CENG519 - TPPhase2 Report

Denizcan Yılmaz
2444172

April 2025

1 Implementation

1.1 General Overview

The covert communication channel was implemented using the **optional fields of the IPv4 header**, specifically the **No Operation (NOP)** option byte. NOPs are one-byte padding fields (`0x01`) typically ignored by standard network processing, making them suitable for hiding information without disrupting protocol behavior.

Each data symbol, which is a fixed-length bit sequence, is encoded by inserting a corresponding number of NOP bytes into the IPv4 options field. For example, a symbol value of 5 results in five `0x01` bytes. Since the IPv4 header allows a maximum of 40 bytes of options, the upper limit for NOP-based encoding is 5 bits per symbol.

To transmit a message, the plaintext is first converted to a binary stream, segmented into fixed-width symbols based on the `nop_bits` parameter, and then each symbol is transmitted using one or more packets. The number of NOPs in the IP header encodes the symbol.

Two special markers are defined:

- **START marker**: $2^{\text{nop\_bits}}$
- **END marker**: $2^{\text{nop\_bits}} + 1$

These markers signal the beginning and end of each covert message.

1.2 Sender (`sec/sender_tpphase2.py`)

The sender is responsible for **encoding the message into IPv4 packets** and transmitting them to the receiver. The main components of the sender are:

1. **Bitstream Conversion:** The plaintext is converted into a binary stream and padded to align with the symbol width (`nop_bits`). The stream is then segmented into fixed-width symbols.
2. **NOP Encoding:** Each symbol is encoded by inserting the corresponding number of NOP bytes into the IPv4 options field. Padding is added as necessary to maintain 4-byte alignment.
3. **Packet Transmission:** For each symbol, the sender transmits a user-defined number of packets (`pps`, packets-per-symbol) to introduce redundancy. A delay (`delay`) is applied between each packet to control transmission timing.

4. **Markers:** The sender transmits 3 START packets before the message and 3 END packets after to help the receiver identify boundaries.

5. **Metrics Logging:** After each transmission, the sender logs transmission time, number of bits, throughput, and configuration parameters to `sender_log.csv`.

Command-line parameters used by the sender:

- `--message`: Text to encode
- `--nop-bits`: Bits per symbol (1–5)
- `--pps`: Redundancy level
- `--delay`: Inter-packet delay in seconds
- `--target-ip`, `--port`: Network destination
- `--repeat`: Number of message repetitions

1.3 Receiver (`insec/receiver_tpphase2.py`)

The receiver passively **sniffs UDP packets** and decodes the number of NOPs in each packet's IP header to reconstruct the original message. Its functionality includes:

1. **Sniffing:** The receiver uses Scapy to capture incoming packets on the specified interface and filters for a specific UDP port.

2. **State Machine:**
   - **Waiting State:** Waits for a START marker to begin message reception.
   - **Receiving State:** Captures symbol values, performs majority voting every `pps` packets, and ignores redundant START/END markers. Terminates on receiving the END marker.

3. **Symbol Decoding:** Performs majority voting across packets-per-symbol and reconstructs the binary stream.

4. **Bitstream Reconstruction:** Converts the binary stream back into characters and removes padding artifacts.

5. **Logging:** Logs the number of bits received, duration, throughput, and decoded message to `receiver_log.csv`.

6. **Watchdog Timeout:** Sets a per-run timeout to prevent indefinite waiting. The timeout is rearmed after each message to ensure robustness.

The receiver accepts the following **parameters**:

- `--nop-bits`: Bits per symbol
- `--pps`: Redundancy level
- `--port`, `--iface`: Network settings
- `--repeat`: Number of expected messages
- `--timeout`: Inactivity timeout per message

This architecture allows for systematic performance evaluation under varying network conditions and encoding strategies, which are logged in a structured format for post-experiment analysis and visualization.

2 Experiment and Data Collection

To evaluate the performance and capacity of the implemented covert channel, a series of controlled experiments were conducted using a predefined grid of parameter configurations. The experiments were designed to capture the effect of different encoding strategies and timing characteristics on the throughput and reliability of the channel.

2.1 Parameter Sweep

The following parameters were varied systematically:

- **NOP Mapping Bits** (`nop_bits`): {1, 2, 3, 4}
  Specifies the number of bits encoded per symbol. A symbol value is mapped to an equivalent number of NOP bytes in the IPv4 options field.
- **Packets Per Symbol** (`pps`): {1, 2, 3}
  Controls the redundancy level; each symbol is sent using multiple identical packets to support majority-vote decoding.
- **Inter-packet Delay** (`delay`): {0.05, 0.1} seconds
  Defines the time interval between consecutive packets, which affects both throughput and stealth.

The full parameter space was explored by performing an exhaustive grid search over the Cartesian product of these values, i.e., for each combination of (`nop_bits`, `pps`, `delay`). Each configuration was repeated **5 times** to capture variability and allow for statistical evaluation. Each run was conducted with the same message of **15 characters**. At the end, both files have **120 rows** of measurements.

2.2 Data Collection Procedure

For each configuration:

1. The **sender** was executed with the given `message`, `nop_bits`, `pps`, and `delay` parameters for a specified number of repetitions (`repeat`).
2. Simultaneously, the **receiver** was started with matching `nop_bits` and `pps` parameters and was configured to capture the same number of messages.
3. After each run, both sender and receiver appended a structured row to their respective CSV log files.

The receiver logged the collected fields to `receiver_log.csv` while the sender logged to `sender_log.csv`.

3 Analysis and Plotting

3.1 Message Transmission Accuracy

All messages transmitted through the covert channel were received correctly and completely. This results in a **100% message transmission accuracy** across all tested configurations. No message loss or corruption was observed during any of the experimental runs. This confirms the reliability of the protocol implementation under the tested network conditions.

3.2 Grouped Analysis by Encoding Bits (NOP bits)

The first analysis groups results by the number of bits encoded per symbol (NOP bits). Increasing this value allows more data to be embedded per packet, reducing the overall number of packets needed to transmit a message.

Table 1: Performance grouped by NOP bits

NOP bits	Average Duration (s)	95% CI (Duration)	Average Throughput (bps)	95% CI (Throughput)
1	30.10	(24.89, 35.30)	5.03	(4.07, 5.99)
2	15.42	(12.80, 18.04)	9.73	(7.90, 11.56)
3	10.51	(8.76, 12.27)	14.10	(11.54, 16.66)
4	8.08	(6.76, 9.40)	18.18	(14.95, 21.42)

The results show a clear improvement in throughput as NOP bits increase. This is expected, since more bits per symbol mean fewer packets are required. However, higher values increase header size and visibility.

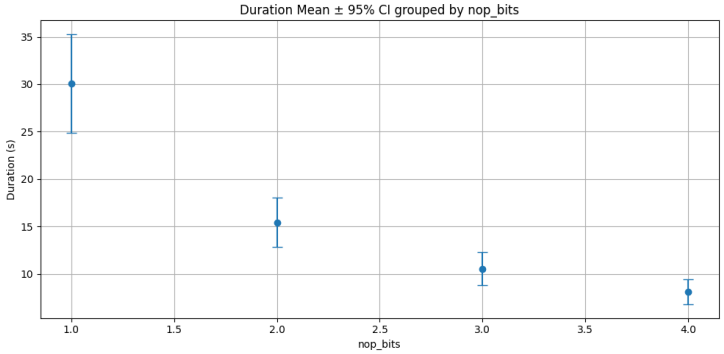

Figure 1: Mean Duration vs. NOP bits

3.3 Grouped Analysis by Redundancy Level (Packets per Symbol)

This analysis groups results by the packets-per-symbol (PPS) parameter, which controls redundancy. As expected, increasing redundancy reduces throughput but may increase resilience in noisier environments.

Table 2: Performance grouped by packets per symbol

Packets per Symbol	Average Duration (s)	95% CI (Duration)	Average Throughput (bps)	95% CI (Throughput)
1	8.38	(6.86, 9.90)	18.58	(15.79, 21.36)
2	16.06	(13.03, 19.09)	9.92	(8.37, 11.47)
3	23.64	(19.13, 28.16)	6.79	(5.71, 7.86)

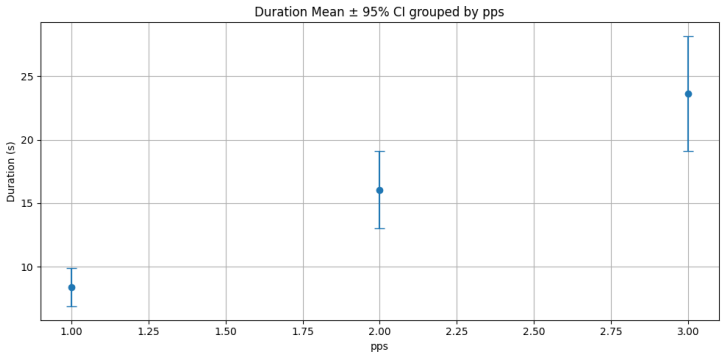

Figure 2: Mean Duration vs. PPS

The results confirm the expected inverse relationship between redundancy

and throughput. Even though message reliability was 100%, higher PPS configurations can still be useful in less controlled environments.

3.4 Grouped Analysis by Inter-Packet Delay

The final grouping examines the effect of inter-packet delay. Shorter delays allow higher throughput, as expected.

Table 3: Performance grouped by inter-packet delay

Delay (s)	Average Duration (s)	95% CI (Duration)	Average Throughput (bps)	95% CI (Throughput)
0.05	12.71	(10.41, 15.01)	14.20	(11.91, 16.48)
0.10	19.34	(15.84, 22.84)	9.32	(7.83, 10.82)

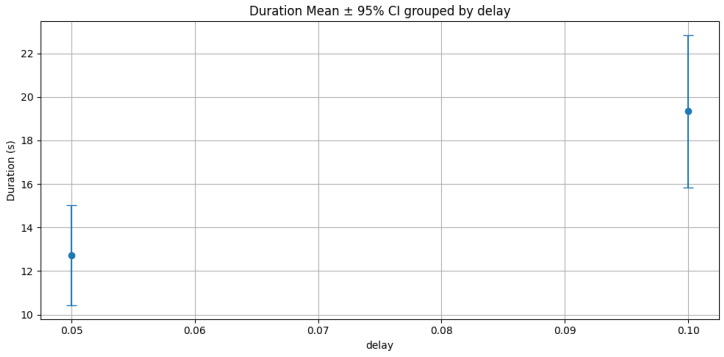

Figure 3: Mean Duration vs. Inter-Packet Delay

4 Discussion and Future Work

While the implemented covert channel demonstrates high reliability and adjustable capacity under controlled conditions, several limitations and opportunities for improvement remain.

- **Detection Susceptibility:** The use of IPv4 option fields, especially with a high number of NOPs, increases the risk of detection by intrusion detection.
- **Timing Sensitivity:** The inter-packet delay plays a crucial role in determining both throughput and stealth. However, it is sensitive to network jitter, which could reduce decoding reliability.
- **Advanced Encoding Schemes:** Future versions could explore alternative encodings (e.g., bitwise NOP encoding, multiple header options, or using padding fields in other protocols like TCP or ICMP).

- **Robustness Improvements:** Adding error correction, adaptive redundancy, or synchronization recovery would make the system more reliable in dynamic or lossy environments.

5 Project Repository

`https://github.com/denizcan-yilmaz/middlebox`

The repository contains the implementation, Docker configuration files, and experimental data files (`receiver_log.csv` and `sender_log.csv`) located under the `tpphase2_report/data` directory. Additionally, the report and the Python analysis script used for generating plots and evaluating results are included in the `tpphase2_report` directory.