A Dual-Layer Framework for Detecting and Mitigating Covert Timing Channels

Sapna V.M. * , Karan Bhat Sumbly , Ghanashyam Mahesh Bhat , Vinay Kumar S R , Prasad B. Honnavalli

Posted Date: 18 September 2025

doi: 10.20944/preprints202509.1597.v1

Keywords: covert channel; decision trees; detection; inter-arrival times; kernel density estimation; machine learning; network communications; prevention; statistics; tcp timestamp; ttl; two-layer

Article

A Dual-Layer Framework for Detecting and Mitigating Covert Timing Channels

Sapna V.M. *, Karan Bhat Sumbly, Ghanashyam Mahesh Bhat, Vinay Kumar S R and Prasad B. Honnavalli

Department of Computer Science, PES University, Bengaluru, India
* Correspondence: sapnavm@pes.edu

Abstract

Covert timing channels pose serious challenges in secure computing environments, where even minor information leaks can lead to severe consequences. These channels exploit subtle timing variations to bypass conventional safeguards such as firewalls, intrusion detection systems, and encryption, making them particularly difficult to identify. In this work, we propose a two-layered detection and mitigation strategy to address this threat. The first layer employs a decision tree classifier supported by well-defined classification rules, while the second layer introduces additional verification measures to strengthen detection accuracy. To evaluate the proposed method, we developed a controlled testbed capable of simulating multiple covert timing channel scenarios. Experimental results show that our approach effectively identifies and limits covert timing activity, even when advanced evasion techniques are applied. This study provides a practical contribution toward improving network resilience and defending critical infrastructures against covert communication threats.

Keywords: covert channel; decision trees; detection; inter-arrival times; kernel density estimation; machine learning; network communications; prevention; statistics; tcp timestamp; ttl; two-layer

1. Introduction

A covert channel is a concealed communication channel employed by two communicating entities to obscure communication patterns and evade security measures. Although covert channels come in different forms, network covert channels are gaining popularity. Such covert channels transmit secret information via network communication schemes via network communication protocols such as TCP/IP protocols, a structured carrier [1], and their header, time fields, or inter-packet ordering that deviates from the communication schemes' original design [2].

Because of advances in computing power, malicious entities can now effectively use covert channels for data exfiltration, botnet orchestration, and remote command transmission while avoiding detection by firewalls, intrusion detection systems, and anti-viruses due to the limited generalizability and scalability of detection and analysis technologies. Network covert channels have proven efficient in facilitating numerous nefarious operations, primarily because they enable two hosts to exchange information secretly by inserting data into a legitimate traffic flow [3,4]. Covert channels can serve legitimate purposes, such as implementing digital watermarking in VoIP communication, establishing traceback techniques, and circumventing censorship. However, covert channels are exploited for illegal or fraudulent activities like malware transmission, penetration attacks, and data exfiltration, making them one of the growing cybersecurity threats to governments and businesses [4]. The problem stems from the fact that there are numerous ways to conceal information in a covert channel, yet existing technologies can only address specific sorts of channels.

The presented work focuses primarily on covert channel communications based on the TCP/IP protocol stack, especially timing covert channels exploiting TCP/IP time fields. Such timing-covert

 doi:10.20944/preprints202509.1597.v1

channels are difficult to detect without a dedicated mechanism. This paper sheds light on a two-layered architecture for detecting and preventing timing covert channels in computer networks using proactive and reactive approaches. The approaches are designed to work with different network protocols. For detection purposes, decision tree classification emerged as the best option for detecting covert channels. The discussion also includes the methods to disrupt the covert channels once detected.

2. Related Work

Covert channels have become a significant topic within cybersecurity. The concept originated in 1983 with Butler Lampson during a process of transferring information. Since then, growing interest in this field has led to major advancements in detecting and preventing covert channels.

In their research, Felix Iglesias and Robert Annessi introduced an innovative method for detecting covert channels within TCP/IP traffic [3]. Their approach, based on Descriptive Analysis of Traffic (DAT) detectors, is applied in a passive warden context. These detectors are engineered to perform an in-depth analysis of network traffic, establishing a flexible framework that can identify a wide range of covert channel techniques highlighted in the existing body of research. By integrating DAT detectors into network intrusion detection systems, they offer a swift and effective solution for lightweight analysis of network traffic. The methodology proposed by Iglesias and Annessi leverages multimodality estimations and the aggregate of auto-correlation coefficients as key elements of the DAT approach. These components serve as filters that help to effectively sift through incoming traffic, thereby minimizing dependence on more computationally demanding detection methods. The researchers underline the unique characteristics of covert channels, noting that these can be identified through specific shapes and patterns in the data. This insight proves the adaptability of DAT detectors, which can be easily updated to encompass new detection characteristics and methodologies. Furthermore, the study explores the theoretical underpinnings of the DAT detectors, including the distinctive features and statistical estimations that facilitate the detection of covert channels. This exploration into the theoretical aspects enriches the understanding of how DAT detectors can be effectively employed in the ongoing battle against covert communications in network traffic.

In addition to the work described in [3], the authors broadened their efforts in [4] to properly evaluate DAT detectors for the specific scenario of covert timing channels. A testing environment was established to create and evaluate covert timing channels by implementing eight widely used covert timing channels. The detection of channels was analyzed and tested using supervised classification. The DAT detector parameters were tweaked, and generalized decision rules were generated using decision tree learners to properly classify the covert channels. In [5], F. Iglesias et al. extended the studies by incorporating more features into the DAT vector, which, although increased computational costs, improved detection performance. The actual implementation of the decision tree to classify covert timing channels followed a three-phase technique: dataset generation phase, model obtaining, and the rule generalization phase. The observation window was set at five minutes, and the primary goal was to identify covert and overt channels using machine learning models. Based on the data, a more generalized decision tree model was proposed, which produced optimal outcomes.

In the study referenced as [6], the authors present bccstego, a sophisticated framework designed for detecting unusual patterns or behaviors in network packet headers. While effective, these detection methods can potentially impair device performance and lengthen processing times. Concurrently, another investigation sought to devise techniques for integrating covert traffic into both live communication systems and previously captured traffic flows. The ccgen tool, highlighted in [7], emerges as an open-source solution that acts as a straightforward script plugin, encompassing 18 different covert communication mechanisms. This tool can be seamlessly integrated with existing network technologies, facilitating the deployment of covert channels.

Further exploration revealed that covert communication could be achieved between two devices within the same local area network (LAN) using ARP (Address Resolution Protocol) broadcast request messages. This technique involves one device sending ARP broadcast requests to a specifically targeted

 doi:10.20944/preprints202509.1597.v1

recipient, along with a shared encryption seed. This process encrypts and surreptitiously transmits secret data, embedding the hidden information within the ARP request instead of the standard ARP content, thus directing it to the intended receiver [8].

Additionally, PCap stego was developed as a tool for establishing network covert channels capable of generating authentic traffic captures in the .pcap format. This innovation is particularly noteworthy as it does not alert to the presence of covert data communications, making it an ideal platform for researchers and practitioners to conduct experiments on network covert channels. It facilitates the rapid creation of datasets through automated techniques, supporting extensive experimental analysis.

3. Two Layered Approach

The objective of this research is to utilize the best of the existing covert channel detection models and develop a novel prevention mechanism to mitigate network timing covert channels. A two-layered detection method is proposed, emphasizing the importance of speeding up the process. It is crucial to note that due to their limited generalizability, general-purpose detection systems cannot be deployed effectively. Instead, each detection system must be meticulously designed for a specific covert channel [7]. The first layer focuses on the detection and prevention of covert communications, specifically by monitoring changes in timing-related parameters within packet headers, with a particular emphasis on Time-to-Live (TTL) values and TCP Timestamps. The second layer aims at identifying and preventing covert communications based on the inter-arrival time (IAT) of packets, through the application of Machine Learning techniques. For this purpose, a Decision Tree model is incorporated to derive classification rules for effective detection. By employing a two-layer approach, this research avoids the overhead typically associated with covert channel detection.

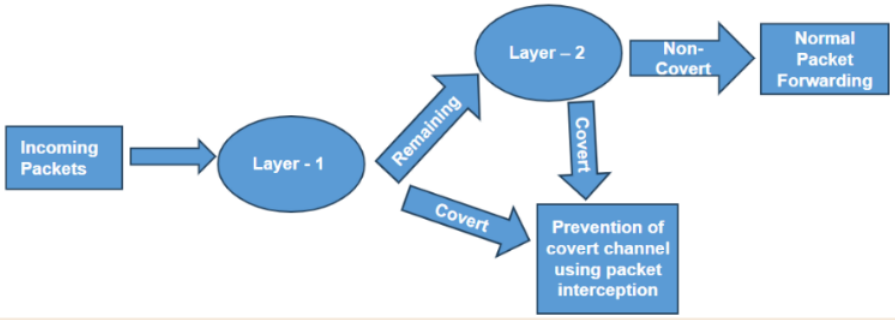

Figure 1. Multi-layer covert channel detection and prevention.

3.1. Layer 1

Layer 1 utilizes a proactive approach where flows are analyzed for suspicious covert communications. The term "flow" describes the movement of packets from a source IP address to a destination IP address. This layer is solely focused on covert communication using timing-related packet-header information, that is, TTL values and TCP timestamps.

3.1.1. TTL Value based communication

Under the Internet Protocol, the Time-to-Live (TTL) is defined as an 8-bit field. In the header of an IPv4 packet, the TTL field is situated as the 9th octet within the standard 20-octet header. Typically, the TTL values for packets within a specific source-destination IP pair remain constant. This consistency is expected under normal network conditions, as the TTL value is set by the originating host and decremented by routers along the packet's path to its destination. In the context of covert channels that exploit TTL values, this 9th octet is manipulated to encode bits of information, thereby altering its original purpose to control packet lifespan and instead using it to clandestinely transmit data. To detect such covert channels, network traffic is monitored for a duration of 5 minutes before drawing any conclusions. The detection strategy proactively assumes that any flow with a number of distinct TTL values greater than 1 in a 5-minute monitoring window is potentially covert.

 doi:10.20944/preprints202509.1597.v1

3.1.2. TCP timestamp based communication

The Timestamps option in the TCP header allows endpoints to maintain an up-to-date measurement of the round-trip time (RTT) of the network between them. This measurement is crucial as it aids each TCP stack in setting and adjusting its retransmission timers, optimizing data flow efficiency and reliability. In the context of covert channels exploiting TCP timestamps, the least significant bit (LSB) of the timestamp is manipulated to synchronize with the bit of data intended for covert transmission. After this manipulation, the packet is sent, enabling the stealthy transfer of information embedded within the TCP timestamp field. We proactively assume that any TCP packet with a TCP timestamp field is potentially covert.

3.2. Layer 2

Layer 2 employs a reactive strategy where in-depth analysis and classification are performed. The emphasis of layer two is on scrutinizing flows that have the potential to exploit packet IATs for covert communication. For layer 2, network packets are sniffed, and their timestamp is recorded along with the source IP address, and destination IP address. It is prudent to take into account just unique timestamp values after acquiring timestamp data. For each source-destination IP pair, referred to as a flow [5], timestamp values are segregated, and packet IATs are subsequently calculated. If a flow has documented the arrival of n packets, then a total of n-1 IATs are derived. This calculation is based on the principle that the IAT for a given packet is determined by subtracting the arrival time of the preceding packet from the arrival time of the current packet, hence the need for n-1 calculations to account for the sequential nature of packet arrivals within a flow [5].

In this layer, Kernel Density Estimations (KDE) are employed to approximate the probability density function of the IAT distribution. Analyzing the KDE estimation plots facilitates understanding the tendencies and characteristics of covert communications. This KDE-IAT plot serves as the basis for extracting several statistical features, including:

- Total unique values (U) - The total distinct IAT values (ms)
- Total multimodality distribution peaks (S_k) - The number of significant peaks displayed by the kernel density plot
- Widthave (μ_{ws}) - The mean of the standard deviations of S_k peaks
- Percentage of IATs with the mode IAT value (p(Mo))
- Total number of packets (pkts)
- Estimation of the covert-byte equivalent symbol (c) - The number of possible covert bytes conveyed in a flow is estimated by the computation of c, which is based on empirical tables [2]. S_k is used to estimate how many distinct symbols the covert channel may be using. For instance, c becomes, c = (pkts –1)/7 if S_k indicates the possibility of a binary channel, assuming that seven IAT values are required to convey a hidden byte-equivalent symbol [2].

Once the aforementioned statistical features are identified, they are fed into a decision tree classifier described in [5], from which detection results are derived. The generalized decision tree simplifies the detection process into two primary patterns indicative of potential covert communications.

The first observed pattern is characterized by a significant number of packets pkts > 170 within the 5-minute observation window, but with a low potential for covert bytes c > 25). This pattern is typical for flows where most inter-arrival times (IATs) cluster very closely around a single central value, leading to a skewness (S_k) value of one or close to one. The second pattern focuses on the majority of identified covert channels, which display distributions where packet IATs are concentrated within a 1 ms to 50 ms range around central values. Moreover, when the mode value of IAT accounts for less than 70% of IATs, there is a high probability of observing a large number of estimated covert bytes (c> 50). Additionally, when the ratio of the number of unique values (in millisecond resolution) to estimated covert bytes (U/c < 0.9) is slightly below one, it indicates the presence of a structured pattern in IAT values, likely resulting from algorithmic patterns [5].

 doi:10.20944/preprints202509.1597.v1

By utilizing these generalized patterns, the second layer significantly enhances the accuracy of covert channel classification. This methodological approach allows for the effective identification of covert communications by leveraging statistical analysis and machine learning techniques.

To facilitate comprehension, Figure 2 presents a straightforward example. This example showcases a covert communication scenario between a client and server across two hosts, utilizing a total of nine packets with inter-arrival times (IATs) alternating between three and five seconds to represent bits 0 and 1, respectively. This results in a bimodal distribution pattern of IATs. In this case, the total number of packets (pkts) is 9. The total number of unique IATs (U) is 2, corresponding to the 5-second and 3-second intervals. Consequently, the packet mode percentage (p(Mo)) for the 3-second IAT is calculated as (5/8)*100 = 62.5%. Given that the data plot displays only two peaks, the number of multimodality distribution peaks (S_k) is identified as 2. The covert bytes (c) are determined by the formula (pkts-1)/7 = 1.14 bytes. These calculated values are subsequently utilized by the decision tree classifier to ascertain the presence of a covert channel. The KDE plot for this example is depicted in Figure 4, illustrating the bimodal distribution of IATs.

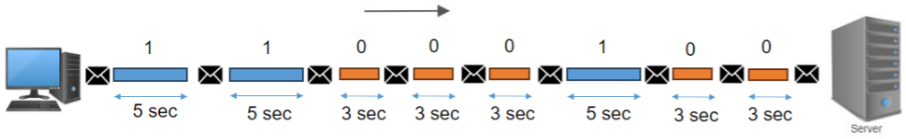

Figure 2. Illustration of timing based covert channels.

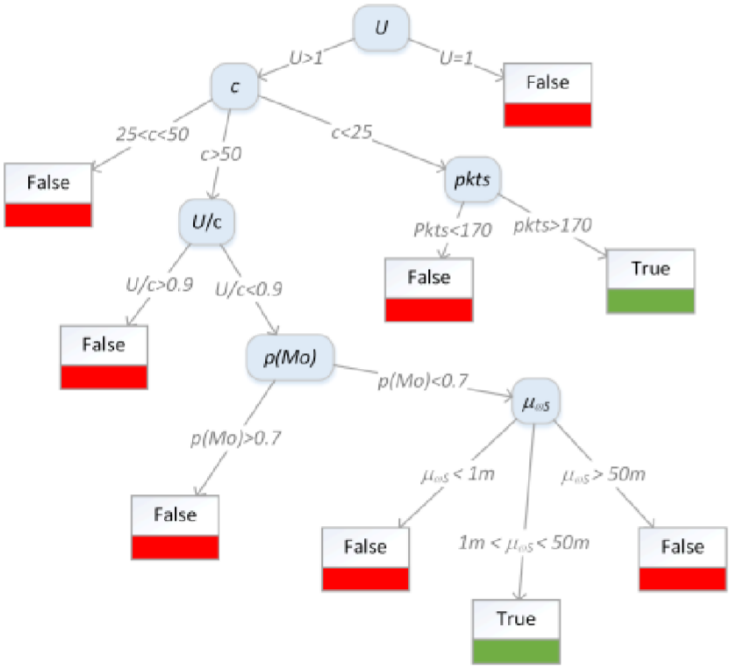

Figure 3. Decision tree to classify a flow as covert and non-covert.

In real-world scenarios, the inter-arrival times (IATs) used to represent bits in covert communications are unlikely to be perfectly consistent, due to the inherent variability introduced by factors like network congestion, packet processing delays, and overall network latency. These variances make it challenging to directly interpret the timing patterns as binary data without considering the noise and variations in the timings. To address this issue and accurately identify covert communications based on IATs, Kernel Density Estimation (KDE) plots are employed. By applying KDE to the IAT values, it's possible to visualize and analyze the distribution of these times, helping to distinguish between normal variations in packet timing and deliberate manipulations intended for covert commu-

 doi:10.20944/preprints202509.1597.v1

nications. KDE plots provide a smoothed, continuous view of the IAT distributions, highlighting the underlying patterns that may not be immediately apparent from the raw data. Peaks in a KDE plot can indicate the presence of timing intervals that are used more frequently, which could be a sign of covert communication if these intervals correspond to encoded binary data.

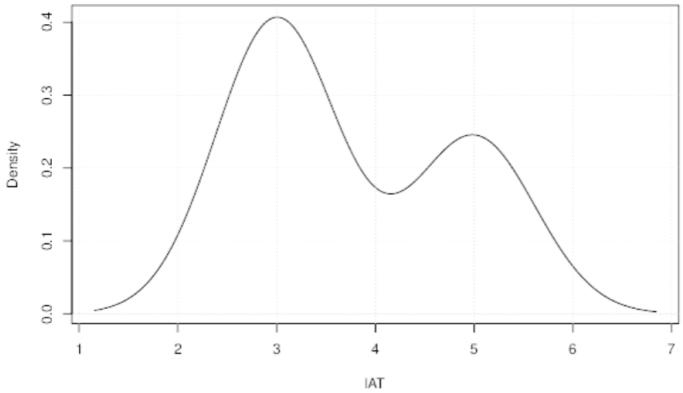

Figure 4. Kernel density estimation for packets IATs.

Figure 5 and Figure 6 show KDE-IAT plots for an actual Bi-Modal Covert Channel and a Non Covert Channel respectively.

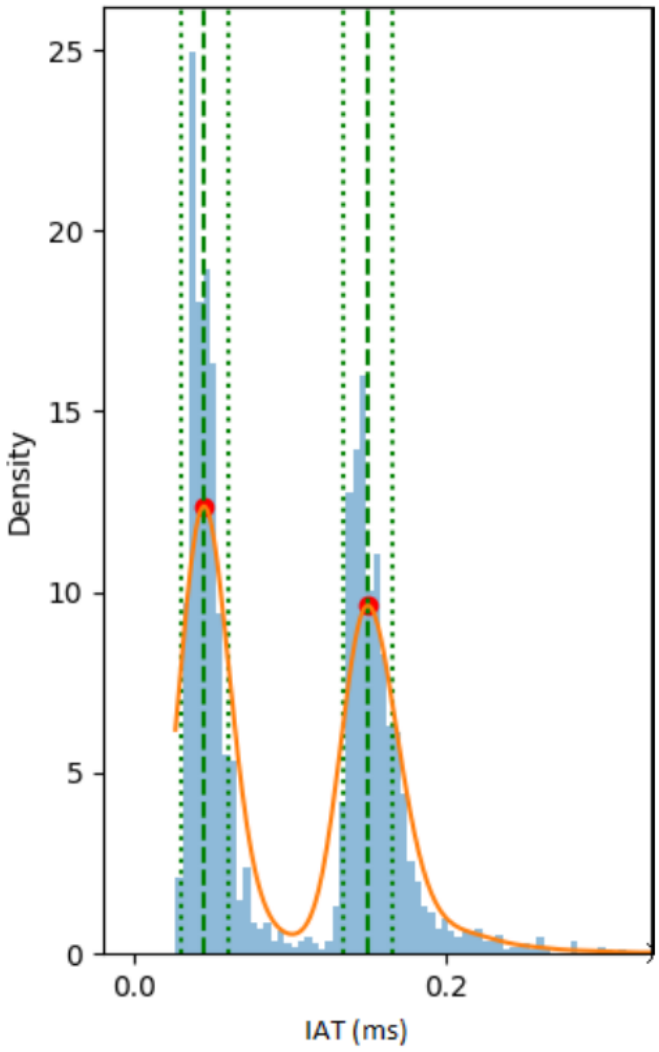

Figure 5. KDE-IAT plot for Bi-Modal Covert Channel.

 doi:10.20944/preprints202509.1597.v1

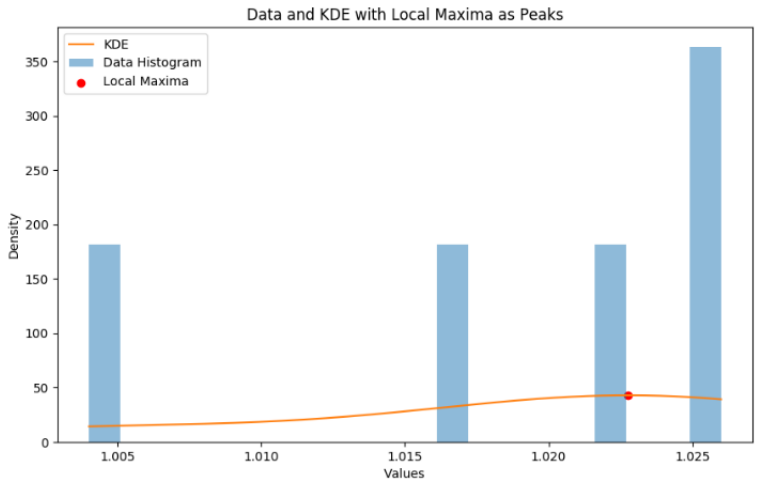

Figure 6. KDE-IAT plot for Non-Covert Channel.

4. Prevention

Prevention is done for timing covert channels that employ any of the following communication mechanisms: a) communications based on varying TTL values b) communications using TCP timestamps c) communications using packet inter-arrival times.

4.1. Communications Based on Varying TTL Values

The prevention mechanism for thwarting covert communications that exploit varying Time-to-Live (TTL) values operates at Layer 1, utilizing Netfilter hooks and kernel modules for its implementation. Netfilter hooks are integral to packet filtering within the Linux kernel, providing a powerful means to intercept and manipulate network packets as they traverse the network stack.

As established earlier, any network flow identified with more than one distinct TTL value within a 5-minute monitoring window is flagged as potentially engaging in covert communications. Upon detection, the prevention strategy involves standardizing the TTL value of every packet within the implicated flow. This standardization process adjusts the TTL of each packet to the maximum TTL value observed in that flow during the monitoring period, an operation facilitated by custom kernel modules. Once flagged covert, the TTL values of packets keep on getting dynamically updated in case a higher value is observed after the 5 monitoring period. Modifying the TTL necessitates adjustments to the IP header, specifically, recalculating the packet's checksum to maintain integrity and ensure proper forwarding by intermediate routers. This checksum recalculation is crucial because altering the TTL value changes the header content, upon which the checksum is based.

Consider a scenario in which two communicating parties, designated as A (the sender) and B (the receiver), aim to covertly transfer a binary string "01011101" utilizing variations in Time-to-Live (TTL) values. In this context, a TTL value of 50 is utilized to represent the binary bit '0', while a TTL value of 100 is used to denote the binary bit '1'. This communication is observed to occur within our predefined 5-minute monitoring window, which is the period during which our detection system scrutinizes network traffic for potential covert channels.

During this window, the series of packets transmitted from A to B would exhibit alternating TTL values corresponding to the binary sequence they intend to communicate. Specifically, the sequence of TTL values for the packets sent from A to B would be as follows, based on the binary string "01011101": 50, 100, 50, 100, 100, 100, 50, 100. This pattern of TTL values is a direct representation of the intended binary message, with each TTL value serving as a covert signal for a specific bit in the sequence.

After the monitoring period ends, our detection system would classify the communication flow from A to B as a covert channel based on the observed TTL variations as the number of TTL values recorded for this flow would be >1. As a countermeasure to disrupt further covert communication attempts, the system would enforce a policy wherein the TTL value of every subsequent packet within this flow is adjusted to the maximum observed value, in this case, 100.

 doi:10.20944/preprints202509.1597.v1

This corrective action ensures that if the communicating parties, A and B, attempt to transmit the same binary string "01011101" using TTL variations again, the mechanism employed by our system would alter the TTL values of all transmitted packets to 100. Consequently, from B's perspective, the series of received packets would now exhibit a uniform TTL value of 100 across the board, effectively translating to a binary sequence of "11111111". This alteration deviates significantly from the original intended message "01011101", thereby rendering the covert communication channel ineffective.

4.2. Communications Using TCP Timestamps

The prevention mechanism for thwarting communications utilizing TCP timestamps also operates within Layer 1, leveraging Netfilter hooks and kernel modules for implementation. A Netfilter hook function intercepts TCP packets and processes their TCP options, specifically on the TCP timestamp option.

The manipulation of the TCP timestamp's least significant bit (LSB) is the fundamental component of this preventive measure. Covert communications often hinge on subtle modifications of the LSB to encode secret data. By proactively altering the LSB to a consistent value (setting it to 1), the system effectively neutralizes the covert channel's ability to transmit hidden information through this vector. This is achieved by applying a bitwise 'OR' operation to the timestamp value, which ensures the LSB is always set to 1, irrespective of its original state.

Adjusting the TCP timestamp does not require a manual recalculation of the packet's checksum. In Linux, the TCP checksum is automatically recalculated after a packet has traversed through all network hooks right before the packet is dispatched. This automated checksum recalculation ensures the integrity of the packet is maintained after the timestamp modification, preserving the packet's validity for subsequent network operations.

Furthermore, in scenarios where TCP offloading is enabled, the responsibility for checksum calculation shifts to the Network Interface Card (NIC) just before the packets are transmitted to the physical layer. This hardware-level processing ensures that even with our modifications to the TCP timestamp, the packet's checksum is accurately computed, guaranteeing the packet's proper formation and readiness for transmission across the network.

4.3. Communications Based on Packet IATs

The prevention mechanism for communications exploiting packet inter-arrival times (IATs) is implemented in Layer 2. This approach is necessitated due to the subtlety of covert communications that utilize variations in IATs for clandestine data transfer. Initially, introducing random delays into the network traffic might appear as an effective countermeasure to disrupt these covert channels. However, such a broad application of delays could inadvertently increase the overall network latency, adversely affecting the quality of service for legitimate users and not just the entities engaged in covert communications. To circumvent this challenge without compromising network performance for regular traffic, the strategy involves applying targeted delays specifically to those flows identified as covert during the detection phase. This nuanced approach ensures that only the traffic flows suspected of harboring covert communications are subjected to delay modifications. By doing so, it maintains the integrity and efficiency of the network for legitimate users, while effectively disrupting the timing patterns that covert channels rely upon for data transmission.

The prevention mechanism against covert communications leveraging inter-arrival times (IATs) uses the Hierarchical Token Bucket (HTB) queuing discipline (qdisc) within the Linux kernel's traffic control (tc) framework. The HTB algorithm is instrumental in creating a structured yet flexible mechanism for queue management, enabling differentiated handling of network traffic, particularly for identified covert flows versus normal traffic. Using HTB, distinct queues for each identified covert flow are created. Additionally, a default queue is maintained for all other traffic, ensuring that legitimate network operations proceed without undue delay or interference.

Upon detection of a covert flow, the system dynamically imposes delays on the packets within that flow. The delay strategy is designed to disrupt the covert communication pattern by making the

 doi:10.20944/preprints202509.1597.v1

IAT distribution uniform, thereby obfuscating the encoded information. Specifically, a delay in the range of $M/2 \pm M/2$ is added to each packet in the covert flow, where M represents the maximum IAT observed within the 5-minute monitoring window for that flow. This approach effectively neutralizes the bimodal or multimodal distribution patterns that are characteristic of covert communications, rendering the attempted data transmission unintelligible.

5. Packaging

The covert channel detection and prevention tool consists of a combination of multiple programs written in different programming languages; while most of the processes run in user space, there are a few that need to be added as Kernel Modules. The interoperability between these processes is achieved by file-based inter-process communication, mainly with JSON files, and communication between user space and kernel space is achieved via Kernel objects.

To make the tool accessible to normal users, we built a GUI-based application using the Django framework. It consists of HTML, CSS, and Javascript-based frontend and Django backend. Since the tool we have developed works on the concept of packet capture and analysis for the detection of covert channels and employs a prevention mechanism, it is ideal to deploy the tool in one of the routers. In order to control the tool deployed on the router one can open the Django application from any machine in the same network and control the detection and prevention mechanism after a successful admin login. With the codebase consisting of the usage of multiple programs written in multiple languages, these are streamlined to execute in the correct order using shell scripts. These shell scripts are in turn executed by the Django backend based on the admin's action on the GUI application.

The admin can log into the tool running on the router and initialize a 5-minute scan for covert channels. After successful scanning, it will list all the possible covert channels and the corresponding type of covert channel algorithm they use. The admin can then apply a prevention mechanism to these channels with a simple toggle switch selectively based on the type of covert channel that the admin wants to prevent.

6. Test Environment

The test environment is designed to test software in a virtual environment with several virtual machines and data transmission via hidden timing channels. The test environment consists of a client, a router, and a server. The client sends a covert message and the server decodes the same to get the original message. The test bed consists of 4 covert timing channels where our data is tested. The 4 covert timing channels implemented are: a) One threshold technique b) Time stamp manipulation c) Packet burst encoding [5] d) TTL value manipulation

6.1. One Threshold Technique

This method sets a threshold th, for inter-arrival times (IATs). Delays surpassing th are interpreted as 1s, while delays below th are regarded as 0s [5].

6.1.1. Client-Side Algorithm

1. Take the data to be sent as input from a file/user input in ASCII format
2. Convert the data to binary bits
3. Construct a UDP packet with the destination IP and Port of the Server
4. If the bit to be sent is "0", send the packet immediately
5. If the bit to be sent is "1", send the packet after a predefined interval

6.1.2. Server-Side Algorithm

1. Create an empty bitstring
2. Accept the packets coming from the client side and associate an arrival timestamp with each packet

 doi:10.20944/preprints202509.1597.v1

3. If the time difference between 2 consecutive packets is less than the predefined interval, append "0" to the bitstring, else append "1"
4. Convert bitstring back to ASCII format

6.2. Time Stamp Manipulation

This technique manipulates packet idts using the least significant bit of the TCP timestamp. A minimum inter-packet time (tb) of 10ms must be maintained. If the LSB matches the covert sign, the packet is delivered; otherwise, the condition is verified again after time tw [5].

6.2.1. Client-Side Algorithm

1. Take the data to be sent as input from a file/user input in ASCII format
2. Convert the data to binary bits
3. Create a TCP connection to the server side
4. If the bit to be sent is "0", wait till the LSB of the TCP timestamp is even and then send the packet
5. If the bit to be sent is "1", wait till the LSB of the TCP timestamp is odd and then send the packet

6.2.2. Server-Side Algorithm

1. Create an empty bitstring
2. Accept the packets coming from the client side
3. If the LSB of the TCP timestamp is even, append "0" to the bitstring, else append "1"
4. Convert bitstring back to ASCII format

6.3. Packet Bursting Encoding

This approach involves sending packet bursts with a waiting time interval, denoted as tw. The quantity of packets within a burst explicitly corresponds to the covert symbol or code fragment being transmitted. For instance, a burst of three plus 1 packets is employed to convey the symbol "3".

6.3.1. Client-Side Algorithm

1. Take the data to be sent as input from a file/user input in ASCII format
2. Convert the data to a binary string
3. Split the bitstring into small bitstrings of length 4
4. Create a TCP connection to the server side
5. Convert each small binary bitstring to integers between 0 and 15 say X
6. For every interval of 5 seconds, send X+1 packets, corresponding to each binary bitstring

6.3.2. Server-Side Algorithm

1. Create an empty bitstring
2. Accept the packets coming from the client side
3. Store the incoming packets in the pandas' data frame along with their arrival time.
4. Once the TCP connection is terminated, use hierarchical clustering using the ward's linkage to cluster the packets based on arrival time
5. Calculate the number of packets in each cluster say Y
6. Convert integer Y-1 into binary and append it to the bitstring for every cluster
7. Convert bitstring back to ASCII format

6.4. TTL Value Manipulation

This approach involves the manipulation of TTL values in the IPv4 header for covert communication involves a technique where specific TTL values, or a range of TTL values, are designated to represent binary data—specifically, bits 1 and 0.

 doi:10.20944/preprints202509.1597.v1

6.4.1. Client-Side Algorithm

1. Take the data to be sent as input from a file/user input in ASCII format
2. Convert the data to a binary string
3. A predetermined TTL value (or range of values) is assigned to represent the binary '1' bit.
4. A different TTL value (or range of values) is assigned to represent the binary '0' bit.
5. Create a UCP connection to the server side
6. Send packets to the server.

6.4.2. Server-Side Algorithm

1. Create an empty bitstring
2. Accept the packets coming from the client side
3. A predetermined TTL value (or range of values) is assigned to represent the binary '1' bit.
4. A different TTL value (or range of values) is assigned to represent the binary '0' bit.
5. Append '0' or '1' to the bitstring according to the TTL values of the incoming packets.
6. Convert bitstring back to ASCII format

7. Results and Summary

This section offers a summary and insights derived from a series of tests and experiments aimed at detecting covert channels within a network. Our methodology is structured around a two-layered approach, specifically designed to identify and disrupt covert communications.

7.1. Layer 1

In the first layer of our detection strategy, we focus on identifying covert communications through two primary indicators:

7.1.1. TCP Timestamp Manipulation

We analyze the optional TCP timestamp values present in packets. Covert channels may manipulate these fields to encode hidden messages.

7.1.2. TTL Value Ranges

We examine the TTL values of packets. Typically, packets maintain TTL values within specific ranges to ensure successful delivery. Deviations from these expected ranges can signal covert activities.

By using a proactive approach, we were able to successfully detect and prevent such covert channels.

7.2. Layer 2

Layer 2 of our approach employs more sophisticated techniques to both confirm the presence of covert channels and implement targeted disruptions. This layer is crucial for handling the complexities of covert communications that might evade simpler detection methods. At this layer, we leverage statistical analysis and a machine learning model (Decision Tree Classifier) to analyze the network packet flows. Specifically, we focus on:

7.2.1. Kernel Density Estimations (KDE)

These are used to approximate the probability density function of the Inter-Arrival Times (IATs) of packets. By analyzing the KDE plots, we can better understand the characteristics of covert communications, identifying peaks and patterns that may not be visible through simpler analyses.

7.2.2. Decision Tree Classifiers

Incorporating machine learning, decision tree classifiers are trained on features extracted from the network traffic, including those identified through KDE analysis. These classifiers can generalize

 doi:10.20944/preprints202509.1597.v1

the decision-making process to identify not just known patterns of covert communications but also potential new methods as they emerge.

Upon confirming the presence of covert channels, Layer 2 implements precise disruption techniques aimed at neutralizing the identified covert communications without impacting legitimate network traffic. These methods include selectively manipulating packet delays, traffic control, and queue management.

Figure 7 showcases the effective disruption of a bi-modal covert communication channel (Figure 5), transforming it into a channel with an unimodal, uniform KDE-IAT distribution. This alteration obstructs any potential for covert data transmission, signifying the successful neutralization of the covert communication attempt.

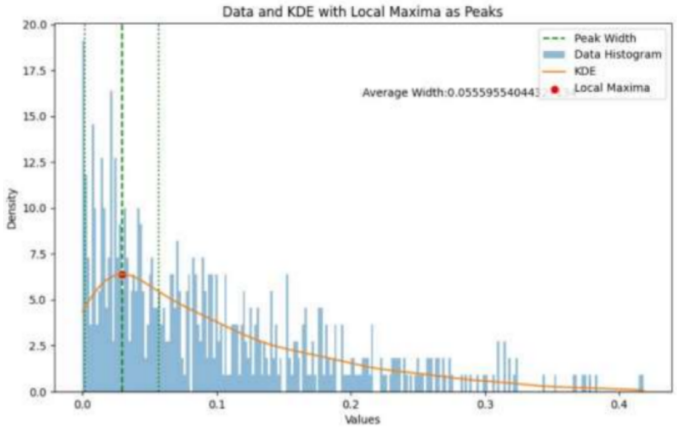

Figure 7. KDE-IAT plot for Bi-Modal Covert Channel after disruption.

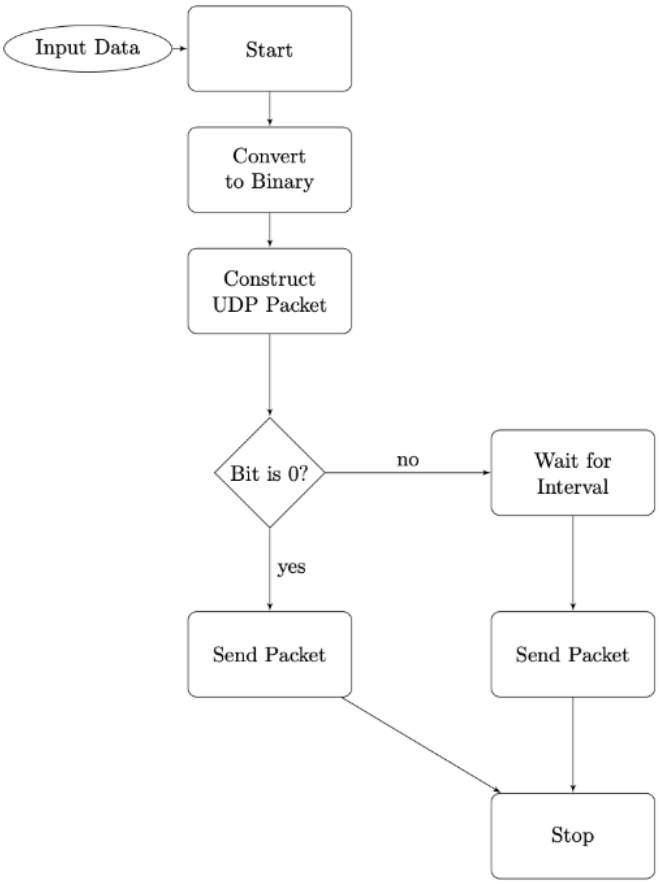

Figure 8. One Threshold Technique - Client Side Algorithm.

 doi:10.20944/preprints202509.1597.v1

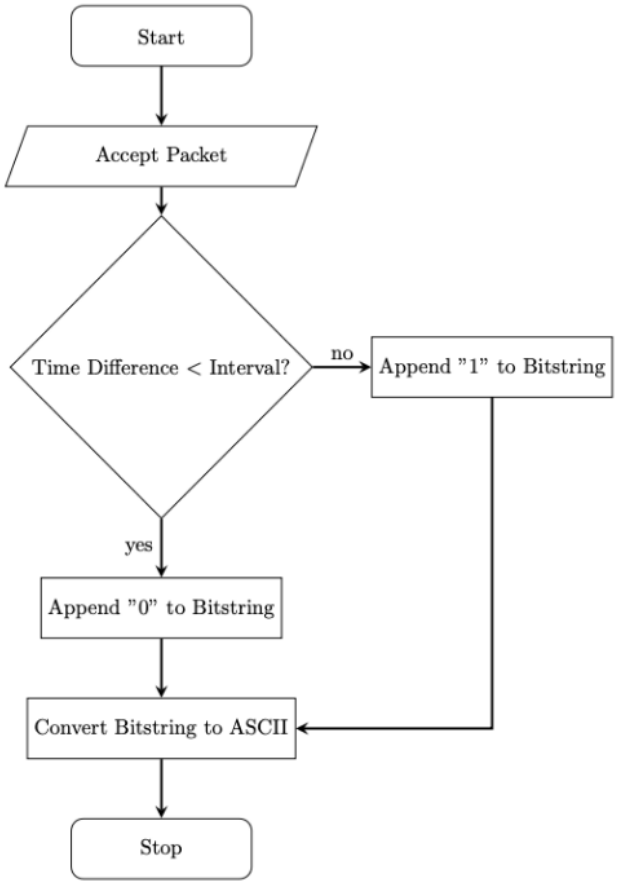

Figure 9. One Threshold Technique - Server Side Algorithm.

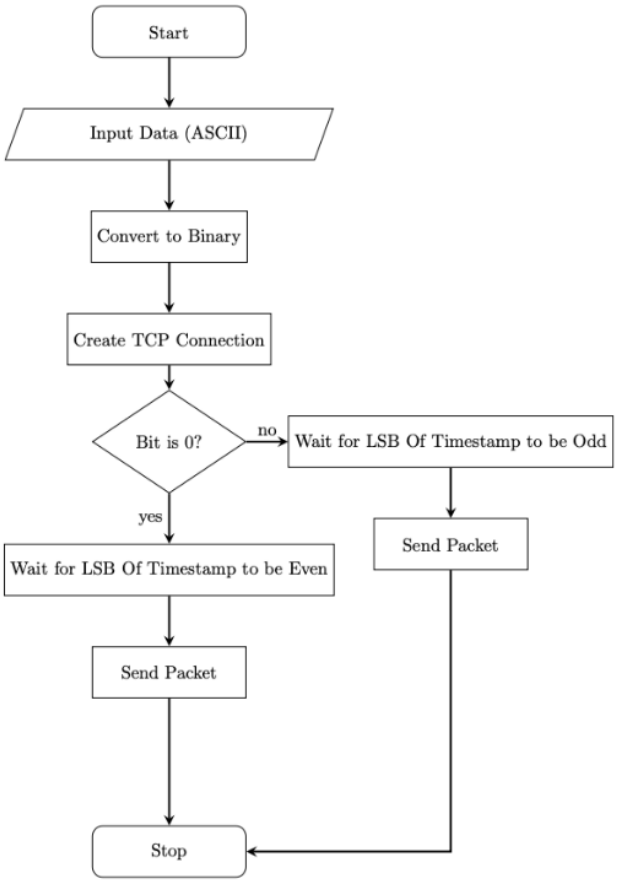

Figure 10. Time stamp manipulation - Client Side Algorithm.

 doi:10.20944/preprints202509.1597.v1

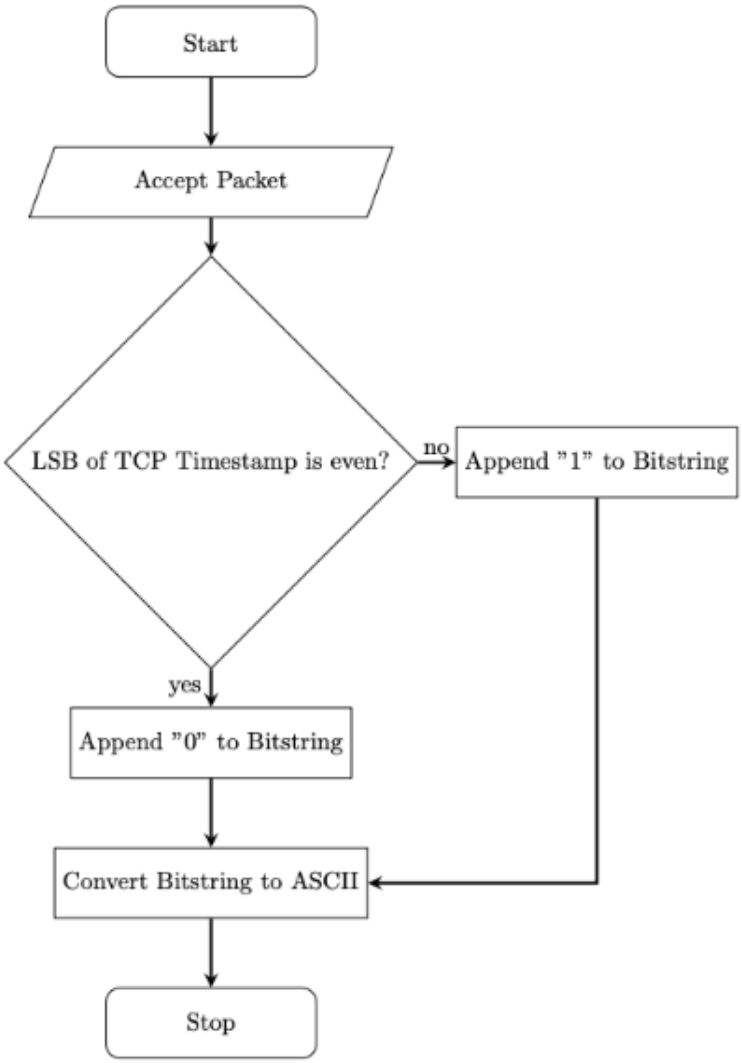

Figure 11. Time stamp manipulation - Server Side Algorithm.

 doi:10.20944/preprints202509.1597.v1

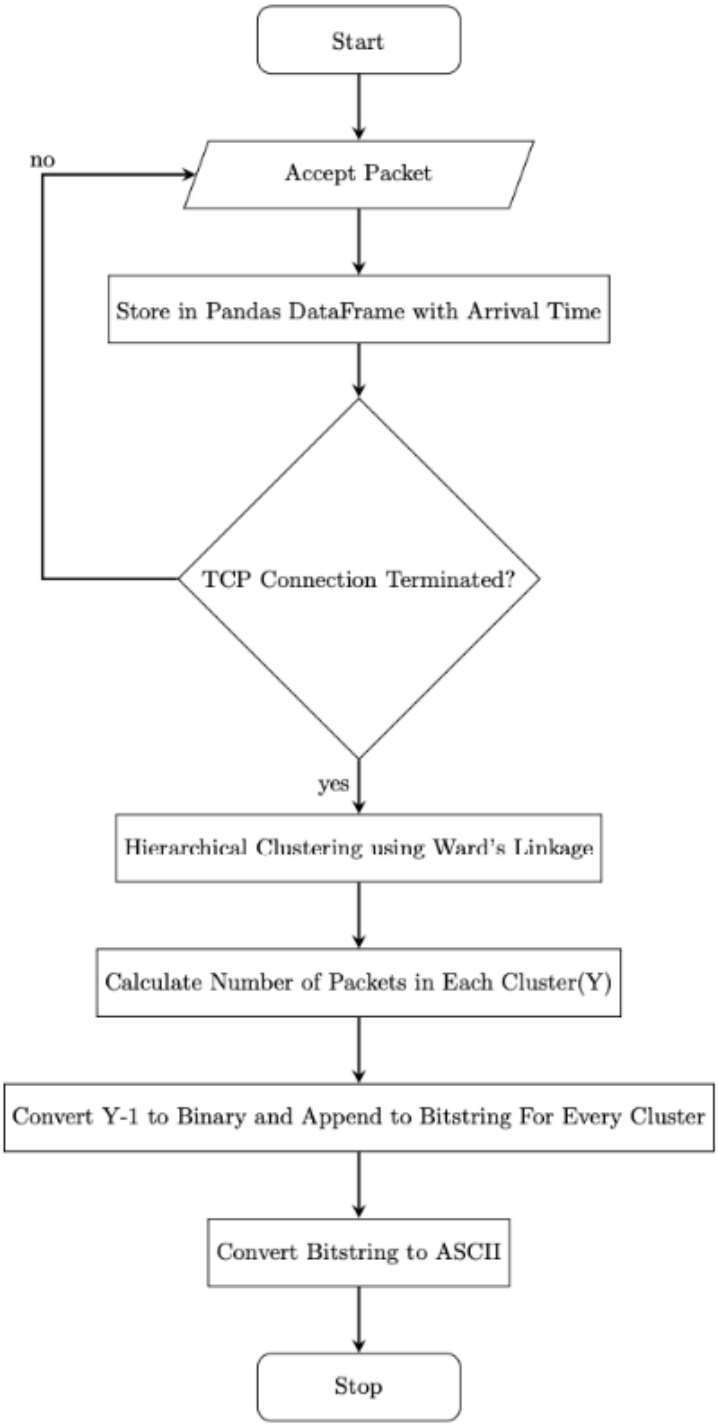

Figure 12. Packet bursting encoding - Client Side Algorithm.

 doi:10.20944/preprints202509.1597.v1

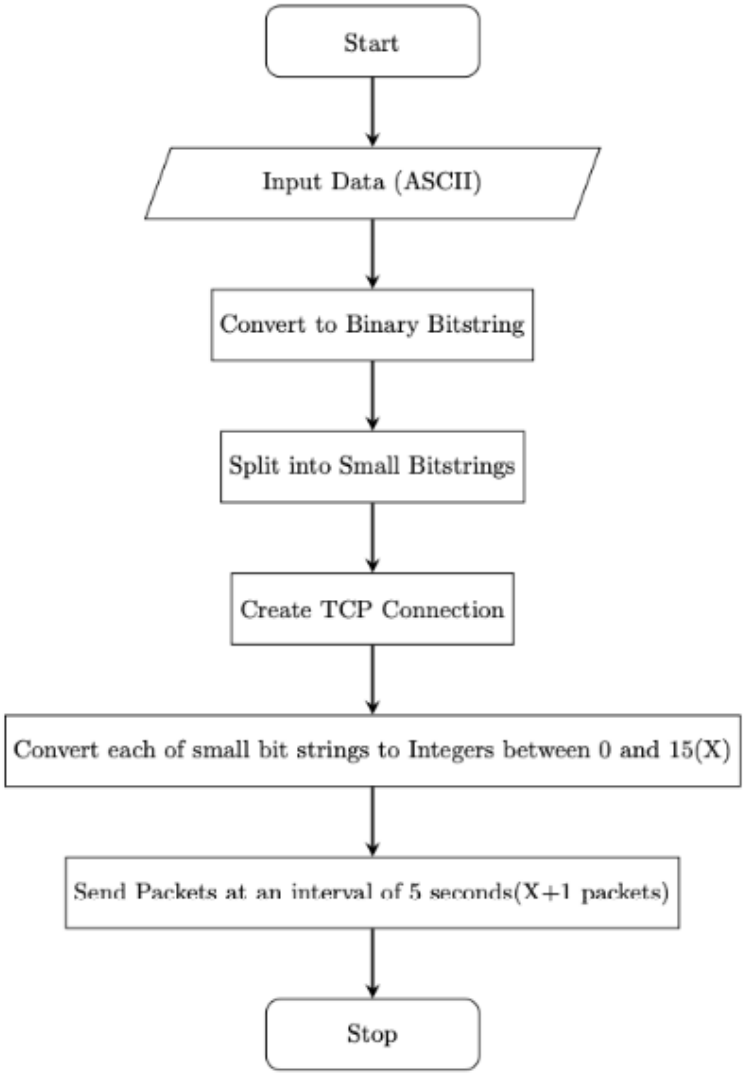

Figure 13. Packet bursting encoding - Server Side Algorithm.

8. Conclusions

This study presented a comprehensive approach to detecting and mitigating covert channels within network traffic, utilizing a two-layered detection mechanism focusing on TCP timestamp and TTL value manipulations. Through detailed analysis and real-time monitoring, we effectively identified covert communications characterized by distinct patterns, such as bimodal distributions in packet inter-arrival times (IATs) and variations in TTL values. By employing Kernel Density Estimation (KDE) and Decision Tree classifiers, we were able to discern between covert and non-covert channels based on statistical features extracted from network flows. The subsequent disruption of identified covert channels was achieved through precise manipulations of packet TTL values and TCP timestamps, as well as the strategic introduction of delays into covert flows, leveraging the Linux kernel's traffic control mechanisms. Our experiments, conducted in a controlled test environment, demonstrate the viability of this method in identifying and disrupting covert channels without significantly impacting legitimate network traffic. While the current study provides a solid foundation for detecting and mitigating covert channels, several avenues for future research and development remain open. Further exploration into advanced machine learning models and artificial intelligence could enhance the detection accuracy and efficiency. Furthermore, developing automated response mechanisms that can dynamically adjust mitigation strategies based on the type and severity of detected covert channels can be developed. This could involve more sophisticated traffic shaping, selective delay introduction, or packet alteration tactics to disrupt covert communications with minimal impact on legitimate traffic.

 doi:10.20944/preprints202509.1597.v1

As network traffic manipulation involves privacy and legal considerations, future work must also address the ethical implications of deploying such detection and mitigation techniques. Establishing guidelines and best practices for responsibly handling covert channel detection in compliance with legal standards and privacy norms is essential.

Lastly, to foster collaboration and further advancement in network security, we aim to make the covert channel detection and prevention tool open-source. By releasing the tool under an open-source license, we encourage contributions from the broader cybersecurity community, facilitating peer review, refinement, and adaptation for diverse network environments.

Author Contributions: Conceptualization, S.V.M. and P.B.H.; methodology, S.V.M.; software, K.B.S., G.M.B., K.L.K.G., and P.V.B.; validation, S.V.M., K.B.S., and P.B.H.; formal analysis, S.V.M.; investigation, K.B.S., G.M.B., K.L.K.G., and P.V.B.; resources, P.B.H.; data curation, K.B.S. and G.M.B.; writing—original draft preparation, S.V.M. and K.B.S.; writing—review and editing, S.V.M. and P.B.H.; visualization, G.M.B. and K.L.K.G.; supervision, S.V.M. and P.B.H.; project administration, S.V.M.; funding acquisition, P.B.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: The study was conducted in accordance with the Declaration of Helsinki, and approved by the Institutional Review Board of PES University (protocol code CS-2024-001 and date of approval 15 January 2024).

Informed Consent Statement: Not applicable for studies not involving humans.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy considerations related to network traffic analysis.

Acknowledgments: The authors would like to acknowledge the PESU Center for Information Security, Forensics, and Cyber Resilience (C-ISFCR) for providing the research infrastructure and support for this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IAT	Inter-Arrival Time
TTL	Time-to-Live
TCP	Transmission Control Protocol
DAT	Descriptive Analysis of Traffic
KDE	Kernel Density Estimation
HTB	Hierarchical Token Bucket
LSB	Least Significant Bit
RTT	Round-Trip Time
NIC	Network Interface Card
ARP	Address Resolution Protocol
LAN	Local Area Network
GUI	Graphical User Interface

9. Introduction

This is sample text to show the document compiles with references [1,9,15].

References

1. Elsadig, M.A.; Fadlalla, Y.A. Network Protocol Covert Channels: Countermeasures Techniques. *International Journal of Computer Applications* **2017**, *170*, 1–6.
2. Iglesias, F.; Bernhardt, V.; Annessi, R.; Zseby, T. Analytic Study of Features for the Detection of Covert Timing Channels in Network Traffic. *Journal of Cyber Security* **2017**, *2017*, 245–270.

 doi:10.20944/preprints202509.1597.v1

3. Iglesias, F.; Annessi, R.; Zseby, T. DAT detectors: uncovering TCP/IP covert channels by descriptive analytics. *Security and Communication Networks* **2016**, *9*, 3011–3029.
4. Iglesias, F.; Meghdouri, F.; Annessi, R.; Zseby, T. CCgen: Injecting Covert Channels into Network Traffic. *Proceedings of the 2022 ACM Conference on Computer and Communications Security* **2022**, pp. 1–11.
5. Iglesias, F.; Bernhardt, V.; Annessi, R.; Zseby, T. Decision Tree Rule Induction for Detecting Covert Timing Channels in TCP/IP Traffic. *1st International Cross-Domain Conference for Machine Learning and Knowledge Extraction* **2017**, pp. 105–122.
6. Repetto, M.; Caviglione, L.; Zuppelli, M. bccstego: A Framework for Investigating Network Covert Channels. *Proceedings of the 2021 IEEE International Conference on Communications* **2021**, pp. 1–6.
7. Zuppelli, M.; Caviglione, L.; Mazurczyk, W.; Schaffhauser, A.; Repetto, M. Code Augmentation for Detecting Covert Channels Targeting the IPv6 Flow Label. *Proceedings of the 2021 IEEE Global Communications Conference* **2021**, pp. 450–456.
8. Bedi, P.; Dua, A. ARPNetSteg: Network Steganography Using Address Resolution Protocol. *International Journal of Information Technology* **2020**, *66*, 671–677.
9. Elsadig, M.A.; Fadlalla, Y.A. Packet Length Covert Channel: A Detection Scheme. *Proceedings of the 2021 Conference on Network Security* **2021**, pp. 1–7.
10. Zuppelli, M.; Caviglione, L. pcapStego: A Tool for Generating Traffic Traces for Experimenting with Network Covert Channels. *Proceedings of the 2021 IEEE Conference on Communications* **2021**.
11. Elsadig, M.A.; Gafar, A. Covert Channel Detection: Machine Learning Approaches. *IEEE Access* **2022**, *10*, 38391–38405.
12. Zuppelli, M.; Repetto, M.; Schaffhauser, A.; Mazurczyk, W. Code Layering for the Detection of Network Covert Channels in Agentless Systems. *IEEE Transactions on Dependable and Secure Computing* **2022**, *19*, 2282–2294.
13. Pant, D.; Wason, M.; Chahal, J.S. Cross VM Covert Channel Implementation. *Proceedings of the 2018 International Conference on Information Security* **2018**.
14. Dua, A.; Jindal, V.; Bedi, P. Covert Communication using Address Resolution Protocol Broadcast Request Messages. *Proceedings of the 2021 IEEE Conference on Computer Networks* **2021**, pp. 1–6.
15. Wessa, P. Kernel Density Estimation (v1.0.12) in Free Statistics Software (v1.2.1). Office for Research Development and Education, 2015. Available online: http://www.wessa.net/rwasp_density.wasp/ (accessed on August 26, 2025).