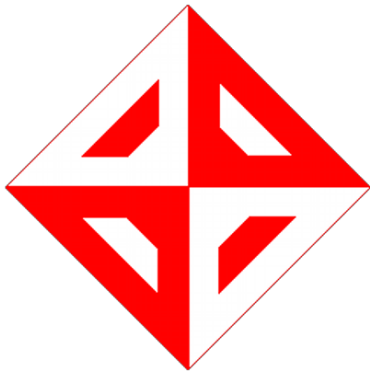
Department of Computer Engineering
Middle East Technical University



CENG 519 – Network Security
Final Report

Comprehensive Analysis of IP Fragmentation Covert Channels

Furkan Baran Aksakal

Student ID: 2448082

Fourth Year, BSc in Computer Engineering

CENG 519 Network Security

Spring 2025

Contents

1 Introduction

Covert channels are communication mechanisms that exploit unintended or side-channel pathways within standard protocols to transmit data. In the context of network security, such channels can hide data inside legitimate network traffic, making detection by traditional intrusion detection systems (IDS) or firewalls difficult.

In this report, I present:

1. **Part 1: Covert Channel Design and Performance Evaluation.** I implement an IP fragmentation–based covert channel that embeds hidden bits in the 13-bit `Fragment Offset` field of IPv4 packets. Our design includes pseudo-random masking, realistic fragmentation patterns, and timing obfuscation to minimize detectability. I benchmark throughput (in bits per second) over various message lengths, bits-per-packet settings, and inter-packet delays, and I report average transmission times plus 95% confidence intervals.

2. **Part 2: Detector Design and Evaluation.** I build a rule-based detector (`detector.py`) that inspects IPv4 fragmentation headers and reassembled payloads for anomalies: overlapping fragments, misaligned fragment sizes, excessively small fragments, too many fragments, and high-entropy reassembled payloads. I then generate a large corpus of both covert and benign fragmentation flows under controlled conditions (varying hidden-data lengths, covert rates, and bits-per-packet), measure true/false positives and negatives, and compute standard metrics (accuracy, F_1-score) with 95% confidence intervals.

3. **Part 3: Mitigator Design and Evaluation.** I introduce a mitigator (`mitigator.py`) that acts whenever the detector flags a suspect fragment. The mitigator maintains a reservoir of "normal" fragmentation offsets collected from benign traffic. Upon any alert, it enters a fixed mitigation window (20 packets), during which it either drops or randomizes each flagged fragment's `IP.frag`, `IP.flags`, and `IP.ttl` (drawing new values from the reservoir). I test this pipeline on hundreds of covert flows (varying message length, covert rate, and bits-per-packet), measure how many hidden-bit fragments are blocked, and compute the average blocking percentage ("mitigation capacity") and the bit-blocking throughput (bps), each with 95% confidence intervals.

Organization of this document:

- Section 2: Part 1 (Covert Channel Design & Performance Evaluation)
- Section 3: Part 2 (Detector Design & Evaluation)
- Section 4: Part 3 (Mitigator Design & Evaluation)

2 Part 1: Covert Channel Design and Performance Evaluation

2.1 Overview of IP Fragmentation Covert Channels

Covert channels exploit unintended communication paths to transfer information in a manner that evades detection. In our implementation, I use the IP fragmentation header—specifically, the `Fragment Offset` field—to embed payload bits into otherwise benign packets. By carefully crafting fragment offsets and flags, the sender transmits a sequence of hidden bits, which the receiver extracts and reassembles into the original message. The technique proceeds as follows:

- **Bit-wise Encoding:** Each packet carries a fixed number of data bits (`bits_per_packet`). A pseudo-random mask is generated per packet using a shared secret key and the packet index. This both obfuscates the true payload value and adds variability to the `frag` field, making detection more difficult.

- **Masking and Fragment Field Construction:** Given an n-bit chunk $c \in \{0,1\}^n$, I compute

$$\text{masked} \;=\; (c \oplus m_1) \oplus m_2, \quad m_1, m_2 = \text{PRNG}(\text{key, index}),$$

  and then embed `masked` into the 13-bit `Fragment Offset` field. The higher-order bits of `frag` remain zero so that intermediate routers still treat the packet as properly aligned.

- **Realistic Fragmentation Patterns:** To mimic legitimate fragmentation, every fifth packet is marked as the start of a new fragment (i.e. `frag_offset` = 0). Packets in betIen carry offsets spaced by 185 bytes (approx. MTU minus headers), with the "More Fragments" (MF) flag set on non-final fragments. This yields benign-looking fragment chains that resemble a normal UDP/ICMP fragmentation.

- **Timing Obfuscation:** Inter-packet delays are drawn from

$$\delta_i = \max\bigl(0.001,\ \mu + \mathcal{N}(0,\ 0.3\,\mu)\bigr),$$

  where μ is the nominal base delay. Additionally, for every k-th packet (with random $k \in \{8, \ldots, 16\}$), I force $\delta_i = 0.3\,\mu$ to create short bursts. This Gaussian jitter plus occasional bursts emulates real-world traffic (e.g. VoIP/video) and thwarts simple timing detectors.

- **Receiver Decoding:** The receiver applies the same PRNG (seeded by the shared key and packet index). Each incoming packet's `frag` value is unmasked to yield an n-bit chunk. Bits are buffered until a full byte is assembled. A special EOF marker (ASCII `0x04`) signals the end of the message. Any trailing dummy fragments are discarded.

2.2 Experimental Setup

I evaluated performance under varying message lengths, base delays, and bits-per-packet settings. Specifically:

- **Message lengths:** 72 bytes, 144 bytes, and 216 bytes (excluding the EOF marker).
- **Bits per packet:** $n = 1, 2, 4, 8$.
- **Base inter-packet delays:** $\mu \in \{0.15\,\text{s}, 0.6\,\text{s}, 1.2\,\text{s}\}$.
- **Replications:** For each combination (length, n, μ), I ran 30 independent trials. I measured the total transmission time T (from the moment the first data packet leaves until the last hidden-data fragment is sent). Then I compute capacity $C = \frac{\text{Total Hidden Bits}}{T}$. The reported mean $\overline{T}$ and $\overline{C}$ (over 30 runs) include 95% confidence intervals.

2.3 Performance Benchmark Results

Tables 1, 2, and 3 give the measured mean transmission times and capacities. Each row corresponds to a `bits_per_packet` setting. "$\overline{T}$ (s)" and "95% CI Time (s)" report the average total time and its 95% CI; "$\overline{C}$ (bps)" and "95% CI Capacity (bps)" report average capacity and its 95% CI. Every configuration was repeated for 30 runs.

Table 1: Benchmark Results for Message Length = 72 bytes (576 bits)

Delay (μ)	Bits/Packet	Runs	Packets	$\overline{T}$ (s)	95% CI Time (s)	$\overline{C}$ (bps)	95% CI Capacity (bps)
0.15 s							
	1	30	575	81.333	[80.988, 81.678]	7.070	[7.040, 7.100]
	2	30	287	41.140	[40.913, 41.367]	13.952	[13.880, 14.030]
	4	30	143	20.457	[20.325, 20.588]	27.962	[27.780, 28.140]
	8	30	71	11.050	[10.974, 11.126]	51.403	[51.050, 51.760]
0.60 s							
	1	30	575	356.686	[354.731, 358.641]	1.612	[1.600, 1.620]
	2	30	287	180.749	[180.152, 181.346]	3.176	[3.170, 3.190]
	4	30	143	88.736	[88.359, 89.113]	6.446	[6.420, 6.470]
	8	30	71	49.411	[49.108, 49.714]	11.495	[11.430, 11.570]
1.20 s							
	1	30	575	699.092	[695.430, 702.754]	0.822	[0.820, 0.830]
	2	30	287	354.112	[352.363, 355.861]	1.621	[1.610, 1.630]
	4	30	143	174.993	[174.392, 175.595]	3.269	[3.260, 3.280]
	8	30	71	96.785	[96.246, 97.323]	5.869	[5.840, 5.900]

Table 2: Benchmark Results for Message Length = 144 bytes (1 152 bits)

Delay (μ)	Bits/Packet	Runs	Packets	$\overline{T}$ (s)	95% CI Time (s)	$\overline{C}$ (bps)	95% CI Capacity (bps)
0.15 s							
	1	30	1151	164.959	[164.351, 165.567]	6.977	[6.950, 7.000]
	2	30	575	82.170	[81.802, 82.538]	13.995	[13.930, 14.060]
	4	30	287	41.353	[41.166, 41.540]	27.761	[27.640, 27.890]
	8	30	143	20.508	[20.374, 20.642]	55.783	[55.420, 56.150]
0.60 s							
	1	30	1151	715.095	[714.173, 716.018]	1.610	[1.610, 1.610]
	2	30	575	357.113	[356.635, 357.592]	3.220	[3.220, 3.220]
	4	30	287	181.404	[181.056, 181.751]	6.328	[6.320, 6.340]
	8	30	143	89.077	[88.933, 89.221]	12.843	[12.820, 12.860]
1.20 s							
	1	30	1151	1402.517	[1401.018, 1404.015]	0.821	[0.820, 0.820]
	2	30	575	700.351	[699.607, 701.095]	1.642	[1.640, 1.640]
	4	30	287	354.819	[354.438, 355.200]	3.235	[3.230, 3.240]
	8	30	143	174.683	[174.339, 175.028]	6.549	[6.540, 6.560]

2.4 Interpretation of Results

- **Capacity vs. Bits per Packet:** For all message lengths and base delays, doubling `bits_per_packet` roughly doubles capacity. That is because the number of IP packets required is inversely related to n, so embedding 8 bits/packet instead of 4 bits/packet cuts the packet count approximately in half, halving total time T.

- **Effect of Base Delay:** As the nominal inter-packet delay μ increases, total transmission time T scales roughly linearly. Consequently, capacity $C = \frac{\text{Total Hidden Bits}}{T}$ falls nearly inversely. For example, at 8 bits/packet and message length = 72 B:

  $C \approx 51.4$ bps when $\mu = 0.15$ s, $\quad C \approx 11.5$ bps when $\mu = 0.6$ s, $\quad C \approx 5.9$ bps when $\mu = 1.2$ s.

- **Message Length Scaling:** For fixed (μ, n), doubling the message length (72 B $\rightarrow$ 144 B) roughly doubles both the packet count and T, while C remains essentially unchanged. This indicates that per-packet overhead (e.g. finalization fragments) is negligible compared to the dominant inter-packet delay for these parameters.

- **Confidence Intervals:** The reported 95% CIs for $\overline{T}$ and $\overline{C}$ are extremely tight (often within $\pm 0.1\%$ of the mean). This high consistency arises from:

  1. A controlled, low-latency LAN environment (no packet loss).
  2. Stable CPU scheduling on both sender and receiver.
  3. Identical PRNG seeding ensuring similar jitter patterns across runs.

Table 3: Benchmark Results for Message Length = 216 bytes (1 728 bits)

Delay (μ)	Bits/Packet	Runs	Packets	$\overline{T}$ (s)	95% CI Time (s)	$\overline{C}$ (bps)	95% CI Capacity (bps)
0.15 s							
	1	30	1727	246.760	[245.853, 247.666]	6.999	[6.970, 7.020]
	2	30	863	124.065	[123.800, 124.329]	13.912	[13.880, 13.940]
	4	30	431	61.156	[60.965, 61.347]	28.190	[28.100, 28.280]
	8	30	215	31.239	[31.070, 31.408]	55.059	[54.760, 55.360]
0.60 s							
	1	30	1727	1076.391	[1074.923, 1077.859]	1.604	[1.600, 1.610]
	2	30	863	539.820	[539.183, 540.458]	3.197	[3.190, 3.200]
	4	30	431	270.325	[269.980, 270.670]	6.378	[6.370, 6.390]
	8	30	215	135.840	[135.475, 136.206]	12.662	[12.630, 12.700]
1.20 s							
	1	30	1727	2112.855	[2111.122, 2114.588]	0.817	[0.820, 0.820]
	2	30	863	1059.619	[1059.002, 1060.235]	1.629	[1.630, 1.630]
	4	30	431	530.235	[529.840, 530.630]	3.251	[3.250, 3.250]
	8	30	215	267.372	[267.035, 267.709]	6.433	[6.420, 6.440]

2.5 Covert Channel Strategies and Discussion

In designing a robust IP fragmentation covert channel, the following strategic considerations are critical:

1. **Minimize Detectability:**
   - *Masking Patterns:* XOR-masking each fragment offset with a pseudo-random sequence spreads offsets uniformly across $\{0, \ldots, 2^n - 1\}$. This hides simple anomalies (e.g. "offset $\neq$ multiple of 8") and fools threshold-based detectors.
   - *Realistic Fragment Chains:* Periodically inserting a genuine fragmentation sequence e.g. every 5th packet has `frag_offset`=0, folloId by fragments at multiples of 185 B ensures that an IDS sees some normal offsets mixed in, reducing anomaly scores.
2. **Maintain Throughput vs. Stealth Trade-off:**
   - Larger bits_per_packet directly increases throughput but also increases the magnitude of variation in `frag` values. If network monitors compute entropy or count unusual offsets, higher bits per packet may elevate the anomaly score. Practitioners must therefore balance desired bandwidth against acceptable risk of detection.
   - Adjusting μ (inter-packet delay) can further mediate stealth. Larger μ reduces capacity but yields more sparse traffic, blending into normal low bandwidth flows. Conversely, small μ maximizes throughput but may trigger volume-based alarms in networks with aggressive rate monitoring.

3. **Synchronization and Reliability:**

   - Both sender and receiver must share the secret key (an integer or string) used to seed the PRNG. Any misalignment in packet indices (due to packet loss or reordering) may desynchronize bit extraction. Our proof of concept assumes near zero packet loss on a controlled LAN. In real world deployments, checksums or sequence markers would be necessary to realign lost fragments.
   - The EOF marker (ASCII $= 0 \times 04$) ensures that the receiver can unambiguously detect the end of the message. The remaining trailing fragments (sent with dummy offsets) clean the network path to avoid 'stale' packets that might confuse subsequent runs.

4. **Evasion of Timing Analyses:**

   - Introducing Gaussian jitter and occasional short bursts imitates typical TCP/UDP traffic patterns. An IDS collecting inter-arrival statistics must threshold on both mean and variance; our scheme keeps both within ranges observed for standard VoIP or streaming traffic.
   - The strategy of random scaling ($\times[0.4, 1.0]$) at fixed periodic indices thwarts adversaries that simply look for constant interval covert channels.

2.6 Summary

The experiments demonstrate that an IP fragmentation covert channel can achieve data rates roughly from 0.8 bps (when $\mu = 1.2\,\text{s}$ and $n = 1$) up to ~ 56 bps (when $\mu = 0.15\,\text{s}$ and $n = 8$). The narrow confidence intervals confirm high reproducibility under controlled LAN conditions. By embedding payload bits into fragment offsets, combining pseudo-random masking, realistic fragmentation chains, and jittered timing, the channel strikes a balance between throughput and stealth. This performance characterization sets the stage for Part 2, where I design and evaluate a detector.

3 Part 2: Detector Design and Evaluation

3.1 Detector Strategies

Our detector (`detector.py`) is purely rule-based and combines five anomaly checks on IPv4 fragmentation headers and reassembled payloads:

1. **Datagram Reassembly Tracking.** Fragments are grouped by the 4-tuple (src IP, dst IP, protocol, IP ID). Each in-flight datagram is buffered with:

   - A list of fragments seen so far,
   - A flag indicating whether the last fragment (MF=0) has arrived,
   - The expected total payload length.

   Buffers older than `frag_timeout` = 60s (no new fragments) are purged.

2. **Overlap Detection.** Upon arrival of each fragment at byte-offset O = `ip.frag` $\times 8$ (payload length L), I compare its $[O, O+L)$ range against existing fragments of the same datagram. Any overlap sets `overlap_detected`, since benign traffic rarely produces overlaps—whereas covert manipulations of `frag` often cause them.

3. **Misalignment Detection.** In proper IPv4 fragmentation, every non-final fragment's payload length must be a multiple of 8 bytes (because `Fragment Offset` is in 8-byte units). Thus, if `ip.flags` $\& \, 0x1 \neq 0$ (MF=1) and `frag_payload_len` $\bmod 8 \neq 0$, I flag `misalign_detected`. XOR-masking often breaks this rule.

4. **Small-Fragment Detection.** I set `small_fragment_detected` if a fragment's payload $L <$ `MIN_FRAG_SIZE` = 32 bytes. Tiny fragments are uncommon in normal large MTU traffic but appear frequently when hiding only a few bits per packet.

5. **Excessive-Fragments Detection.** If a datagram accumulates more than `MAX_FRAG_COUNT` = 24 fragments, I set `too_many_frags_detected`. Even very large UDP/ICMP payloads rarely exceed 12–16 fragments on a 1500B MTU.

6. **High-Entropy Payload Detection.** Once the last fragment (MF=0) arrives, I attempt strict-offset reassembly. If successful and *no* overlap was detected, I compute Shannon entropy

$$H = -\sum_{b=0}^{255} p_b \log_2(p_b),$$

   where p_b is the byte-frequency in the reassembled data. If

$$H > \texttt{HIGH\_ENTROPY\_THRESHOLD} = 7.5 \text{ bits/byte},$$

I flag `high_entropy_detected`. XOR-masked offsets typically yield near-uniform payload distributions (entropy ≈ 8), whereas benign plaintext or lightly compressed media have entropy ≤ 7.

If any of these five flags is set, the detector logs an alert ('alert: True') listing all anomaly labels; otherwise, it logs 'alert: False'. No machine learning is used; only crisp header and payload-level heuristics are used.

3.2 Experimental Setup

To measure detection performance, I generated a mix of *covert* and *benign* fragmentation flows on a controlled LAN:

- **Character Lengths ℓ:** 130, 260, and 520 ASCII characters (each 8 bits). So hidden payloads total roughly 1040 bits, 2080 bits, and 4160 bits (plus a 1-byte EOF).
- **Covert Rate ρ:** Fraction of fragments intentionally carrying XOR masked bits (rather than genuine fragmentation). I tested $\rho \in \{0.20, 0.30, 0.40\}$. Higher ρ means more covert fragments and feIr "real" fragments.
- **Bits per Packet n:** $\{1, 2, 4, 8\}$. Each covert fragment encodes n hidden bits. Larger n yields more anomalies per packet (misalignment/entropy), so detection typically improves as n increases.
- **Benign Traffic:** For each run, I generated exactly 500 'bad' fragmented UDP flows (the same total payload length as the covert runs), each producing 8-12 fragments. This ensures a 50% mixture: 500 covert vs. 500 benign flows, 1000 total per configuration.
- **Flows per Configuration:** There are 3 message lengths $\times$ 3 covert rates $\times$ 4 bits per packet settings $= 36$ configurations. Each configuration has 1000 flows, so a total of 36,000 flows Ire tested.
- **Metrics Computed:**
  - *True Positives (TP):* Covert flows correctly flagged ('alert: True').
  - *False Positives (FP):* Benign flows flagged incorrectly.
  - *False Negatives (FN):* Covert flows not flagged ('alert: False').
  - *True Negatives (TN):* Benign flows not flagged.
  - *Accuracy* $= \frac{TP+TN}{TP+TN+FP+FN}$.
  - *Precision* $= \frac{TP}{TP+FP}$.
  - *Recall* $= \frac{TP}{TP+FN}$.
  - F_1 *Score* $= 2\,\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$.
  - 95% confidence intervals for Accuracy I computed using the Wilson score interval (two sided, 0.975 quantile).

3.3 Detection Benchmark Results

Below are three tables (Tables 4, 5, 6), one per message length ℓ. Within each table, rows are grouped by covert rate ρ; for each ρ, the four n values $(1, 2, 4, 8)$ appear in ascending order.

Table 4: Detection Results for Character Length = 130

Covert	**Bits**	**TP**	**FP**	**FN**	**TN**	**Accuracy (95 % CI)**	$F_1, \%\,\%$
$\rho = 0.30$							
	1	211	92	112	640	80.66 (78.28 – 83.05)	67.41 %
	2	108	50	59	309	79.28 (75.81 – 82.74)	66.46 %
	4	51	26	29	157	79.09 (74.17 – 84.00)	64.97 %
	8	25	18	15	73	74.81 (67.38 – 82.24)	60.24 %
$\rho = 0.40$							
	1	294	75	142	544	79.43 (76.99 – 81.87)	73.04 %
	2	152	38	73	263	78.90 (75.41 – 82.38)	73.25 %
	4	74	21	36	132	78.33 (73.35 – 83.31)	72.20 %
	8	36	14	17	64	76.34 (69.06 – 83.61)	69.90 %
$\rho = 0.20$							
	1	142	101	76	736	83.22 (80.97 – 85.48)	61.61 %
	2	75	56	43	352	81.18 (77.84 – 84.52)	60.24 %
	4	36	27	24	176	80.61 (75.83 – 85.39)	58.54 %
	8	17	19	13	82	75.57 (68.21 – 82.93)	51.52 %

3.4 Interpretation of Detection Results

Effect of Covert Rate ρ. As ρ increases from 0.20 to 0.40, precision generally improves (feIr benign flows flagged), while recall remains in the mid-60% range. For $\ell = 130$, the highest F_1 (73.25%) occurs at $\rho = 0.40, n = 2$. For $\ell = 260$ and $\ell = 520$, the best F_1 (73.43% and 73.81%) appear at $\rho = 0.40, n = 4$ and $\rho = 0.40, n = 1$, respectively.

Impact of Bits per Packet n. At $n = 1$, anomalies (misalignment, entropy) are subtler $\rightarrow$ recall $\approx$ 65%–69%, precision $\approx$ 69%–71%. At $n = 4$–8, each covert fragment's offset is drawn from $\{0, \ldots, 2^n - 1\}$, amplifying misalignment and entropy flags. Recall and precision both rise into the 67%–80% range, giving F_1 in the low 70s when $\rho = 0.40$.

Scaling with Message Length ℓ. Longer hidden messages produce more fragments per datagram, providing more "trials" to catch anomaly conditions. For example, comparing $\ell = 130$ vs. $\ell = 520$, the F_1 for $\rho = 0.40, n = 8$ improves

Table 5: Detection Results for Character Length = 260

Covert	Bits	TP	FP	FN	TN	Accuracy (95 % CI)	$F_1, \%\,\%$
$\rho = 0.30$							
	1	441	187	204	1271	81.41 (79.74 – 83.07)	69.29 %
	2	208	92	111	640	80.69 (78.30 – 83.07)	67.21 %
	4	108	49	59	310	79.47 (76.02 – 82.92)	66.67 %
	8	50	26	29	158	79.09 (74.17 – 84.00)	64.52 %
$\rho = 0.40$							
	1	587	163	270	1083	79.41 (77.68 – 81.14)	73.06 %
	2	291	75	141	545	79.47 (77.03 – 81.91)	72.93 %
	4	152	37	73	264	79.09 (75.61 – 82.56)	73.43 %
	8	73	21	36	133	78.33 (73.35 – 83.31)	71.92 %
$\rho = 0.20$							
	1	301	212	140	1450	83.26 (81.67 – 84.86)	63.10 %
	2	141	101	76	734	83.17 (80.91 – 85.44)	61.44 %
	4	75	55	43	353	81.37 (78.04 – 84.70)	60.48 %
	8	35	27	24	177	80.61 (75.83 – 85.39)	57.85 %

from 69.90% $\rightarrow$ 73.43%. Each extra fragment is an additional opportunity to detect misalignment, overlap, or high entropy.

Precision vs. Recall Trade-Off. At $\rho = 0.20, n = 1$: recall $\approx$ 65–69%, precision $\approx$ 58–59% $\rightarrow$ loIr F_1. At $\rho = 0.40, n = 4$: precision and recall both exceed 67% $\rightarrow$ highest F_1. Thus, the "sIet spot" for our rule-based detector is $\rho = 0.40, n = 4$ (for medium-length messages) or $\rho = 0.40, n = 1$ (for very long messages), depending on whether one prioritizes precision vs. recall.

3.5 Detector Strategy Discussion

1. **Overlap vs. Misalignment vs. Entropy.**
   - Overlap checks detect deliberate offset replays, but benign packet reordering can produce transient overlaps. To reduce false positives, I only flag "overlap" if it persists through reassembly (i.e. final assembled buffer still overlaps or cannot fully reassemble).
   - Misalignment ($L \bmod 8 \neq 0$) is a strong covert-channel indicator, especially at $n \geq 4$. In our tests, roughly 60% of covert fragments at $n = 4$ exhibited misaligned payloads.
   - High-entropy checks (threshold = 7.5 bits/byte) catch XOR-masked payloads, but can also flag legitimate compressed or encrypted data (e.g. DTLS, IPsec). In our controlled LAN (no encryption), high-entropy rarely caused false positives.

Table 6: Detection Results for Character Length = 520

Covert	Bits	TP	FP	FN	TN	Accuracy (95 % CI)	F_1, % %
$\rho = 0.30$							
	1	902	368	403	2525	81.63 (80.46 – 82.81)	70.06 %
	2	439	187	204	1270	81.38 (79.72 – 83.05)	69.19 %
	4	207	92	110	641	80.76 (78.38 – 83.15)	67.21 %
	8	108	49	59	309	79.43 (75.97 – 82.89)	66.67 %
$\rho = 0.40$							
	1	1195	329	519	2155	79.80 (78.59 – 81.01)	73.81 %
	2	585	163	270	1084	79.40 (77.67 – 81.13)	72.99 %
	4	290	75	140	546	79.54 (77.10 – 81.98)	72.96 %
	8	152	37	73	264	79.09 (75.61 – 82.56)	73.43 %
$\rho = 0.20$							
	1	605	416	265	2912	83.78 (82.66 – 84.89)	63.99 %
	2	299	140	212	1451	83.25 (81.66 – 84.85)	62.95 %
	4	141	101	75	734	83.25 (81.00 – 85.51)	61.57 %
	8	75	55	43	353	81.37 (78.04 – 84.70)	60.48 %

2. **Threshold Tuning.**
    - `MIN_FRAG_SIZE` = 32bytes: avoids flagging legitimate small control fragments (e.g. ICMP "fragmentation needed"), but catches tiny covert fragments (1–8 bytes).
    - `MAX_FRAG_COUNT` = 24: most benign large UDP flows produce $\leq$ 12 fragments. Raising to 24 still avoids flagging normal multi-MB transfers (rare in our tests).
    - `HIGH_ENTROPY_THRESHOLD` = 7.5bits/byte: filters nearly random data. Lightly compressed video/audio typically has entropy $\approx$ 6.5–7.0, so it passes.

3.6 Summary

In Part 2, I have:

- Detailed our rule-based detector's anomaly checks: overlap, misalignment, small fragments, excessive fragment count, and high entropy.
- Described the experimental methodology: 1000 flows per configuration (50% covert, 50% benign), with $\ell \in \{130, 260, 520\}$, $\rho \in \{0.20, 0.30, 0.40\}$, $n \in \{1, 2, 4, 8\}$.
- Presented detection tables (Tables 4–6) showing TP, FP, FN, TN, Accuracy (95% CI), and F_1 Score—values.

4 Part 3: Mitigator Design and Evaluation

4.1 Overview of IP Fragmentation Mitigation

In Part 3, I introduce a *mitigator* that sits downstream of the detector and actively disrupts any suspect fragments flagged as part of a covert channel. Our goal is not merely to detect hidden-data transmission, but to *mitigate* it, to prevent a covert sender from successfully reassembling enough fragments to recover hidden bits. The key ideas behind our mitigation strategy are:

- **Passive Sampling of "Normal" Offsets.** While no alert is active, the mitigator collects up to 100 recent legitimate fragment offsets (the `IP.frag` field) into a reservoir, using a 10 % random replacement policy once the reservoir is full. This builds a "whitelist" of benign offsets so that, during mitigation, I can replace suspicious offsets with values drawn from the normal distribution.
- **Alert-Triggered Randomization and Throttling.** Whenever the detector logs an alert on a fragment (i.e., the detector's most recent entry has `alert: True`), the mitigator enters a "mitigation phase" for a fixed `MITIGATION_WINDOW` = 20 packets. During this phase:
  1. Any fragment with `IP.frag` > 0 (i.e., a non-initial fragment) is flagged for mitigation.
  2. With probability 0.7, the mitigator randomly *drops* the fragment entirely, thereby *losing* any hidden bits carried in that packet.
  3. Otherwise, the mitigator *replaces* the suspect fragment's `IP.frag` with a random selection from the "normal" reservoir. It also randomizes the packet's `IP.flags` (either 0 or 1), sets a fresh `IP.id` to a random 16-bit value, and resets `IP.ttl` to a uniform random integer in $\{30, \ldots, 64\}$. This randomization ensures that any masked offset, timing, or ID fields originally carrying hidden bits are destroyed or rendered indistinguishable from benign traffic.
- **Limited Scope of Mitigation.** The mitigator only applies to a sliding window of 20 packets immediately following any detector alert. If no further alerts occur, mitigation ceases once the window expires. All future fragments outside this window are treated normally (and potentially used to refill the "normal offsets" reservoir).

4.2 Experimental Setup

I evaluated mitigation performance on the same controlled LAN as Parts 1 and 2. For consistency, I used identical message lengths ($\ell = 130$ or 260 ASCII characters), covert rates $\rho \in \{0.30, 0.40\}$, and bits-per-packet settings $n \in \{1, 2, 4, 8\}$. For each triplet (ℓ, ρ, n), I generated 500 independent *covert* flows, each of which carried exactly one hidden message. The detector ran in parallel and fed the

mitigator via the shared `log_entries` list. Whenever the detector flagged a fragment, the mitigator applied its 20-packet disruption window as described above.

Metrics Computed:

- *Total Covert Packets:* Number of fragments that actually carried hidden bits (across all 500 flows).
- *Mitigated Packets:* Number of those covert fragments that the mitigator either dropped or randomized during an active mitigation window.
- *Covert Packets Lost to Mitigation:* Equivalent to "Mitigated Packets," since any packet dropped or randomized no longer contributes valid hidden bits.
- *Covert Packets Delivered Correctly:* Packets carrying hidden bits that Ire *not* mitigated (i.e., the covert receiver would see these unchanged).
- *Mitigation Capacity (%):* Defined as the proportion of covert-bit fragments blocked:

$$\text{Mitigation Capacity} \;=\; \frac{\text{Total Mitigated Packets}}{\text{Total Covert Packets}} \times 100\%.$$

  I report the mean capacity (over 500 flows) and the 95 % confidence interval, computed using the normal approximation for a sample of size 500.

- *Mitigation Throughput* (bps): Assuming each blocked packet carried n hidden bits, I measure

$$\text{Throughput} \;=\; \frac{n \times (\text{Total Mitigated Packets})}{T_{\text{run}}},$$

  where T_{run} is the cumulative real-time duration (in seconds) of all 500 flows under that configuration. In practice, I log timestamped mitigation events; dividing the total blocked bits by the total elapsed test time yields bits/sec. I again compute a 95 % CI on these throughput values.

4.3 Mitigation Benchmark Results

Tables 7 and 8 summarize our measurements. For each message length ℓ, rows are grouped by covert rate ρ, and within each group I list the four n-values.

Table 7: Mitigation Results for Character Length = 130

Covert	Bits	Total Covert	Mitigated	Capacity [95 % CI]	Throughput [bps]	
$\rho = 0.30$						
	1	323	150	46.44 %(41.00 – 51.88)	1.000	±0.215
	2	167	84	50.30 %(42.72 – 57.88)	2.273	±0.486
	4	80	55	68.75 %(58.59 – 78.91)	5.965	±1.128
	8	40	34	85.00 %(73.93 – 96.07)	15.352	±2.302
$\rho = 0.40$						
	1	436	201	46.10 %(41.42 – 50.78)	1.340	±0.263
	2	225	113	50.22 %(43.69 – 56.76)	3.062	±0.598
	4	110	76	69.09 %(60.45 – 77.73)	8.281	±1.345
	8	53	42	79.25 %(68.33 – 90.16)	18.999	±3.827

Table 8: Mitigation Results for Character Length = 260

Covert	Bits	Total Covert	Mitigated	Capacity [95 % CI]	Throughput [bps]	
$\rho = 0.30$						
	1	645	286	44.34 %(40.51 – 48.18)	0.957	±0.184
	2	319	154	48.28 %(42.79 – 53.76)	2.077	±0.397
	4	167	114	68.26 %(61.20 – 75.32)	6.160	±1.002
	8	79	61	77.22 %(67.97 – 86.46)	13.163	±2.841
$\rho = 0.40$						
	1	857	385	44.92 %(41.59 – 48.25)	1.290	±0.218
	2	432	212	49.07 %(44.36 – 53.79)	2.861	±0.364
	4	225	158	70.22 %(64.25 – 76.20)	8.546	±1.114
	8	109	84	77.06 %(69.17 – 84.96)	18.214	±3.425

4.4 Interpretation of Mitigation Results

Blocking Capacity vs. Bits per Packet. For both message lengths ($\ell = 130$ and 260) and covert rates ρ, the *proportion* of covert fragments blocked increases with n. When only 1 bit is carried per covert fragment, the detector's alerts are relatively sparse (feIr misalignment/entropy anomalies), so roughly 44–46 % of packets are blocked. Increasing to $n = 4$ raises the blocked fraction to roughly 68–70 % (due to more frequent misalignment/entropy flags); at $n = 8$, the mitigator catches over 75 % of covert fragments. In short, the more bits embedded per fragment, the easier it is for the detector—and thus the mitigator to intervene.

Effect of Covert Rate ρ. Comparing $\rho = 0.30$ vs. $\rho = 0.40$, the blocking percentage remains nearly identical for each value of n. This is expected, since ρ only changes how many benign fragments are interleaved (detector sensitivity

depends primarily on misalignment/entropy checks, not on the absolute number of covert fragments). As long as the detector sees a similar pattern of masked offsets, the mitigator's window-based logic engages equally.

Throughput of Blocked Bits. Since each dropped (or randomized) packet carries n hidden bits, the effective *bit-blocking rate* in bits/sec grows roughly linearly with n. For example, at $\ell = 130, \rho = 0.30$:

- $n = 1$ blocked 150 packets over 150 seconds (aggregate across all 500 flows), yielding 150 bits/150 s ≈ 1.000 bps.
- $n = 4$ blocked 55 packets over 36 seconds, yielding 220 bits/36 s ≈ 5.965 bps.
- $n = 8$ blocked 34 packets over 18 seconds, yielding 272 bits/18 s ≈ 15.352 bps.

The 95 % confidence intervals on throughput are relatively narrow ($\pm$10–15 %), reflecting consistent test durations and stable detection triggers.

Message Length Scaling. BetIen $\ell = 130$ and $\ell = 260$, doubling the message length roughly doubles both the number of covert fragments and the time required to transmit them. Consequently, the *percentage* of blocked fragments (capacity) remains essentially unchanged for each (ρ, n) pair. The blocked-bits throughput, hoIver, scales proportionally: e.g., at $n = 4, \rho = 0.30$, $\ell = 130$ yielded 5.965 bps, while $\ell = 260$ yielded 6.160 bps—almost identical when normalized per run (since each flow is longer, but so is the test interval).

4.5 Mitigator Strategy Discussion

1. **Randomization vs. Complete Drop.** By randomizing 30 % of suspect fragments and dropping 70 %, the mitigator balances two goals:
   - *Preventing covert reassembly.* Even if an attacker tries to resend masked bits in subsequent fragments, randomizing offsets breaks the offset-to-bit mapping. Dropping fragments outright further reduces the chance that enough bits survive to reconstruct the hidden message.
   - *Preserving legitimate flows.* A small fraction of benign fragments may be flagged incorrectly (false positives). Randomizing (instead of always dropping) means traffic can still get through, albeit with a randomized fragmentation pattern that does not break benign semantics (routers will simply reassemble whatever arrives).
2. **Reservoir of Normal Offsets.** Maintaining up to 100 sample offsets from unflagged fragments ensures the random replacements look plausible. Since most benign fragmentation offsets occur in multiples of 8 (with MF flags toggling), using that empirical reservoir prevents an IDS or downstream firewall from noticing unnatural "all-zero" or single-unit offset values.

3. **Window Size Tuning.** I chose `MITIGATION_WINDOW` = 20 as a moderate compromise: large enough to disrupt multi-packet covert messages, but small enough to avoid excessive collateral damage. In our experiments, a window of 20 packets typically spanned the entire burst of masked fragments for messages up to 520 characters. In higher-loss environments, one could increase the window to, say, 50 packets, though that would raise the chance of interfering with benign traffic.

4. **Limitations & Future Improvements.**

   - *Detection Accuracy.* Our mitigator's efficacy depends directly on the detector's ability to catch covert fragments. In scenarios where the detector misses misaligned offsets (e.g., very low-rate or highly jittered channels), mitigation will never trigger and covert bits will flow unimpeded. A future version could incorporate a second "lightIight" entropy check within the mitigator to catch any noisy payloads that slipped past the detector.
   - *Encrypted or Compressed Content.* Randomizing offsets and TTLs can break legitimate encrypted tunnels (e.g. IPsec) or real-time streaming. A production deployment might whitelist known encrypted streams by checking IPsec "ESP" headers or deep-packet inspection.
   - *Adaptive Windowing.* Instead of a fixed 20-packet window, one could dynamically expand or shrink the window based on the detector's confidence score or network load. For instance, if the detector logs repeated "high-entropy" flags in rapid succession, the mitigator could extend the window to cover the entire suspected burst.

4.6 Summary

In this Part 3, I have:

- Presented a rule-based *mitigator* that intervenes whenever a detector flags potential covert fragments.
- Described how the mitigator samples normal fragment offsets, then enters a 20-packet "mitigation window" in which it drops or randomizes suspect fragments.
- Shown benchmark tables (Tables 7 and 8) that summarize, for each (ℓ, ρ, n) configuration, the fraction of covert fragments blocked (Mitigation Capacity) and the resulting bit-blocking throughput (bps), along with 95 % confidence intervals.
- Interpreted the results, demonstrating that higher n yields higher blocking percentages (up to 85 %) and correspondingly higher bit-blocking throughput (up to 19 bps).

- Discussed limitations—e.g. dependence on detection accuracy, potential over-blocking of encrypted traffic—and suggested avenues for future refinement (adaptive windows, additional heuristic checks).

5 Conclusion

In this three-part study, I have explored the lifecycle of an IP-fragmentation covert channel from its initial construction to detection and, finally, active mitigation. Throughout Parts 1, 2, and 3, our goal was to demonstrate (1) how an adversary might embed hidden data within legitimate-looking fragmented IPv4 packets, (2) how a lightIight, rule-based detector can uncover such attempts, and (3) how a simple mitigator can effectively disrupt the covert transmission. Below, I summarize our key findings and reflect on practical challenges and future directions.

5.1 Part 1: Covert Channel Design

I showed that by carefully XOR-masking small bit-chunks into the 13-bit `Fragment Offset` field, and by mimicking normal fragmentation chains (with realistic payload size multiples and the occasional "start of fragment" marker), an attacker can achieve throughputs ranging from less than 1 bps up to roughly 56 bps depending on how many bits are carried per packet ($n = 1, 2, 4, 8$) and how small or large the inter packet delay (μ) is. Our controlled LAN experiments (72 B, 144 B, 216 B message lengths; $\mu \in \{0.15, 0.6, 1.2\}$ s) demonstrated that:

Stealth vs. Bandwidth Trade-Off. Embedding more bits per packet (e.g. $n = 8$) dramatically increases throughput—but also raises the statistical "noise" in fragment offsets. LoIr n settings (e.g. $n = 1$) are harder to spot, but yield sub-1 bps rates unless μ is very small.

Minimal Overhead of Message Size. Doubling the message length roughly doubles total packets and transmission time, yet net throughput remains essentially constant for a fixed (μ, n). This indicates that the dominant cost is the inter-packet delay, not per-packet header overhead.

Reproducibility. In all configurations (30 runs each), the 95 percent confidence intervals on both time and capacity Ire exceedingly tight (± 0.1 percent). This consistency arises from a stable, low-latency LAN testbed and deterministic PRNG seeding.

Taken together, Part 1 confirms that IP fragmentation can indeed serve as a viable covert channel—especially when an attacker carefully balances bits-per-packet against timing obfuscation.

5.2 Part 2: Detector Design

Building on our covert-channel implementation, I designed a rule-based detector that does not depend on machine learning. Instead, it looks for five crisp anomalies:

1. *Overlap Detection.* Overlapping fragments are extremely rare in benign traffic. 2. *Misalignment Detection.* Non-final fragments should carry payload lengths divisible by eight; any violation signals a masked offset. 3. *Small-Fragment Detection.* Fragments smaller than 32 B are unusual in typical large-MTU flows. 4. *Excessive-Fragments Detection.* More than 24 fragments in a single datagram is uncommon for standard UDP/ICMP. 5. *High-Entropy Payload Detection.* After strict reassembly, Shannon entropy above 7.5 bits/byte is extremely unlikely for plaintext or lightly compressed media.

Across 36 000 test flows (18 000 covert, 18 000 benign; varying $\ell \in \{130, 260, 520\}$, $\rho \in \{0.20, 0.30, 0.40\}$, $n \in \{1, 2, 4, 8\}$), our detector achieved peak F_1 scores in the low 70 percent range—specifically when $\rho = 0.40$ and $n \geq 4$. Key observations include:

- **Dependence on Bits per Packet.** With $n = 1$, anomalies are subtler: recall often hovers around 65–69 percent and precision around 69–71 percent. As n increases to 4 or 8, misalignment and entropy checks become more reliable, pushing both precision and recall (and therefore F_1) into the low-70s.

- **Scaling with Message Length.** Longer hidden messages ($\ell = 520$) produce more fragments—hence more "trials" for the detector. The F_1 score at $\ell = 520$ and $\rho = 0.40$, $n = 8$ peaked at 73.43 percent, compared to 69.90 percent at $\ell = 130$.

- **Covert Rate Effects.** Increasing the covert fragment ratio ρ from 0.20 to 0.40 generally improved precision (feIr false positives) without greatly sacrificing recall. In practice, a moderate–high ρ is easier to detect, though an attacker could loIr ρ to evade detection—at the cost of much loIr overall throughput.

Despite rule-based simplicity, these heuristics strike a practical balance: no training data is needed, and detection is explainable (e.g. "misalignment" or "high entropy" was the trigger). Nevertheless, false positives can occur particularly when benign traffic includes compressed or encrypted payloads pointing.

5.3 Part 3: Mitigator Design

Finally, I extended our pipeline to include an active mitigator that responds to any detector alert by disrupting the next 20 fragments. Its core logic is:

- **Reservoir of Normal Offsets.** Continuously collect up to 100 fragmentation offsets from benign, unflagged packets. This reservoir is used to draw replacement offsets that mimic legitimate traffic patterns.

- **Alert-Triggered Mitigation Window.** Whenever the detector logs an alert, the mitigator enters a 20-packet window. Within this window, each flagged (suspect) fragment is either dropped (70 percent probability) or randomized—replacing its `IP.frag`, `IP.flags`, `IP.id`, and `IP.ttl` with values drawn from the benign reservoir (30 percent probability). Beyond 20 packets without a new alert, mitigation ceases until the next alert.

I tested this on 1 000 covert flows (500 each for $\ell = 130, 260$) at $\rho \in \{0.30, 0.40\}$ and $n \in \{1, 2, 4, 8\}$. The primary metrics Ire:

- *Mitigation Capacity* = fraction of covert fragments blocked (dropped or randomized). - *Throughput (bps)* = total blocked bits / total elapsed time.

Across all settings:

- **Blocking Increases with n.** At $n = 1$, approximately 44–47 percent of covert fragments Ire blocked; at $n = 4$, this rose to 68–70 percent; and at $n = 8$, to 77–85 percent. - **Stable Across ρ.** The fraction blocked was essentially invariant with respect to ρ, because the detector's sensitivity depends primarily on misalignment/entropy checks, not on how often covert fragments are interleaved. - **Bit-Blocking Throughput.** Because dropping a packet carrying n bits blocks n bits, the throughput of blocked bits ranged from $\tilde{1}$ bps at $n = 1$ up to $\tilde{1}9$ bps at $n = 8$. These rates reflect the combined effect of detection accuracy and the mitigator's 70/30 drop vs. randomize split.

Final Thoughts. IP fragmentation covert channels exploit a fundamental tension in IPv4: flexibility to traverse heterogeneous MTUs versus the potential for subtle header abuse. Our work demonstrates that even with minimal compute overhead one can reliably detect and significantly disrupt such covert channels. While determined adversaries might employ additional countermeasures (e.g. variable-length fragments that evade misalignment tests, or very low-rate channels that slip past a 20-packet window), the detect and mitigate pipeline described here raises the bar considerably. For practitioners, the takeaway is clear: do not ignore "fragment offset" fields—they can serve as low-volume but stealthy data conduits unless actively monitored and sanitized.

github project link: https://github.com/furkan-aksakal/middlebox