Term Project Phase 2

CENG519 - Network Security

Covert Channel via Modify IP Fragmentation fields: Experimental Report

Furkan Baran Aksakal
Student ID: 2448082

1. Objective

The purpose of this experiment is to design and evaluate a covert channel using IP fragmentation fields, specifically the fragment offset field. The channel is implemented in a parametrized fashion, enabling configuration of key parameters such as:

- Bits per packet (1 to 8 bits)
- Message length (randomized messages of specified length)
- Inter-packet delay (in seconds)
- Key value for basic obfuscation

The performance of this channel is analyzed by measuring:

- Average time to receive full messages
- Throughput (capacity in bits/second)
- 95% confidence intervals for timing measurements

2. Methodology

Implementation Details

- The sender encodes a binary message bitstream into the IP fragment offset field.
- A modular key-based transformation is applied to obfuscate the bit value.
- The receiver reverses the transformation to recover the message.
- Message completion is detected via a fixed end-of-message marker (`\x04`).

Using the Fragmentation Field for Covert Communication

The IP fragmentation offset field is a 8-bit field within the IP header that typically indicates the position of a fragment in the original data payload. However, in this covert channel, we repurpose this field to embed covert bits. Since the field is preserved in transit and not inspected by most network security tools, it is well-suited for hidden communication.

To send covert data, we split the message into bits and encode these bits directly into the fragment offset field. The receiver then extracts the value from the same field and decodes it into the original bitstream. This method does not require actual packet fragmentation — we simply set the field with our covert bits.

Encoding and Decoding Technique

Encoding:

```
encoded = (int(bits_str, 2) + key) % (2 ** bits_per_packet)
```

Decoding:

```
decoded_value = (frag - key) % (2 ** bits_per_packet)
bits = format(decoded_value, f'0{bits_per_packet}b')
```

This ensures that the covert bits are transformed during transmission and safely restored at the receiver using modular arithmetic. The key and bits-per-packet values are agreed upon by both sender and receiver.

Parametrization

- `--bits`: Number of bits encoded per packet (1 to 8)
- `--delay`: Time delay between packets (e.g., 0.001 to 0.1 seconds)
- `--key`: Integer key used to shift encoded bits
- `--message`: Message to send via covert channel
- `--message-length`: Number of characters to send via covert channel (random alphanumeric); if this parameter is enabled, the –message parameter is overrided, and random message is sent. Purpose of this parameter is to create message that is includes higher number of char.

Metrics Measured

- Time to transmit and decode full message (in seconds)
- Effective capacity: Total bits sent / transmission time
- Repeated runs (n=5) per setting for statistical robustness

3. Experiment Design

Controlled Parameters

- Message Length: 25 characters

- Bits Per Packet: 1 to 8
- Delay: 0.01 seconds (constant)

Measured Across

- 5 trials per configuration

Output Collected

- Duration per transmission
- Throughput per configuration
- Confidence intervals for time per configuration

4. Results

Bits	Delay	Mean Time (s)	95% CI (s)	Capacity (bps)
1	0.01	6.386	(6.349, 6.423)	31.32
2	0.01	3.137	(3.106, 3.168)	63.76
3	0.01	1.990	(1.965, 2.015)	100.50
4	0.01	1.560	(1.538, 1.582)	128.21
5	0.01	1.193	(1.172, 1.214)	167.64
6	0.01	1.034	(1.015, 1.053)	193.42
7	0.01	0.882	(0.866, 0.898)	226.76
8	0.01	0.702	(0.687, 0.717)	284.90

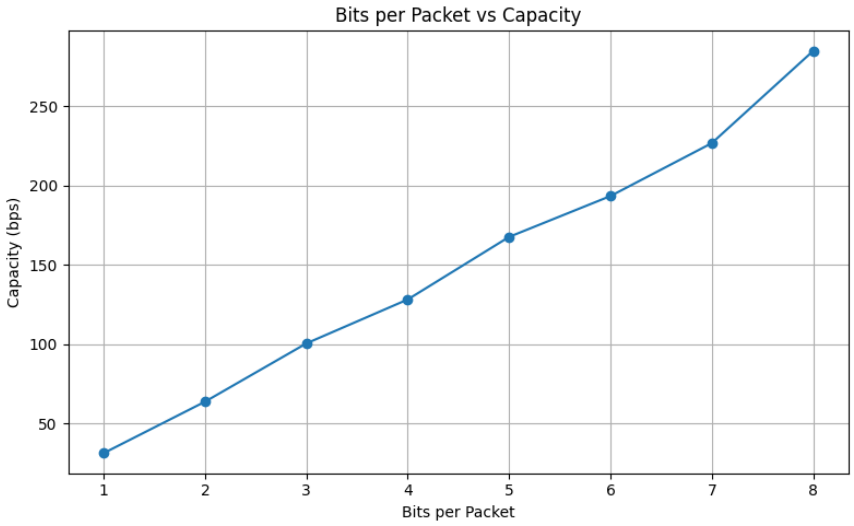

Figure 1: Bits per Packet vs Capacity

Notes:

- Each message consists of 25 characters = 200 bits.
- Capacity is computed as `200 / mean time (bps)`.
- Confidence intervals reflect 95% certainty based on 5 trial samples.
- Higher bits-per-packet result in significantly faster transmission and increased throughput.
- Higher bits-per-packet significantly increases the risk about detection. For example, you can choose 1 bit for decreasing risk.

Additional Delay Sensitivity Campaign (Bits = 2)

Delay (s)	Mean Time (s)	95% CI (s)	Capacity (bps)
0.01	3.153	(3.065, 3.241)	63.43
0.05	7.491	(7.389, 7.593)	26.70
0.10	12.497	(12.383, 12.612)	16.00
0.25	27.803	(27.723, 27.884)	7.19
0.50	53.776	(53.645, 53.907)	3.72
1.00	104.996	(104.921, 105.070)	1.90

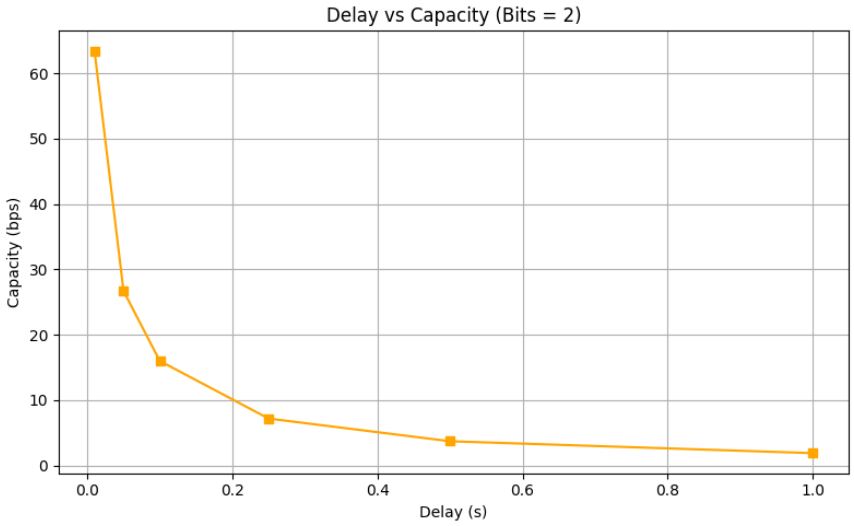

Figure 2: Delay vs Capacity (Bits = 2)

Note: In the delay sensitivity experiment, the number of bits per packet was fixed at 2 while delay values were varied. The results clearly demonstrate an inverse relationship between delay and covert channel capacity.

5. Conclusion

The experiment confirms that IP fragmentation fields can be effectively used as a covert channel with tunable capacity. Performance is directly influenced by `bits-per-packet` and inter-packet `delay`.

- With 8 bits per packet and a minimal delay of 0.01s, the channel achieved its highest throughput, reaching approximately 285 bps. However, this also increases the risk of detection due to the predictability and frequency of modifications.
- The throughput of the covert channel scales non-linearly with the number of bits encoded per packet. This indicates diminishing returns after a certain point, suggesting a practical balance must be struck between performance and stealth.
- As the delay between packet transmissions increases, the overall capacity of the channel decreases proportionally. This is expected and provides a tunable parameter for hiding transmission patterns in background traffic.
- The parametric design enables flexibility in configuring the system to match specific threat models or operational constraints. By adjusting parameters such as delay, bits per packet, and key offset, the covert channel can be optimized for either stealth or bandwidth.

This covert channel provides a strong foundation for further research into detection avoidance, noise resistance, and performance under real-world network conditions.

How to run receiver in insec as a example: python3 receiver.py –bits 2 –key 2
How to run sender in sec as a example: python3 sender.py –bits 2 –key 2
Every parameter has default value, but –key and – bits are important. These parameters should be same for receiver and sender as above example. Key is integer key used to shift encoded bits and parameter of bits is bits per packet.