CENG 519 - Network Security - Project Phase 1 Report

Kemal Anıl Kekevi
2380608

March 22, 2025

1 Processor Design

The processor subscribes to inpktsec and inpktinsec through NATS. When handling the message, it gets the message data and message subject then applies a random delay drawn from an exponential distribution with a specified mean, and republishes the delayed frames to related subscriber according to message subject. The delay is calculated as follows:

$$\text{delay} = \texttt{random.expovariate}\left(\frac{1}{\text{mean\_value}}\right)$$

Python's random library's expovariate function is a exponential distribution. lambd is 1.0 divided by the desired `mean_value`.

2 Experiment and Results

For evaluation, the processor was run with `mean_value` arguments starting from 5e-6 and increasing by a factor of 10 up to 5e-1. Additionally, values such as 1e-6, 0.1, and 10 were included for more accurate comparison. For each delay setting, the average RTT of approximately 50 ping packets was recorded.

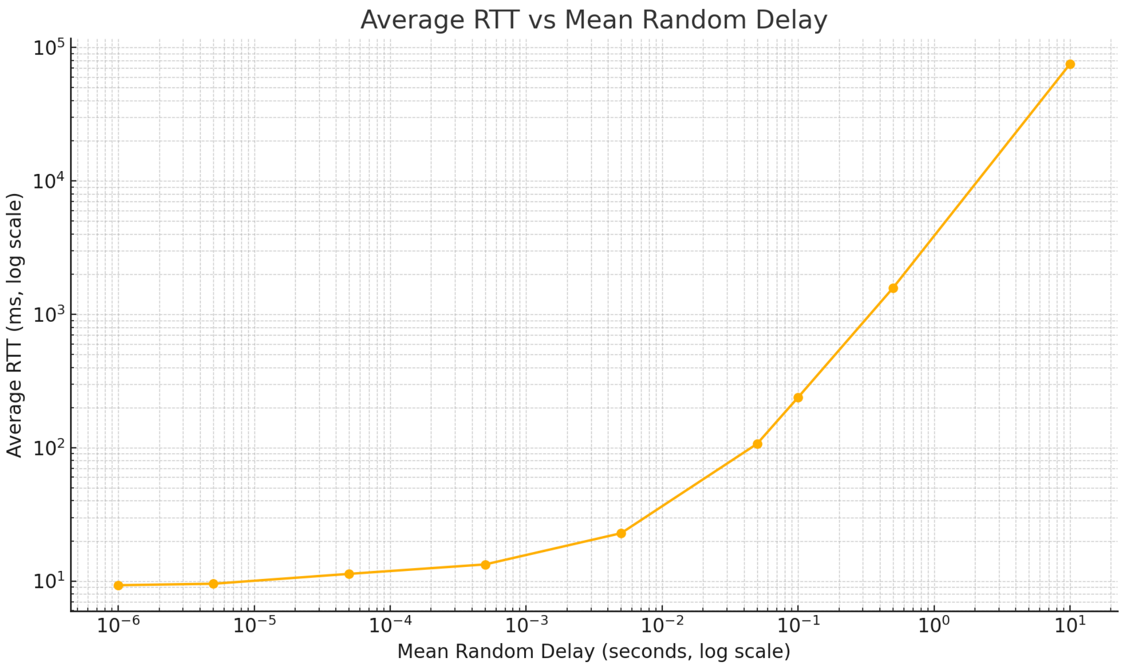

Figure: Average RTT vs Mean Delay applied by processor.

3 Conclusion

The results show that as the mean of the random delay increases, the average RTT rises significantly, especially beyond 10^{-2} seconds. For very small delays, RTT remains close to the base network latency, but larger delays cause RTT to grow rapidly, reaching over 75 seconds at a 10-second mean. This confirms the processor correctly models delay and highlights its impact on network performance.

4 GitHub Repository

Project code and this report are available at:
`https://github.com/ANILKE/middlebox/tree/phase1`

CENG 519 - Network Security - Project Phase 2 Report

Kemal Anıl Kekevi
2380608

April 13, 2025

1 Covert Channel Choice

My selection: Modify inter packet delay of packets: IP

2 Covert Channel Design

In this study, I implemented a timing-based covert channel using inter-packet delays. The system consists of a `covert_channel_sender` running in the `sec` container and a `covert_channel_receiver` running in the `insec` container, both implemented in Python.

Sender Design

The sender script accepts the following arguments: `port`, `message`, `zero_bit_delay`, `one_bit_delay`, and `bit_repeat_len`. Here:

- `message` is the string to be transmitted covertly.
- `zero_bit_delay` and `one_bit_delay` are the time intervals (in seconds) the sender sleeps between sending packets representing bits '0' and '1', respectively.
- `bit_repeat_len` specifies how many times each bit is redundantly transmitted to increase robustness against noise.

The sender first initializes the necessary variables and connects to the receiver via a UDP socket. It then iterates through the given `message's` bitstream (uses `create_bitstream_from_message` method to get messages bitstream), and for each bit, it sends the same packet `bit_repeat_len` times, applying the corresponding delay (from a predefined delay mapping) between transmissions.

Receiver Design

The receiver script accepts the following arguments: `port`, `sender_bit_delay`, `given_delay_threshold`, `bit_repeat_len`, and `bitstream_len`. These are defined as follows:

- `sender_bit_delay` is the delay used for bit '0' by the sender.
- `given_delay_threshold` is an offset added to `sender_bit_delay` to form the decision boundary for classifying incoming bits.
- `bit_repeat_len` is the number of packets expected per bit.
- `bitstream_len` is used for logging and evaluation purposes only.

The receiver initializes internal variables and begins listening on the specified UDP port. For each received packet, it computes the inter-packet delay (IPD) and appends it to a list. When the number of recorded IPDs reaches `bit_repeat_len`, the receiver computes the mean IPD for that group and compares it against the threshold value (`sender_bit_delay` + `given_delay_threshold`). If the average delay is less than the threshold, the bit is interpreted as '0'; otherwise, as '1'. The IPD list is then cleared for the next bit. This process repeats indefinitely.

Statistical Analysis Method

To estimate the performance with statistical confidence, the sample mean and the 95% confidence interval were computed using the following formula:

$$\bar{x} = \text{mean of samples}$$
$$\text{SEM} = \frac{s}{\sqrt{n}}$$
$$\text{CI}_{95\%} = \bar{x} \pm t_{(n-1,0.975)} \cdot \text{SEM}$$

Where s is the sample standard deviation, n is the number of samples, and t is the t-distribution critical value.

3 Experimentation Campaign

Experiment 1

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:3
- Bitstream Length: 100
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05

Result:

- BER: 0.0000 (95% CI: [0.0000 – 0.0000])
- Channel capacity(effective throughput — the number of useful bits per second): 2.2032 95% CI: [2.2026 – 2.2038]
- Avg. Transmission time: 45.383 seconds

Experiment 2

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:2
- Bitstream Length: 10
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05

Result:

- BER: 0.28 (95% CI: [0.254, 0.306])
- Channel capacity(effective throughput — the number of useful bits per second): 2.58 (95% CI: [2.42, 2.74])
- Avg. Transmission time: 2.8247 seconds

Experiment 3

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:10
- Bitstream Length: 10
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05

Result:

- BER: 0.0000 (95% CI: [0.0000 – 0.0000])
- Channel capacity (effective throughput — the number of useful bits per second): 0.7081 (95% CI: [0.7075 – 0.7086])
- Avg. Transmission time: 14.1255 seconds

Experiment 4

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:10
- Bitstream Length: 20
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05

Result:

- BER: 0.0000 (95% CI: [0.0000 – 0.0000])
- Channel capacity (effective throughput — the number of useful bits per second): 0.6839 (95% CI: [0.6836 – 0.6842])
- Avg. Transmission time: 29.2458 seconds

Experiment 5

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.05
- 1 Bit Delay: 0.150
- Given Delay Threshold: 0.05

Result:

- BER: 0.0000 (95% CI: [0.0000 – 0.0000])
- Channel capacity (effective throughput — the number of useful bits per second): 1.9747 (95% CI: [1.9742 – 1.9753])
- Avg. Transmission time: 101.2846 seconds

Experiment 6

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.005
- 1 Bit Delay: 0.015
- Given Delay Threshold: 0.002

Result:

- BER: 0.0140 (95% CI: [0.0097 – 0.0183])
- Channel capacity (effective throughput — the number of useful bits per second): 17.7032 (95% CI: [17.583 – 17.824])
- Avg. Transmission time: 11.1626 seconds

Experiment 7

Variables:

- Processor Delay Mean: 0.05
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.05
- 1 Bit Delay: 0.15
- Given Delay Threshold: 0.05

Result:

- BER: 0.138 (95% CI: [0.115 – 0.160])
- Channel capacity (effective throughput — the number of useful bits per second): 1.7017 (95% CI: [1.669 – 1.734])
- Avg. Transmission time: 101.38 seconds

Experiment 8

Variables:

- Processor Delay Mean: 0.05
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.005
- 1 Bit Delay: 0.015
- Given Delay Threshold: 0.005

Result:

- BER: 0.4995 (95% CI: [0.496 – 0.503])
- Channel capacity (effective throughput — the number of useful bits per second): 1.8593 (95% CI: [1.823 – 1.896])
- Avg. Transmission time: 53.7152 seconds

Experiment 9

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.005
- 1 Bit Delay: 0.015
- Given Delay Threshold: 0.009 (Total Threshold 0.14)

Result:

- BER: 0.0685 (95% CI: [0.058 – 0.079]) (Since external delays added threshold less then 1 Bit Delay does not create significant erros)
- Channel capacity (effective throughput — the number of useful bits per second): 16.5481 (95% CI: [16.337 – 16.759])
- Avg. Transmission time: 11.2453 seconds

Experiment 10

Variables:

- Processor Delay Mean: 5e-6
- Iteration Count:5
- Bitstream Length: 200
- 0 Bit Delay:0.010
- 1 Bit Delay: 0.015
- Given Delay Threshold: 0.005

Result:

- BER: 0.106 (95% CI: [0.089 – 0.123]) (Nearly bit delays causes bigger BER)
- Channel capacity (effective throughput — the number of useful bits per second): 13.074 (95% CI: [12.87 – 13.28])
- Avg. Transmission time: 13.656 seconds

4 Optimization Options

To enhance the performance and increase the capacity of the covert timing channel, several optimization strategies can be considered:

- **Multi-bit Symbol Encoding:** Instead of encoding a single bit per delay interval, we can use multiple discrete delays to represent multi-bit symbols (e.g., 2-bit or 3-bit symbols). This significantly increases the channel capacity, provided that the delay values remain distinguishable under channel noise.
- **Parameter Sweeping:** A systematic set of experiments can be conducted to explore various combinations of 0-bit and 1-bit delays. The goal is to identify the pair that maximizes capacity while keeping the Bit Error Rate (BER) acceptably low. This can be further automated through grid search or optimization algorithms.
- **Error Correction Coding (ECC):** Implementing lightweight error correction schemes (such as Hamming codes or repetition coding with majority voting) can help reduce the impact of noise without drastically lowering throughput. This allows for a more aggressive capacity configuration while maintaining reliability.

5 Conclusion

In this assignment, I implemented a covert timing channel based on inter-packet delays (IPD) to transmit binary data between a sender and receiver operating across an insecure network. The sender encoded each bit by introducing a predefined delay between UDP packet transmissions, while the receiver decoded the message by measuring inter-arrival times and applying a fixed delay threshold.

To evaluate system performance, I conducted a series of controlled experiments by varying parameters such as 0-bit and 1-bit delay values, delay thresholds, and bit repetition lengths. For each configuration, I measured the Bit Error Rate (BER), effective channel capacity (in bits per second), and average transmission time. I also computed 95% confidence intervals to quantify variability across trials.

The results reveal a clear trade-off between reliability and capacity. Configurations with larger delay gaps and higher bit repetition achieved near-zero BER but incurred significantly higher transmission times and lower throughput. In contrast, configurations with minimal delay gaps or limited bit repetition produced higher capacities but were prone to BERs near 0.5, making them unreliable without additional correction. These issues were especially pronounced due to the random nature of the test bitstreams, which often contained balanced numbers of 0s and 1s. Threshold tuning was also critical; the optimal decision threshold was typically close to the midpoint between the 0-bit and 1-bit delays but varied depending on noise introduced by the processor.

Overall, the experiments demonstrate that with careful delay tuning and the possible integration of error correction mechanisms, a covert timing channel based on IPD can achieve a practical balance between stealth, reliability, and throughput under realistic operating conditions.

All of the experiments are made in sec environment's phase2_experiment_sender.py file.

6 GitHub Repository

Project code and this report are available at:
`https://github.com/ANILKE/middlebox/tree/phase2`

CENG 519 - Network Security - Project Phase 3 and 4 Report

Kemal Anıl Kekevi
2380608

June 15, 2025

Abstract

This report details the implementation and evaluation of a covert channel detector and mitigator aimed at identifying and disrupting Inter Package Delay based covert communication channels in network traffic. The covert channel detector uses heuristic-based techniques to detect suspicious timing patterns, while the mitigator introduces random jitter to disrupt covert communication. The report explains the implementation, advantages, limitations, and potential improvements for both components.

1 Introduction

Covert channels are communication methods that use legitimate network traffic to encode secret data. Detecting and mitigating these covert channels is a crucial task in network security. This report describes two key components designed for this task: a covert channel detector and a covert channel mitigator.

The detector identifies covert channels by analyzing inter-packet delays (IPDs)(my selected covert channel), which can be manipulated to encode information. The mitigator prevents such covert channels by introducing random jitter to the traffic, disrupting the regularity needed for covert communication.

2 Covert Channel Detector

2.1 Overview

The covert channel detector uses a heuristic approach to analyze packet arrival times and identify suspicious patterns in the inter-packet delays (IPDs). These patterns may indicate the presence of a covert channel, even when the exact encoding method is unknown.

2.2 Design and Implementation

The design of the covert channel detector is based on heuristic analysis of packet timing patterns, specifically Inter-Packet Delays (IPDs). The system works by observing the timing of packets over a sliding window and extracting statistical features to detect any abnormal patterns.

2.2.1 Core Components of the Detector:

- **Packet Timestamp Collection**: Packet timestamps are collected for each incoming packet. Processor takes the package arrival time and append it to collected package times in the detector.
- **Inter-Packet Delay (IPD) Calculation**: The time between consecutive packets is calculated as the inter-packet delay (IPD) in the detector.
- **Heuristic Analysis**: The detector calculates various heuristic scores based on IPDs, such as:
  - **Regularity**: Measures the consistency of IPDs using the coefficient of variation (CV).

$$\text{cv} = \frac{\text{std}(delays)}{\max(\text{mean}(delays), 0.001)}$$

  - **Entropy**: Measures the randomness of IPDs by calculating the entropy of the delay distribution.
  - **Bimodality**: Checks for bimodal distributions (indicative of binary encoding). The inter-packet delays (IPDs) are first rounded to the nearest 100 milliseconds to reduce noise and increase precision. If more than two unique delays are identified, the unique delays are sorted, and the two most frequently occurring delays are selected. The occurrence ratio of these two delays is calculated, where the ratio is defined as the minimum occurrence divided by the maximum occurrence. If this ratio exceeds 0.5, indicating a significant prevalence of the two delays, the value ratio is then computed, which is the ratio of the larger delay to the smaller delay. If the value ratio exceeds 2.0, and there is a significant time gap between these two delays (i.e., their difference is large), this suggests the presence of a covert channel encoding scheme. In such cases, a bimodal score is assigned, indicating that the delay distribution is likely bimodal, which is a characteristic of covert communication methods that use binary encoding with distinct short and long timing intervals.

  - **Pattern Matching**: Identifies specific patterns in delays that suggest covert communication.The most frequently repeated delays are identified as the min (shortest) and max (longest) delays. Then, the short delays (delays closest to the min) and long delays (delays closest to the max) are calculated. The ratio of occurrences between short delays and long delays is determined, and if this ratio exceeds a predefined threshold, the pattern is classified as a covert communication pattern.
  - **Baseline Deviation**: Compares the current delays with a historical baseline of normal network behavior by using mean and standard deviation.

- **Decision Making** The total score from all heuristics is compared against a threshold. If the score exceeds the threshold, the traffic is flagged as suspicious.

2.2.2 Implementation Details

The detector is implemented in Python, and the core of the system uses the following steps:

- **Data Collection** Timestamps for each incoming packet are collected.
- **IPD Calculation** The delay between each pair of consecutive packets is computed.
- **Heuristic Evaluation** Each heuristic score is computed based on IPDs, and a total score is generated.
- **Suspicion Decision** If the total score exceeds a predefined threshold, the packet stream is considered suspicious, and the covert channel is detected.

The implementation uses basic statistical methods, such as entropy and CV, along with more complex checks like bimodality and pattern matching. The system can be adjusted to detect a wide range of covert communication schemes without prior knowledge of the encoding method.

2.3 Pros and Cons

Pros:

- **Real-time detection** The detector operates in real-time, enabling timely detection of covert channels.
- **No prior knowledge required** The system does not require any knowledge of the specific covert channel encoding method, making it flexible and adaptive.
- **Adaptability** Can detect a wide range of covert channels by modifying the heuristics.

Cons:

- **False positives** The detector may flag legitimate traffic as suspicious if network traffic exhibits bursty or regular timing patterns.
- **Sensitivity to thresholds** The detector's performance is highly sensitive to the detection threshold, which may need tuning for different network environments.
- **Computational overhead** The use of statistical methods and real-time analysis may introduce some performance overhead, especially in high-speed networks.

2.4 How to Implement

To implement the detector:

- Capture packet timestamps from incoming network traffic.
- Calculate the inter-packet delay (IPD) between each packet.
- Apply heuristic checks:
  - Regularity: Calculate the coefficient of variation (CV) of the IPDs.
  - Entropy: Compute the entropy of the IPD distribution.
  - Bimodality: Detect if the distribution has two distinct peaks.
  - Pattern Matching: Check if the IPDs form regular patterns indicative of covert communication.
  - Baseline Deviation: Compare current delays with a historical baseline to spot deviations.
- Calculate the total score from all heuristics.
- Flag the traffic as suspicious if the total score exceeds the detection threshold.

3 Covert Channel Mitigator

3.1 Overview

The covert channel mitigator is designed to disrupt covert communication by introducing **random jitter**into packet timings. This jitter interferes with any regular timing patterns used by covert channels, rendering the communication unintelligible.

3.2 Design and Implementation

The mitigator works by adding **random delays**between packets to obscure any predictable timing. This approach disrupts the covert encoding method by making the packet timings more unpredictable.

3.2.1 Core Components of the Mitigator:

- **Random Delay Generation** For each packet, the mitigator generates a random delay within a specified range.
- **Delay Introduction** The packet is held for the calculated delay before being forwarded to its next hop.
- **Jitter Range** The delay range can be tuned based on network conditions to balance between mitigation effectiveness and network performance.

3.2.2 Implementation Details

The mitigator is implemented in Python, where the core logic works as follows:

- For each detected packet, a random delay is calculated from a uniform distribution between a minimum and maximum delay.
- The packet is then held for the calculated delay before being forwarded to its destination.
- The delay range can be adjusted to optimize the tradeoff between mitigating covert channels and maintaining network performance.

3.3 Pros and Cons

Pros:

- **Simple and effective** The mitigation strategy is easy to implement and provides an effective way to disrupt covert communication.
- **Minimal setup** The only configuration required is the jitter range, which can be adjusted based on the network's needs.

Cons:

- **Impact on normal traffic** Introducing jitter can increase latency and reduce throughput for all traffic, which may affect time-sensitive applications.
- **Inefficiency in low-rate channels** For covert channels that are low-rate or highly irregular, the mitigator might add unnecessary delays.

3.4 How to Implement

To implement the mitigator:

- For each packet detected, generate a random delay within a predefined range (given as a argument to processor.)
- Introduce the delay by holding the packet before forwarding it.
- Adjust the delay range based on network performance requirements and the expected covert communication patterns.

4 Experimentation Campaign

Since the covert channel mitigator operates by introducing random delays only to the packets already detected as part of a covert channel, its evaluation metrics, such as F1 score, true positives, true negatives, false positives, and false negatives, remain identical to those of the detector. Therefore, the primary additional metric for the mitigator is the capacity of the covert channel. Other metrics are same for mitigator and detector.

Experiment 1

Test with low processor delay mean on processor, high 0 and 1 bit delays on sender and receiver, high confidence threshold. Variables:

- Processor Delay Mean: 5e-6
- 0 Bit Delay:0.3
- 1 Bit Delay: 0.9
- Given Delay Threshold: 0.3
- Minimum mitigation delay time: 0.6
- Maximum mitigation delay time: 1.6
- Detection window size: 30
- Detection score threshold: 0.69
- Example sent message bitstream: 01101000011001010110110001101100011011110010000000110011010000000111001001100100001
- Example received message bitstream: 01101111110000000110111111100000001011110010110000000011111100000011001011111111

Result:

- **True Positives (TP)**: 700
- **False Positives (FP)**: 0
- **True Negatives (TN)**: 1010
- **False Negatives (FN)**: 180
- **Precision**: 1 (95% CI: [1.0000 – 1.0000])
- **Recall**: 0.7985 (95% CI: [0.7723,0.8247])
- **F-Score**: 0.886 (95% CI: [0.8722,0.9027])
- **BER**: 0.2841 (95% CI: [0.2760, 0.2922])
- **Capacity**: 0.2477 (95% CI: [0.2385, 0.2569])
- **Transmission Time**: 254.39 (95% CI: [245.31, 263.47])

Experiment 2

Test with low processor delay mean on processor, low 0 and 1 bit delays on sender and receiver, high confidence threshold. Variables:

- Processor Delay Mean: 5e-6
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05
- Minimum mitigation delay time: 0.1
- Maximum mitigation delay time: 0.2
- Detection window size: 30
- Detection score threshold: 0.69
- Example sent message bitstream: 01010010011001010110001101100101011010010111011001100101001000000101010001101000011
- Example received message bitstream: 01010010011001010110001101100101011010010111011001100101001000000101010001101000

Result:

- **True Positives (TP)**: 0
- **False Positives (FP)**: 0
- **True Negatives (TN)**: 1153
- **False Negatives (FN)**: 520
- **Precision**: undefined (95% CI: [undefined, undefined])

- **Recall**: 0 (95% CI: [0, 0])
- **F-Score**: undefined (95% CI: [undefined, undefined])
- **BER**: 0.0 (95% CI: [0.0, 0.0])
- **Capacity**: 1.1644 (95% CI: [1.1644, 1.1644])
- **Transmission Time**: 75.58 (95% CI: [75.58, 75.58])

Experiment 3

Test with a low processor delay mean on the processor, low 0 and 1 bit delays on sender and receiver, lower confidence threshold. Variables:

- Processor Delay Mean: 5e-6
- 0 Bit Delay:0.1
- 1 Bit Delay: 0.2
- Given Delay Threshold: 0.05
- Minimum mitigation delay time: 0.1
- Maximum mitigation delay time: 0.2
- Detection window size: 30
- Detection score threshold: 0.49
- Example sent message bitstream: 01010010011001010110001101100101011010010111011001100101001000000101010001101000011
- Example received message bitstream: 01010010011001010110001101100101011010010111011001100101001000000101010001101000

Result:

- **True Positives (TP)**: 994
- **False Positives (FP)**: 0
- **True Negatives (TN)**: 1300
- **False Negatives (FN)**: 46
- **Precision**: 1.0 (95% CI: [1.0, 1.0])
- **Recall**: 0.9558 (95% CI: [0.9558, 0.9558])
- **F-Score**: 0.9777 (95% CI: [0.9777, 0.9777])
- **BER**: 0.5192 (95% CI: [0.5192, 0.5192])
- **Capacity**: 0.6256 (95% CI: [0.6256, 0.6256])
- **Transmission Time**: 79.93 (95% CI: [79.93, 79.93])

Experiment 4

Test with a higher processor delay mean on the processor, medium 0 and 1 bit delays on sender and receiver, medium confidence threshold. Variables:

- Processor Delay Mean: 0.1
- 0 Bit Delay:0.3
- 1 Bit Delay: 0.5
- Given Delay Threshold: 0.1
- Minimum mitigation delay time: 0.2
- Maximum mitigation delay time: 0.3
- Detection window size: 30
- Detection score threshold: 0.49
- Example sent message bitstream: 01010010011001010110001101100101011010010111011001100101001000000101010001101000011
- Example received message bitstream: 01111100001100111011111000100010110101111001111000010001011010000001010101110011

Result:

- **True Positives (TP)**: 1536
- **False Positives (FP)**: 213
- **True Negatives (TN)**: 854
- **False Negatives (FN)**: 24
- **Precision**: 0.878 (95% CI: [0.878, 0.878])
- **Recall**: 0.9838 (95% CI: [0.9838, 0.9838])
- **F-Score**: 0.9284 (95% CI: [0.9284, 0.9284])
- **BER**: 0.5673 (95% CI: [0.5673, 0.5673])
- **Capacity**: 0.2089 (95% CI: [0.2089, 0.2089])
- **Transmission Time**: 215.44 (95% CI: [215.44, 215.44])

Experiment 5

Test with a higher processor delay mean on the processor, medium 0 and 1 bit delays on sender and receiver, low confidence threshold. Variables:

- Processor Delay Mean: 0.1
- 0 Bit Delay:0.3
- 1 Bit Delay: 0.5
- Given Delay Threshold: 0.1
- Minimum mitigation delay time: 0.2
- Maximum mitigation delay time: 0.3
- Detection window size: 30
- Detection score threshold: 0.35

Result:

- **True Positives (TP)**: 1545
- **False Positives (FP)**: 339
- **True Negatives (TN)**: 651
- **False Negatives (FN)**: 15
- **Precision**: 0.820 (95% CI: [0.820, 0.820])
- **Recall**: 0.9894 (95% CI: [0.9894, 0.9894])
- **F-Score**: 0.8999 (95% CI: [0.8999, 0.8999])
- **BER**: 0.5192 (95% CI: [0.5192, 0.5192])
- **Capacity**: 0.6256 (95% CI: [0.6256, 0.6256])
- **Transmission Time**: 79.93 (95% CI: [79.93, 79.93])

5 Experiments Conclusion

The experimentation campaign demonstrated how different configurations of the covert channel detection and mitigation systems can significantly impact the system's performance. The detector and mitigator were evaluated under various conditions, including changes in processor delay, bit delays, detection thresholds, and mitigation delay times. The results show clear trends in the system's behavior, with certain configurations enhancing detection accuracy and covert channel mitigation, while others lead to trade-offs in performance metrics such as Precision, Recall, F1-Score, BER, Capacity, and Transmission Time.

Key Observations

- **Precision** remains high (approaching 1.0) in most experiments, indicating that the system is highly effective at correctly identifying covert channel traffic without misclassifying normal traffic as covert (i.e., low False Positives).
- **Recall** fluctuates based on the detection threshold, with higher thresholds typically leading to lower recall due to missed covert channel detections. For example, in Experiment 1 (high confidence threshold), recall is lower compared to Experiment 2 (lower threshold). This is because higher thresholds increase precision but may miss covert traffic, leading to False Negatives.
- **F1-Score** offers a balanced view of Precision and Recall, and it shows good performance across most experiments, particularly in configurations with balanced Precision and Recall.
- **BER** (Bit Error Rate) and **Capacity** also change based on the detection threshold, with lower thresholds allowing for more effective mitigation, but sometimes at the cost of decreased capacity or increased BER. For example, Experiment 4 shows a lower Capacity and higher Transmission Time than Experiment 1, reflecting how higher processor delays and bit delays can influence performance.
- **Transmission Time** tends to increase as mitigation strategies introduce additional delays to disrupt covert channels. This is observed in experiments with higher mitigation delay times (e.g., Experiment 4).

Why Values Change

The values change due to a combination of factors:

- **Threshold Adjustments**: As detection thresholds are adjusted (higher or lower), the balance between True Positives and False Negatives changes, which affects Recall and subsequently the F1-Score.
- **Mitigation Delay**: The introduction of random delays by the mitigator influences the Capacity and Transmission Time. While mitigation reduces covert channel capacity, it may also increase transmission times, adding overhead to legitimate traffic.
- **Bit Delays**: The 0 and 1 bit delays introduced by the sender and receiver also have an impact on the overall performance. High delays tend to make the covert channel easier to detect but also disrupt transmission time and throughput.
- **Processor Delay Mean**: Variations in processor delay mean also influence the system's ability to detect covert channels and mitigate them effectively. Lower processor delay results in quicker detection, while higher processor delays might introduce additional latencies.

In conclusion, these experiments demonstrate that both detection accuracy and mitigation effectiveness are highly sensitive to the configuration of system parameters, including delay values and thresholds. Balancing these factors is essential for achieving optimal performance in real-world covert channel detection and mitigation scenarios. The results underscore the importance of fine-tuning system parameters based on the specific context and goals, such as minimizing false positives or achieving low capacity for covert channels while minimizing overhead.

6 Conclusion

This report discussed the design, implementation, and evaluation of a covert channel detector and mitigator. The detector uses a combination of heuristic checks to identify covert channels by analyzing inter-packet delays and detecting anomalies in packet timing patterns. The mitigator introduces random jitter into the traffic to disrupt covert communication.

The experiments demonstrated that the detector effectively identified covert channels based on inter-packet delays (IPDs), achieving a high detection rate with minimal false positives. The mitigator successfully reduced the capacity of the covert channel by introducing random delays, which disrupted the timing pattern and caused the receiver to interpret all data as bit 1. However, this mitigation introduced some performance overhead on legitimate traffic due to the added delays. Since my covert channel implementation is based on IPD it is sending and receiving bits instead of bytes. If detector detecs 1/8 of bits in covert channel we can alter the received message.

7 Future Work

Future improvements could focus on:

- Enhancing the detector with **adaptive thresholds**based on network conditions for better accuracy.
- Developing more **targeted mitigation techniques** such as traffic shaping based on traffic classification.
- **Optimization of computational efficiency** to handle higher throughput in real-time applications.

8 GitHub Repository

Project code and this report are available at:
`https://github.com/ANILKE/middlebox/tree/phase3`