Implementing Covert Channel using TCP Window Size

Mustafa Atahan Tanrıkulu
CENG519 - Network Security

April 2025

Abstract

This report explores the implementation of a covert communication channel using TCP packets over a custom network topology. The covert message is transmitted using modifications to the TCP window size field. We discuss the design, limitations, and experimental analysis of this approach, including practical issues with the TCP handshake under raw socket and containerized environments.

1 Introduction

Covert channels allow information to be communicated in ways that violate security policies or escape traditional detection mechanisms. This project focuses on a covert channel that hides data in the TCP header—specifically in the `window size` field—using raw socket programming in a controlled Dockerized setup.

2 System Design

The experiment consists of a sender, a receiver, and a network switch simulated via Docker containers. Packets are sent using raw sockets without kernel TCP stack involvement, allowing full control over header fields.

- **Sender:** Injects crafted TCP packets using ‘socket(AF_INET, SOCK_RAW, IPPROTO_TCP)‘.
- **Receiver:** Listens using raw sockets and logs packet contents.
- **Middlebox:** Simulates an insecure network path, initially unused in logic but available for expansion.

3 Packet Transmission Method

Although the system was designed to simulate a full TCP 3-way handshake for synchronization and stealth, various low-level network constraints prevented full implementation.

1. The sender transmits a SYN packet with spoofed headers using raw sockets.
2. The receiver listens for SYN packets and responds with a SYN-ACK manually crafted and sent using raw sockets.
3. The sender is intended to detect this SYN-ACK and reply with an ACK to complete the handshake. However, in practice, this step failed due to several challenges:
    - The kernel likely dropped spoofed SYN-ACK or ACK packets.
    - Docker's virtual networking may have interfered with raw socket transmission or reception of crafted packets.
    - Despite attempts to disable TCP resets and checksum offloading using `iptables` and `ethtool`, the issue persisted.
    - We verified behavior using extensive `tcpdump` traces and manually analyzed header bytes and flags.

Due to these constraints, we opted to skip the final ACK and proceed directly to covert transmission after SYN/SYN-ACK exchange.

Covert Channel Design:

Data transmission occurs by embedding information in the TCP window size field of ACK packets. Instead of using the raw window size directly, we send data using:

```
raw_message = payload_char + fixed_offset
```

The receiver decodes it using:

```
payload = window_size - fixed_offset
```

This adjustment provides better flexibility and avoids low window values that might be flagged or misinterpreted by intermediate networking devices.

Each character in the covert message is sent using one crafted ACK packet. The transmission ends with a special EOF marker (ASCII 0x04), after which the receiver writes the collected message to disk.

4 Experimental Results and Analysis

To assess the behavior and stability of our covert channel, we conducted systematic experiments across a variety of transmission delays using a Bash automation script.

Experimental Setup

We used the following Bash script to automate parameter sweeps:

```
#!/bin/bash
DELAYS=(0.01 0.05 0.1 0.2 0.4 0.8)
REPEATS=(1)
TEXT_FILE="covert_text.txt"
mkdir -p sender_results

for delay in "${DELAYS[@]}"; do
  for repeat in "${REPEATS[@]}"; do
    LOGFILE="sender_results/log_delay${delay}_rep${repeat}.csv"
    ./sender "$TEXT_FILE" --delay="$delay" --repeat="$repeat" --logfile="$LOGFILE"
  done
done
```

Each run sent a 256-byte covert message with varying delay parameters (from 10ms up to 800ms), using exponentially distributed inter-packet delays to simulate realistic timing behavior.

The receiver logged each received byte's arrival time and value. After transmission completion, logs were processed using Python scripts to analyze interarrival times, character frequency, and signal timing patterns.

Rolling Average of Interarrival Times

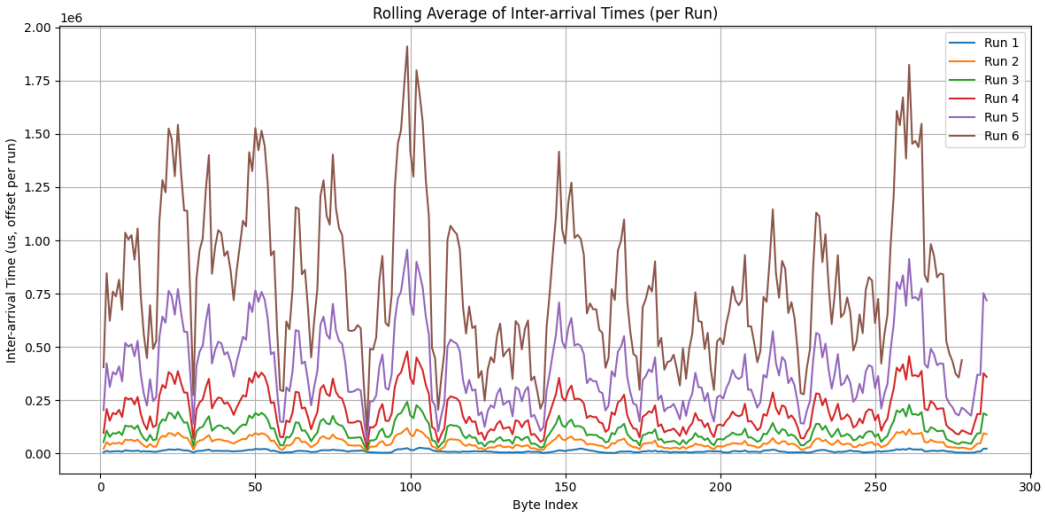

Figure 1: Rolling average of interarrival time (offset by first packet)

Interarrival Offsets per Run

Figure 2: Interarrival offsets by character index for each run

Overall Interarrival Histogram

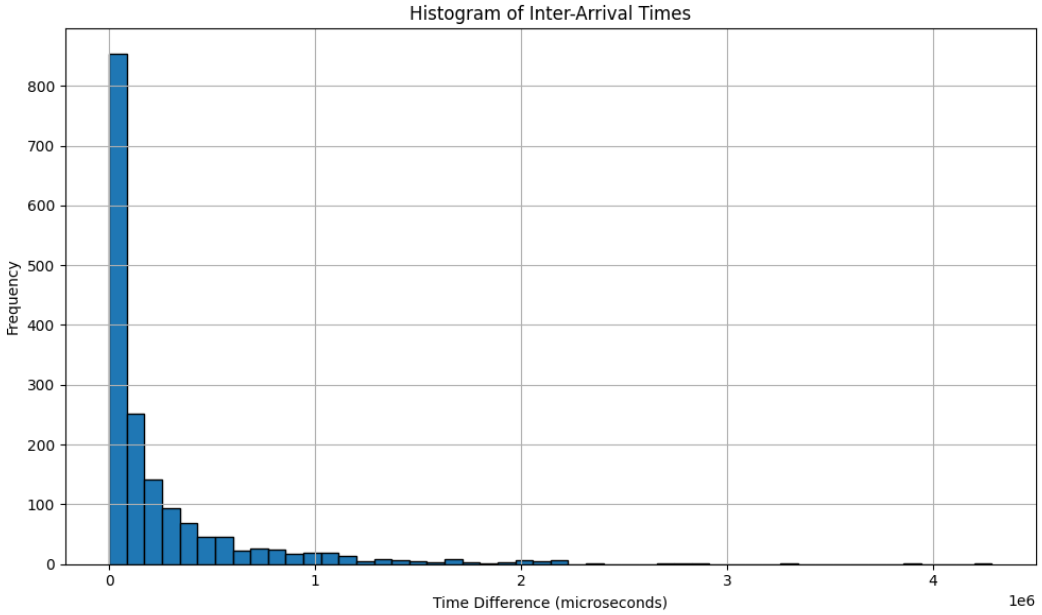

Figure 3: Histogram of all interarrival times

Character Frequency

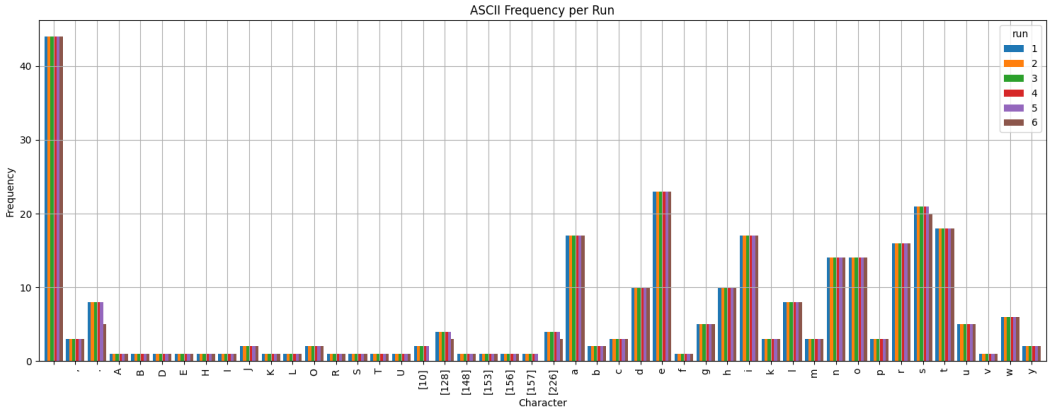

Figure 4: Character frequency distribution in covert text

5 Discussion

The plots show consistent transmission behavior across runs with low timing jitter. The histogram indicates mostly uniform transmission delays, validating the exponential inter-packet delay model. Character frequency is as expected from the Star Wars-inspired covert message.

Despite not completing a full 3-way handshake, the covert channel operated effectively. The workaround involved skipping the final ACK and continuing with ACK packets containing data.

6 Future Work

Several directions can improve stealth, compatibility, and realism:

- Implement real TCP with a proper 3-way handshake using a kernel bypass or custom kernel module.
- Increase stealthiness by varying encoding schemes or hiding data in timing or IP header fields.
- Better mimic real TCP traffic behavior to avoid detection, including proper header field values and window size ranges.

7 Conclusion

This experiment demonstrated a TCP-based covert channel operating via window size manipulation. Although technical barriers prevented a complete handshake implementation, the system successfully transmitted covert messages with precise timing and logging, providing insight into the challenges of raw packet crafting in virtualized environments.

References

[1] Beej's Guide to Network Programming, `https://beej.us/guide/bgnet/`