addition, we investigated effective search techniques for identifying traffic regions that seemed to be covert traffic. Consequently, the disruption of normal traffic caused by non-malicious applications is significantly minimized as a consequence of this precise localization.

Finally, the following are the dissertation summary's contributions:

- We developed covert timing channels, which are particularly intended to conceal covert traffic by changing packet transmitting timing to generate inter-arrival time sequences that can be decoded to restore the original covert data.

- We examined the effectiveness of the covert timing channel based on the bit accuracy and transmission bit rate. And found that network conditions and time delays are played an important role that affects channel performance.

- We discovered the threshold of packet timing delays that allows the covert channel to send information efficiently. This threshold is useful in various situations, such as evaluating how hazardous this channel is and how it should be mitigated or how dependable this channel is for data transmission.

- To deal with CTCs, we discussed how the existing covert channel detection methods need to be changed to identify and alert the existing CTC correctly.

- We developed a method for converting a sequence of traffic inter-arrival times into two-dimensional colored images to detect CTC into an image classification issue. As a result, time-series data characteristics may be recognized in two-dimensional images using various features such as color, points, and lines at the appropriate places in the image.

- Based on image processing and machine learning methods, we developed a CTC detection model. The experimental findings of utilizing this method show that variant covert timing channel detection outperforms current approaches significantly.

- We developed a method for locating traffic areas that seem to be covert traffic.

This enables our method to detect the presence of covert channels and initiates covert mitigation defense mechanisms or traffic blocking applications, substantially improving the quality of service compromised when the whole network flow is terminated.

- Using a limited dataset, we assessed the ability to use the CNN algorithm to build a new model for traffic classification and CTC detection. Furthermore, we developed a network architecture that is more dependable for processing and categorizing traffic images and evaluating the appropriateness of CNN for CTC detection. The integration of high-level features acquired by a CNN hierarchical framework is very efficient for CTC detection and classification.

- We evaluated the most pre-trained CNN models, AlexNet, VGG16, and ResNet, as well as other traditional machine learning techniques, with the proposed shallow CNN detection model, taking into account different circumstances related to the availability of training data.

- We developed a CTC detector that fed a raw data sequence of traffic inter-arrival times into an LSTM recurrent neural network, which learned temporal feature representations from the original data without the need of manual feature engineering. The proposed model can recognize a certain sequence by computing the relationship between the current and previous states and directly learning features from raw data that match their high detection capability.

- We developed an efficient and flexible CTC detection model using a one-dimensional convolutional neural network (1D-CNN). Meanwhile, 1D-CNN is used as a supervised learning technique for time series data. Using the 1D-CNN model, we extracted special features from time-series data and helped in establishing how temporal modeling with spatial properties affects detection model performance. Based on the model findings, we demonstrated that the self-learning 1D-CNN model might be utilized as a reliable solution for the CTC detection problem.

- We proposed a new method for CTC detection based on a combination of LSTM and 1D-CNN deep learning models. Multiple information scales were integrated into this approach to see whether additional improvements could be achieved to create a viable CTC detection problem. We created two hybrid network models, CNN-LSTM and LSTM-CNN, in particular. We tested them using a sequence traffic inter-arrival time dataset to see how the order of network layers impacts the effectiveness of the hybrid detection model.

- We filled a knowledge gap by performing the first study to evaluate the potential of different autoencoder models for CTC detection. This dissertation addressed that gap by comparing five different autoencoders for building an unsupervised CTC detection model. The comparative evaluation provides an overview of how different autoencoders trained using the normal traffic data build effective unsupervised CTC detection methods. And use a multi-label class with autoencoder for CTC localization. The comparative findings in this comparative will be interesting to the network security research because it offers a potential route for creating an efficient covert channel detection model based on deep learning methods, thus reducing the requirement for sufficient and varied cover channel traffic behavior.

8.2 Future directions

Future possibilities for expanding on this work into two paths: developing more effective covert timing channels and improving detection methods against these channels, which can be stated as follows:

- One option for designing more simple and covert channels is to use methods that may improve channel accuracy. For example, self-synchronizing codes and phase-locked loops are sophisticated methods for overcoming synchronization problems induced by network conditions. The former approach uses encoding methods that are particularly intended to identify the loss of transmission failure and recover from it by re-connection. The latter approach uses a closed feedback loop circuit to monitor the output frequency of the output signal with an input signal. Both techniques are successful in communications and may improve accuracy and efficiency in a new covert channel.

  Furthermore, faster packet generators may be implemented with the covert channel, allowing the sender to utilize sequences of inter-arrival time with lower timing intervals. Moreover, this work may be extended by using the same channel design concepts to create additional time event covert channels that utilize other events to convey information. For example, the sender may use CPU requests' inter-arrival timings to construct a basic covert channel and test its effectiveness using methods similar to those described in this research.

There are many methods to broaden our research to improve the detection model against covert timing channels, which are briefly discussed below:

- In this research, just one convolution neural network was used for training data. Another framework that might be constructed is one that utilizes parallel convolution neural networks for each time series data and merges them in the final layer for prediction. It would be interesting to see whether the parallel network improves model accuracy and performance.
- Present a new framework for encoding traffic inter-arrival times as various kinds of images such as Gramian Angular Fields (GAF) and Markov Transition Fields (MTF), then use deep Tiled Convolutional Neural Networks to extract high-level features from these images and classify them.
- The present architecture was used in binary classification to identify covert channels. Multi-class classification may be used to evaluate the proposed framework's ability to categorize various kinds of covert timing channel attacks.
- Present a new framework for converting traffic inter-arrival times to 2D Recurrence Plots (RP) images. The time-series image representation introduces additional feature types that are not available in one dimension, and therefore time series classification may be seen as a texture image identification issue. The CNN model may also use these input images to learn different feature representations automatically for classifying them.

Bibliography

[1] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito, "When Privacy meets Security: Leveraging personal information for password cracking," *arXiv preprint arXiv:1304.6584*, 2013.

[2] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: understanding passwords of chinese web users," in *28th Security Symposium ({USENIX} Security 19)*, 2019, pp. 1537–1555.

[3] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, 2016, pp. 21–26.

[4] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[5] A. Mileva, A. Velinov, and D. Stojanov, "New covert channels in internet of things," 2018.

[6] Y.-a. Tan, X. Zhang, K. Sharif, C. Liang, Q. Zhang, and Y. Li, "Covert timing channels for iot over mobile networks," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 38–44, 2018.

[7] K. Maney, "Bin laden's messages could be hiding in plain sight," *USA Today*, vol. 19, 2001.

[8] J. Tian, G. Xiong, Z. Li, and G. Gou, "A survey of key technologies for constructing network covert channel," *Security and Communication Networks*, vol. 2020, 2020.

[9] P. L. Shrestha, M. Hempel, F. Rezaei, and H. Sharif, "A support vector machine-based framework for detection of covert timing channels," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 274–283, 2015.

[10] P. Derbeko, Y. Manusov, N. Eskira, and S. Faibish, "Covert storage channel communication between computer security agent and security system," Aug. 7 2018, uS Patent 10,044,744.

[11] J. C. Wray, "An analysis of covert timing channels," *Journal of Computer Security*, vol. 1, no. 3-4, pp. 219–232, 1992.

[12] N. Kiyavash, F. Koushanfar, T. P. Coleman, and M. Rodrigues, "A timing channel spyware for the csma/ca protocol," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 477–487, 2013.

[13] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia, "Model-based covert timing channels: Automated modeling and evasion," in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2008, pp. 211–230.

[14] D. C. Latham, “Department of defense trusted computer system evaluation criteria,” *Department of Defense*, 1986.

[15] V. D. Gligor, *A guide to understanding covert channel analysis of trusted systems.* National Computer Security Center, 1994, vol. 30.

[16] A. K. Biswas, D. Ghosal, and S. Nagaraja, “A survey of timing channels and countermeasures,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1–39, 2017.

[17] D. Mellado, E. Fernández-Medina, and M. Piattini, “A common criteria based security requirements engineering process for the development of secure information systems,” *Computer standards & interfaces*, vol. 29, no. 2, pp. 244–253, 2007.

[18] P. R. Gallagher Jr, “A guide to understanding covert channel analysis of trusted systems provides a set of good,” 1993.

[19] 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.* IEEE, 1994, pp. 235–243.

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

[21] K. Borders and A. Prakash, “Web tap: detecting covert web traffic,” in *Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 110–120.

[22] S. Cabuk, "Network covert channels: Design, analysis, detection, and elimination," Ph.D. dissertation, Purdue University, 2006.

[23] S. Gianvecchio and H. Wang, "An entropy-based approach to detecting covert timing channels," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 785–797, 2010.

[24] R. Archibald and D. Ghosal, "A comparative analysis of detection metrics for covert timing channels," *Computers & security*, vol. 45, pp. 284–292, 2014.

[25] P. L. Shrestha, M. Hempel, F. Rezaei, and H. Sharif, "Leveraging statistical feature points for generalized detection of covert timing channels," in *2014 IEEE Military Communications Conference*. IEEE, 2014, pp. 7–11.

[26] F. Iglesias and T. Zseby, "Are network covert timing channels statistical anomalies?" in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017, pp. 1–9.

[27] 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*. Springer, 2017, pp. 105–122.

[28] S. Srinivasan, C. R. Bhat, and J. Holguin-Veras, "Empirical analysis of the impact of security perception on intercity mode choice: A panel rank-ordered mixed logit model," *Transportation Research Record*, vol. 1942, no. 1, pp. 9–15, 2006.

[29] W. Gasior and L. Yang, "Network covert channels on the android platform," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, 2011, pp. 1–1.

[30] 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.

[31] G. Shah, A. Molina, M. Blaze *et al.*, "Keyboards and covert channels." in *USENIX Security Symposium*, vol. 15, 2006.

[32] H. Hovhannisyan, K. Lu, and J. Wang, "A novel high-speed ip-timing covert channel: Design and evaluation," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 7198–7203.

[33] A. El-Atawy, Q. Duan, and E. Al-Shaer, "A novel class of robust covert channels using out-of-order packets," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 116–129, 2015.

[34] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert channel detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 4, pp. 1–29, 2009.

[35] S. Gianvecchio and H. Wang, "An entropy-based approach to detecting covert timing channels," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 785–797, 2010.

[36] R. Archibald and D. Ghosal, "A covert timing channel based on fountain codes," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications.* IEEE, 2012, pp. 970–977.

[37] J. Zhai, G. Liu, and Y. Dai, "A covert channel detection algorithm based on tcp markov model," in *2010 International Conference on Multimedia Information Networking and Security.* IEEE, 2010, pp. 893–897.

[38] 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).* IEEE, 2015, pp. 153–159.

[39] 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).* IEEE, 2017, pp. 1102–1107.

[40] S. Zander, G. Armitage, and P. Branch, "Stealthier inter-packet timing covert channels," in *International Conference on Research in Networking.* Springer, 2011, pp. 458–470.

[41] 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.

[42] 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.

[43] T. Sohn, J. Moon, S. Lee, D. H. Lee, and J. Lim, “Covert channel detection in the icmp payload using support vector machine,” in *International Symposium on Computer and Information Sciences.* Springer, 2003, pp. 828–835.

[44] S. Mou, Z. Zhao, S. Jiang, Z. Wu, and J. Zhu, “Feature extraction and classification algorithm for detecting complex covert timing channel,” *Computers & Security*, vol. 31, no. 1, pp. 70–82, 2012.

[45] O. Darwish, A. Al-Fuqaha, G. B. Brahim, I. Jenhani, and A. Vasilakos, “Using hierarchical statistical analysis and deep neural networks to detect covert timing channels,” *Applied Soft Computing*, vol. 82, p. 105546, 2019.

[46] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Alemany, “Automated mapping of large binary objects using primitive fragment type classification,” *digital investigation*, vol. 7, pp. S3–S12, 2010.

[47] B. Anderson, C. Storlie, and T. Lane, “Improving malware classification: bridging the static/dynamic gap,” in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012, pp. 3–14.

[48] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, “Malware images: visualization and automatic classification,” in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[49] A. Makandar and A. Patrot, “Wavelet statistical feature based malware class recognition and classification using supervised learning classifier,” *Oriental journal of computer science and technology*, vol. 10, no. 2, pp. 400–406, 2017.

[50] K. Han, J. H. Lim, and E. G. Im, “Malware analysis method using visualization of binary files,” in *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, 2013, pp. 317–321.

[51] D. Díaz-Pernil, A. Berciano, F. Peña-Cantillana, and M. A. Gutiérrez-Naranjo, “Bio-inspired parallel computing of representative geometrical objects of holes of binary 2d-images,” *International Journal of Bio-Inspired Computation*, vol. 9, no. 2, pp. 77–92, 2017.

[52] E. Aminanto and K. Kim, “Deep learning in intrusion detection system: An overview,” in *2016 International Research Conference on Engineering and Technology (2016 IRCET)*. Higher Education Forum, 2016.

[53] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, “A survey of machine and deep learning methods for internet of things (iot) security,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1646–1685, 2020.

[54] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, “A survey of deep learning methods for cyber security,” *Information*, vol. 10, no. 4, p. 122, 2019.

[55] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning,” in *2017*

International Conference on Information Networking (ICOIN). IEEE, 2017, pp. 712–717.

[56] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, 2020.

[57] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[58] R. Kumar, Z. Xiaosong, R. U. Khan, I. Ahad, and J. Kumar, "Malicious code detection based on image processing using deep learning," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, 2018, pp. 81–85.

[59] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers & Security*, vol. 77, pp. 871–885, 2018.

[60] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2017, pp. 43–48.

[61] H. Yang and F. Wang, "Wireless network intrusion detection based on improved convolutional neural network," *Ieee Access*, vol. 7, pp. 64 366–64 374, 2019.

[62] R. C. Staudemeyer, “Applying long short-term memory recurrent neural networks to intrusion detection,” *South African Computer Journal*, vol. 56, no. 1, pp. 136–154, 2015.

[63] F. Jiang, Y. Fu, B. B. Gupta, Y. Liang, S. Rho, F. Lou, F. Meng, and Z. Tian, “Deep learning based multi-channel intelligent attack detection for data security,” *IEEE transactions on Sustainable Computing*, vol. 5, no. 2, pp. 204–212, 2018.

[64] R. B. Krishnan and N. Raajan, “An intellectual intrusion detection system model for attacks classification using rnn,” *Int. J. Pharm. Technol*, vol. 8, no. 4, pp. 23 157–23 164, 2016.

[65] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, “Long short term memory recurrent neural network classifier for intrusion detection,” in *2016 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2016, pp. 1–5.

[66] A. F. M. Agarap, “A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data,” in *Proceedings of the 2018 10th international conference on machine learning and computing*, 2018, pp. 26–30.

[67] H. Liu, B. Lang, M. Liu, and H. Yan, “Cnn and rnn based payload classification methods for attack detection,” *Knowledge-Based Systems*, vol. 163, pp. 332–341, 2019.

[68] J. Kim, H. Kim *et al.*, "An effective intrusion detection classifier using long short-term memory with gradient descent optimization," in *2017 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2017, pp. 1–6.

[69] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Security and Communication Networks*, vol. 2017, 2017.

[70] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3854–3861.

[71] F. Farahnakian and J. Heikkonen, "A deep auto-encoder based approach for intrusion detection system," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2018, pp. 178–183.

[72] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, 2014, pp. 4–11.

[73] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5597–5621, 2019.

[74] L. Chappell, "Wireshark 101: Essential skills for network analysis-wireshark solution series," *Laura Chappell University, USA*, 2017.

[75] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[76] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[77] W. Rasband, "1997–2018 imagej," *Bethesda, MD: US National Institutes of Health.*

[78] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[79] T. K. Lee, W. J. Baddar, S. T. Kim, and Y. M. Ro, "Convolution with logarithmic filter groups for efficient shallow cnn," in *International Conference on Multimedia Modeling.* Springer, 2018, pp. 117–129.

[80] J. Li, S. Xie, Z. Chen, H. Liu, J. Kang, Z. Fan, and W. Li, "A shallow convolutional neural network for apple classification," *IEEE Access*, vol. 8, pp. 111 683–111 692, 2020.

[81] F. Lei, X. Liu, Q. Dai, and B. W.-K. Ling, "Shallow convolutional neural network for image classification," *SN Applied Sciences*, vol. 2, no. 1, pp. 1–8, 2020.

[82] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[83] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[84] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[85] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning.* PMLR, 2015, pp. 448–456.

[86] L. Lu, Y. Yang, Y. Jiang, H. Ai, and W. Tu, "Shallow convolutional neural networks for acoustic scene classification," *Wuhan University Journal of Natural Sciences*, vol. 23, no. 2, pp. 178–184, 2018.

[87] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21 954–21 961, 2017.

[88] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[89] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International conference on machine learning.* PMLR, 2014, pp. 1764–1772.

[90] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning.* PMLR, 2013, pp. 1310–1318.

[91] R. C. Aygun and A. G. Yavuz, "Network anomaly detection with stochastically improved autoencoder based models," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud).* IEEE, 2017, pp. 193–198.

[92] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE access*, vol. 6, pp. 48 231–48 246, 2018.

[93] T. Vaiyapuri and A. Binbusayyis, "Application of deep autoencoder as an one-class classifier for unsupervised network intrusion detection: a comparative evaluation," *PeerJ Computer Science*, vol. 6, p. e327, 2020.

[94] B. Bayram, T. B. Duman, and G. Ince, "Real time detection of acoustic anomalies in industrial processes using sequential autoencoders," *Expert Systems*, vol. 38, no. 1, p. e12564, 2021.

[95] S. Abirami and P. Chitra, "Energy-efficient edge based real-time healthcare support system," in *Advances in Computers.* Elsevier, 2020, vol. 117, no. 1, pp. 339–368.

[96] G. D'Angelo and F. Palmieri, "Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features

extraction," *Journal of Network and Computer Applications*, vol. 173, p. 102890, 2021.

[97] B. Yan and G. Han, "Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system," *IEEE Access*, vol. 6, pp. 41 238–41 248, 2018.

[98] D. Rumethart, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[99] M. Ranzato, C. Poultney, S. Chopra, Y. LeCun *et al.*, "Efficient learning of sparse representations with an energy-based model," *Advances in neural information processing systems*, vol. 19, p. 1137, 2007.

[100] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.

[101] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International conference on artificial neural networks.* Springer, 2011, pp. 52–59.

[102] S. Chen, J. Yu, and S. Wang, "One-dimensional convolutional auto-encoder-based feature learning for fault diagnosis of multivariate processes," *Journal of Process Control*, vol. 87, pp. 54–67, 2020.

[103] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot, "Higher order contractive auto-encoder," in *Joint European conference on*

machine learning and knowledge discovery in databases. Springer, 2011, pp. 645–660.

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

[105] R. Soleymani, E. Granger, and G. Fumera, "F-measure curves: A tool to visualize classifier performance under imbalance," *Pattern Recognition*, vol. 100, p. 107146, 2020.