TimeLSB: eBPF-based Covert Channel on TCP Timestamps

Yusuf Şahin
Dept. of Computer Eng. METU, Türkiye
Turk Telekom, Türkiye
yusuf.sahin@metu.edu.tr
yusuftalha.sahin@turktelekom.com.tr

Ertan Onur
Dept. of Computer Eng. METU, Türkiye
ARLEON, Türkiye
eronur@metu.edu.tr
ertan.onur@arleon.com.tr

Can Karacelebi
Dept. of Computer Eng.
METU, Türkiye
can.karacelebi@metu.edu.tr

***Abstract*—This paper presents TimeLSB, a covert channel built on TCP timestamps using extended Berkeley Packet Filter (eBPF) technology. The channel encodes information by modifying the least significant bit of timestamp values through a CRC32-based scheme in the Linux kernel, while a passive receiver reconstructs the hidden message. We implement and evaluate this channel under a range of controlled network impairments, demonstrating that embedding covert bits does not significantly alter the behavior of normal TCP connections. The eBPF implementation introduces only on average 250ns of processing overhead per packet, which corresponds to several million packets per second and is negligible compared to typical forwarding performance. Finally, we analyze detectability using the distinct-timestamp ratio, showing that while absolute values differ from prior work, the metric still provides a stable separation between covert and normal flows. These results highlight the practicality of an eBPF-based covert channel and provide defensive insights for network security operations.**

***Index Terms*—covert channel, eBPF, tc, TCP, TCP timestamp option, in-kernel packet processing.**

I. Introduction

Covert channels are communication mechanisms that operate outside standard network protocols, enabling hidden data transmission designed to evade detection. Unlike traditional security approaches that focus on protecting the content of communication without concealing its existence, covert channels aim to hide the very act of communication itself. Lampson first introduced the concept, defining a covert channel as one not originally intended for information transfer [1]. With the growth of the Internet, protocols and systems across the stack—TCP/IP, wireless PHY/MAC, IoT stacks, and even distributed ML—have been exploited to implement covert channels. Broadly, these methods fall into two categories. In timing channels, covert data is modulated into packet timing (e.g., encoding presence/absence within time slots [2]). In storage channels, header or metadata fields are altered to encode hidden information that the receiver later extracts. Recent work underscores the breadth of storage channels: at the physical layer, CloakLoRa embeds covert data by amplitude-modulating LoRa chirps while leaving CSS frequencies intact [3]; at the MAC layer, randomized IEEE 802.11 MAC addresses are repurposed to convey hidden bits [4]; at the system/application layer, federated learning updates are subtly poisoned to create a one-bit covert channel per training round [5]; and in IoT settings, address/port encodings yield practical channels over TCP/IP and ZigBee without inter-packet delays [6]. Even small manipulations of header fields can cause significant information leakage over time [7], underscoring the importance of studying these channels.

Recent work has demonstrated TCP timestamp–based covert channels [8], showing that the timestamp option offers an attractive carrier. Our work builds on this idea of timestamp embedding but differs in two key ways. First, unlike prior work that relied on user-space tools such as Scapy [8] to modify packets after they left the stack, we implement the covert channel directly in the kernel using eBPF. This avoids extra copying and user/kernel context switches, making the approach more efficient and less intrusive. Second, while prior covert channel studies have proposed in-kernel implementations via kernel modules, such as Rutkowska's ISN-based covert channel [9], these approaches have drawbacks: they lack safety guarantees, risk destabilizing the kernel, and are cumbersome to develop. In contrast, eBPF provides a safe, sand-boxed environment for in-kernel packet manipulation, with dynamic loading and verification that make deployment more flexible and maintainable.

Although some work has explored using eBPF for covert channel detection, for example in IPv6 steganography monitors [10], to the best of our knowledge no prior research has implemented a covert channel itself with eBPF. While the concept of a timestamp-based covert channel was outlined by Giffin [11], we present an alternative implementation that differs in its encoding scheme and leverages eBPF to achieve safe and efficient packet manipulation inside the kernel. In this paper, we introduce TimeLSB, an eBPF-based covert channel that embeds information in the least significant bit of TCP timestamps. The main contributions of this paper are:

- TimeLSB a covert channel embedded in TCP timestamps, implemented through eBPF programs attached at the

This work is partially supported by The Scientific and Technological Research Council of Türkiye (TUBITAK) 1515 Frontier R&D Laboratories Support Program for Türk Telekom 6G R&D Lab under project number 5249902, by METU BAP Project GAP-312-2025-11520, and ARLEON Information, Communication and Cybersecurity Technologies Ltd., Türkiye.

978-3-903176-75-1 ©2025 IFIP

`tc` egress hook[1]. The implementation enables in-kernel manipulation of the timestamp option, achieving high performance with minimal processing overhead.
- We conduct an empirical evaluation of how timestamp rewrites affect normal TCP connections in a simulated environment. This includes analysis of throughput, latency, and overall channel capacity under varying levels of network impairment (loss, delay, reordering, and duplication).
- We perform a detection analysis of the covert channel using a timestamp-based statistical feature on different types of TCP traffic, examining whether it can be used to distinguish normal and covert flows.

The paper is organized as follows. Section II introduces the design and implementation details of the proposed covert channel, TimeLSB, including its eBPF-based sender, supporting maps, and receiver. Section III describes the experimental setup and presents the analysis of channel behavior under both simple and combined impairments, as well as the processing overhead introduced by eBPF. Section III-D discusses the detectability of the channel using the distinct timestamp ratio.

II. TimeLSB

The Transmission Control Protocol (TCP) is the dominant transport-layer protocol used on the Internet. It provides a reliable, connection-oriented communication service between applications, ensuring ordered delivery of bytes across potentially unreliable networks, and is used in the majority of Internet applications, including web browsing, email, and file transfer. Among its extensions, the TCP timestamp option (RFC 7323) adds two 4-byte fields: the Timestamp Value (TSval) and the Timestamp Echo Reply (TSecr). Timestamps are monotonically increasing counters, updated with millisecond granularity, and is widely deployed in modern stacks for round-trip time estimation and protection against sequence number wrap (PAWS) around. TCP timestamp option is optional. However, a recent study [12] reports that most of modern operating systems enable it by default. Windows clients are typically the exception. As our proposed methodology targets server-side stacks (commonly Linux/Unix), Windows defaults are not a blocker for our setting. Because timestamps are ubiquitous and cannot easily be disabled without harming TCP performance, they provide a suitable carrier for covert data. In our design, covert information is embedded into the least significant bit (LSB) of TSval, allowing hidden communication while not disrupting the existing connection.

A network covert channel builds on this principle, enabling two parties to exchange hidden information by embedding it within legitimate traffic flows. In such a scenario, a secret sender disguises messages inside these flows with various methods, while a secret receiver extracts them without disrupting the overt communication. The primary objective of this technique is to bypass monitoring systems such as firewalls or intrusion detection tools, which are designed to enforce security policies and block unauthorized communication. We assume the attacker (sender) has access to the secure network and can execute the program on the host from which information is exfiltrated. There are several ways a covert channel could be implemented using the TCP timestamp option. The common approach is to generate custom packets in user space with tools such as Scapy or raw sockets, directly setting timestamp values. Although simple to prototype, artificial packet payloads must be produced that mimic real TCP traffic. A stealth approach is to modify packets in transit through a man-in-the-middle although in-flight modification of TCP timestamps by intermediaries may terminate connections [8].

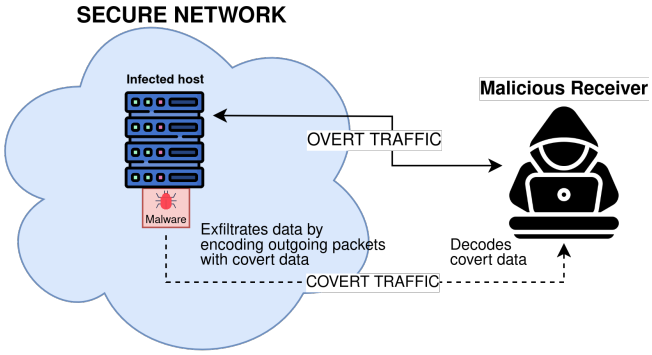

Fig. 1. Covert communication model for TimeLSB.

A. Overall Model

A network covert channel enables hidden communication by embedding secret information within an existing normal connection. As illustrated in Fig. 1, an infected host maintains a legitimate communication flow, referred to as overt traffic, while simultaneously encoding covert information into the same packets. To outside observers, the connection appears to function normally, but a malicious receiver is able to extract the hidden data, which forms the covert traffic. The purpose of this mechanism is to evade security systems such as firewalls or intrusion detection tools by concealing secret communication within otherwise legitimate network activity.

B. Kernel eBPF program

There are several parts of the implementation namely: an eBPF program, eBPF maps which provides data structures for interacting with kernel program through user space, and receiver. The program processes outgoing TCP packets at the `tc` egress hook, immediately before transmission through the network interface. For each packet, once the lower-layer headers are parsed, a CRC32 hash of the first 20 bytes of the TCP header is computed as shown in Fig.2. From this hash, the first byte is extracted and employed as the `bit_index` for addressing a message block (details of message partitioning are provided in Section II-D). As the byte value ranges from 0 to 255, it can index a message block of 32 bytes (256 bits). The selected bit, denoted as `plain_text_bit`, is XORed with

[1]https://github.com/yufusuf/tcp_ebpf_covertchannel

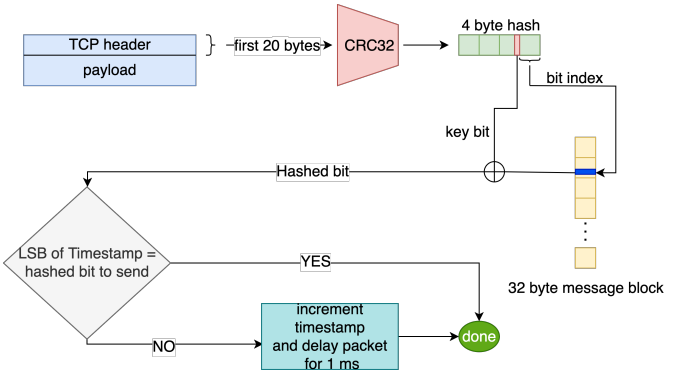

Fig. 2. Timestamp rewrite procedure in TimeLSB.

the bit obtained from the ninth bit of the hash value. This is the bit that is denoted as `key_bit`, and it is arbitrarily chosen from the hash. The resulting hashed bit is then compared with the least significant bit (LSB) of the TCP timestamp field. If equality holds, no modification is performed. Otherwise, the timestamp is incremented, the updated value is written back into the header, and the packet is delayed by 1 millisecond, since timestamps are updated with millisecond granularity.

C. Parsing TCP Timestamp Option

TCP timestamps reside in the TCP header at the 20th byte. Since option lengths can be variable options had to be parsed with an indefinite loop until option with the kind "end of options" is found. However to protect memory safety, eBPF verifier rejects such loops. Instead, on most TCP connections if timestamp option is enabled, aside from SYN and SYN-ACK packets, TCP timestamps reside on 23rd byte. The TSval is extracted directly from there. In this way we save some cpu cycles while losing some generalization.

D. Maps

The kernel program is controlled by a user-space program that updates these data structures. These maps are:

1) tx_count: Since the covert channel is a best effort channel, some degree of reliability is achieved through redundancy. Since we select which bit to send through a hash value, the selection is random. With the use of this map, how many times each bit is sent is tracked and it is ensured that each bit is sent at least `occupation_number` of times [11], which was set to 3 in our implementation.

2) message_map: The message to be sent is segmented into 32 byte blocks and each block's last 4 bytes is the CRC32 digest of the first 28 bytes of the actual message. On the receiver side we know a message block is completely received when calculated digest matches the blocks digest. Without unloading the kernel program the message can be updated and the program state is reset through this map.

E. Imposing Delays

We had to install a fair queue (fq) scheduler as root qdisc on the egress interface because on packets that required their

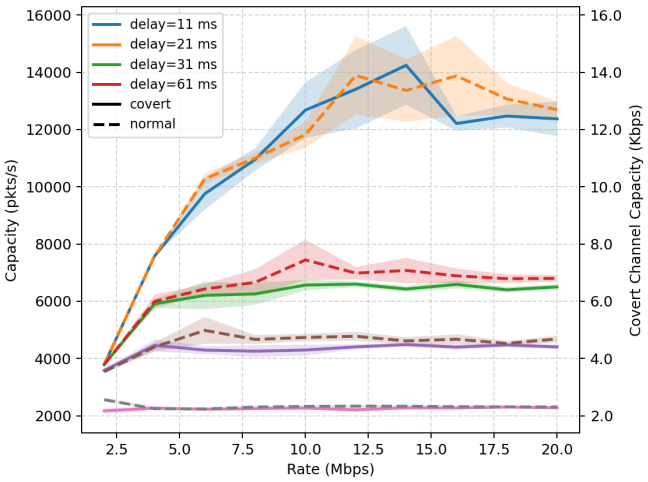

Fig. 3. Combined loss and rate impairment effect on covert and normal channels.

timestamps to be incremented has to be scheduled 1ms later. This is actually not needed for slower links where inter-arrival times of packets are sufficiently large enough. However, due to bursty nature of TCP traffic, where timestamp update frequency (1ms) is smaller than segment transmission rate i.e number of packets have the same timestamp going out from the interface. If some of these are packets got their timestamp lsb rewritten we would have monotonicity of timestamps not respected. In bursts, several timestamps have the same value, so encoding of earlier packets might increment timestamp value, which would cause their timestamp to be larger than subsequent packets. The mechanism used for delays was through a bpf helper function called `bpf_skb_set_tsamp()` which sets the `tstamp` field in `__sk_buff` structure allowing the delaying of 1ms through forcing the scheduling of packet to occur later. This field is not respected in other qdiscs so, fq is employed as the root qdisc.

F. Receiver

The receiver is implemented in C using `libpcap`. It passively sniffs packets originating from the source IP where the sender resides, without interfering with the ongoing TCP connection. For each packet, it mirrors the sender's operations by calculating the hash and determining the corresponding bit received. A message is deemed successfully received once the receiver verifies integrity by comparing the CRC32 of the first 28 bytes with the last 4 bytes of the reconstructed block. After this confirmation, the decoded message is printed.

III. Experimentation and Analysis

The impact of timestamp rewrites was analyzed under a range of channel conditions. The examined parameters included channel rate, packet loss percentage, packet delay and jitter, as well as the degree of packet reordering. To evaluate the isolated effect of each factor, parameters were varied individually while the others were held constant. In addition,

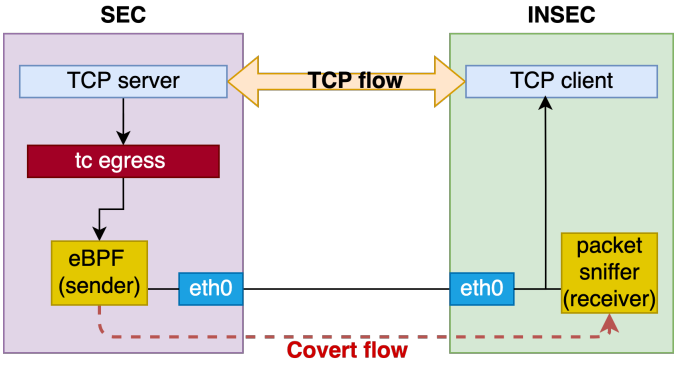

Fig. 4. Experimentation setup for network simulation.

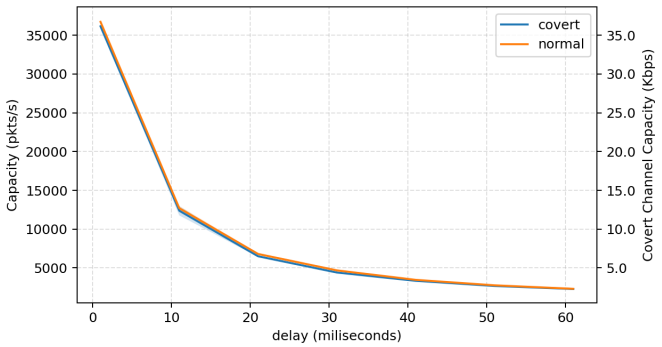

Fig. 5. The impact of delay on the capacity of the covert and overt channels.

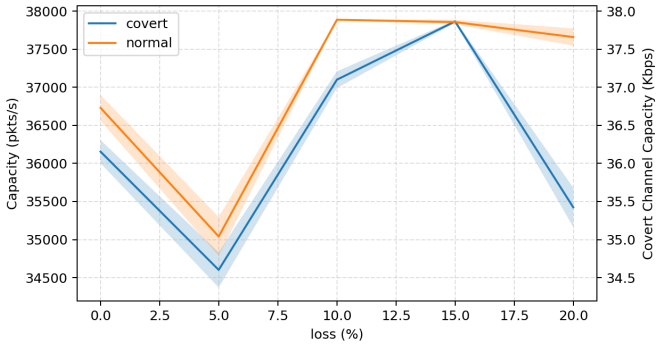

Fig. 6. The impact of loss on the capacity of the covert and overt channels.

experiments were conducted with combined impairments to approximate more realistic channel behavior. To ensure statistical reliability, each experiment was repeated 30 times which includes 95% confidence intervals, and in every run a total of 20,000 packets were captured.

A. Experimentation setup

Our experiments were conducted in a controlled Linux environment using network namespaces, which provide isolated network stacks to emulate distinct hosts and networks on a single machine. This setup allowed us to separate the covert sender and receiver into different namespaces connected via a virtual Ethernet (veth) pair. Specifically, we created two namespaces, named `sec` and `insec`, as shown in Fig. 4. The `sec` namespace emulates a high-security domain and hosts a dummy TCP server, while `insec` emulates a low-security domain and runs a dummy TCP client that generates traffic using `iperf`. Channel impairments and rate limiting were modeled using the Linux `tc` utility in combination with the `netem` [13] queuing discipline, which enables controlled injection of delay, loss, reordering, and duplication. The receiver implementation relies on `libpcap` 1.10. All experiments were executed on a host with an Intel i7-6700@3.40GHz CPU, running Linux 6.8.0 with `libbpf` 1.5.0. The complete source code for the experimental setup and covert channel implementation is available on GitHub[2].

B. Evaluation

In this section, we present the experimental evaluation of the proposed covert channel. The goal is to examine how the TimeLSB behaves under different network conditions and to compare its performance with that of normal TCP traffic. We first analyze the effect of individual impairments such as delay, loss, and rate limitations, before moving on to a more realistic scenario where multiple impairments are combined.

1) Simple impairments to the channel: The results, shown in Fig. 5, 6, 7, compare the capacity of normal and covert flows under a couple of channel impairments and rates. The TCP channel capacity is reported in packets per second on the left y-axis, while the covert channel capacity, measured in Kbps (one bit per segment), is shown on the right y-axis. Across all scenarios: delay, loss and rate; the overall behavior of covert flows closely follows that of normal flows. Delay produces the largest reduction in capacity, but the relative gap between covert and normal traffic remains small.

In addition to the simple impairment tests, the effect of different delay values across channel rates is shown in Fig. 3. This figure illustrates the impact of varying loss levels across different rates, where we observe similar trends apart from some fluctuations. At shorter delays (11 ms), at some points normal flows capacity drops below covert flows. The overlapping confidence intervals indicate that the covert modification introduces insignificant changes as before. At higher delays, similarly, the covert channel consistently shows a slight performance reduction, as seen in previous results.

2) Combined channel impairments: To model a more realistic TCP flow, channel impairment parameters are combined. Capacity decreases as impairments increase from none to medium to high as defined in Table I, reflecting the compounded effect of loss, delay, reordering, duplications, and reduced rate as can be seen in Fig. 8. Importantly, the comparison between covert and normal flows shows that their performance remain almost identical across all regimes. The gap between covert and normal remains small relative to

[2]https://github.com/yufusuf/tcp_ebpf_covertchannel

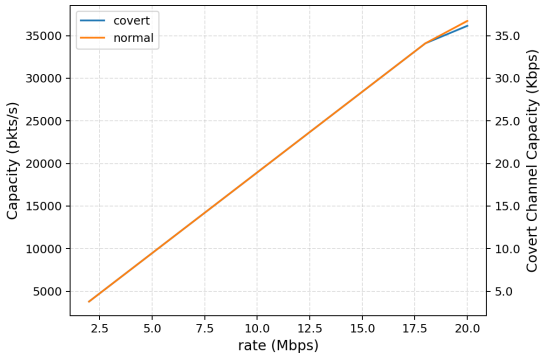

Fig. 7. The impact of rate limiting on the capacity of the covert and overt channels.

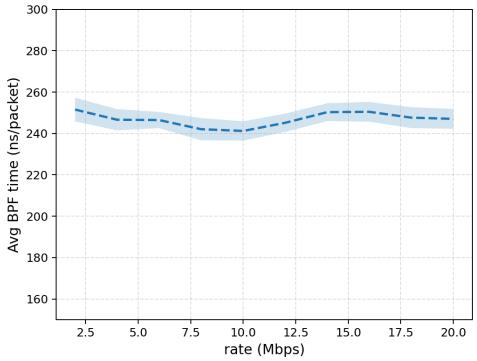

Fig. 9. Average eBPF processing overhead time per packet.

the overall reduction in capacity. This suggests that inserting covert bits does not significantly alter how the covert channel responds to multiple simultaneous impairments, supporting the conclusion that the covert mechanism maintains consistency with normal traffic behavior under realistic network conditions.

TABLE I
IMPAIRMENT REGIMES.

Regime	Loss (%)	Delay (ms)	Rate (Mbps)	Reorder (%)	Dup (%)
None	0.0	0	20	0.0	0.0
Mid	0.5	50	10	0.2	0.001
High	1.0	100	5	0.4	0.01

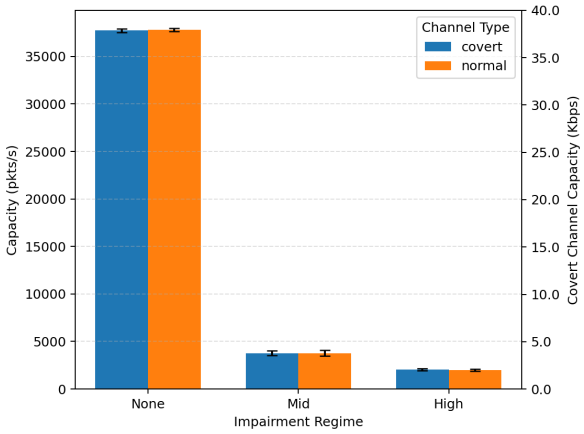

Fig. 8. The impact of combined impairment regimes on the capacity of covert and overt channels.

C. eBPF overhead

The measured eBPF processing overhead is approximately 240–255 ns per packet, which corresponds to roughly 4–5 million packets per second (Mpps). Fig. 9 shows average overhead across different channel rates. The small fluctuations can be due to system-level factors such as scheduling, cache effects, etc.

D. Detection of TimeLSB

Although modern intrusion detection systems mostly rely on machine learning [14], we resort to using a heuristic approach for detecting TimeLSB in this work and leave the machine learning approaches as a future work. We present a distinct timestamp ratio which could be utilized to detect the presence of these covert channels. Detection tests are conducted with the methodology described in [15]. Let the sequence of captured TCP timestamps be $T = \{t_1, t_2, \ldots, t_N\}$, where N is the total number of observed packets. We define the set of distinct timestamp values as $D = \{t_i \mid t_i \in T\}$, with cardinality $|D|$ representing the number of unique timestamp values. Let ΔT denote the range of the observed timestamps: $\Delta T = t_N - t_1$. The distinct ratio R is then computed as

$$R = \frac{|D|}{\Delta T},$$

which measures the fraction of unique timestamp values relative to the total increment in the timestamp field during the capture.

Fig. 10 shows the distinct-timestamp ratio R across rates under different delay settings. Although our absolute values do not match those reported in prior work [15] ($\approx$ 0.75 vs. $\approx$1 for normal and covert flows, respectively), we consistently observe that covert flows achieve higher ratios than normal flows. This separation holds across all rates and delays. We attribute this shift to differences in timestamping behavior in our environment (e.g. segmentation offloads, generated traffic). Importantly, the metric remains discriminative and even on lower bandwidth channels: the gap between covert and normal flows is stable, indicating that distinct-timestamp behavior continues to provide a reliable detection signal, and the threshold for signal could be determined with channel characteristics. Because certain packets timestamps in covert flows are deliberately advanced by an extra 1 ms, the number of unique timestamp values increase more in relation to total increment of timestamps, leading to a higher distinct-timestamp ratio compared to normal flows.

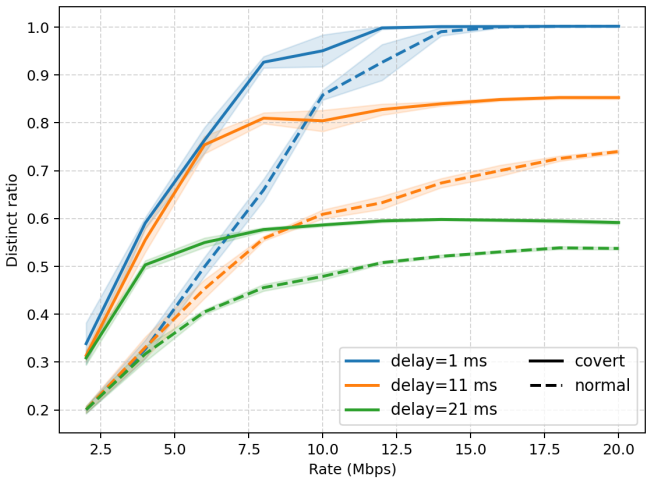

Fig. 10. Distinct-timestamp ratio (R) as a function of channel rate under different delay values.

IV. Conclusions

This paper implemented and analyzed a covert channel on TCP timestamps using an eBPF-based sender and a passive libpcap receiver. The sender encodes information in the least significant bit of TCP timestamp values, guided by a CRC32-based hash; kernel–userspace coordination is handled through eBPF maps.

Our evaluation shows that the proposed covert channel can operate under a wide range of network impairments with minimal impact on throughput and only nanosecond-scale eBPF overhead per packet. Across delay, loss, and combined impairment regimes, covert and normal flows exhibit nearly identical channel behavior, confirming that the embedding does not fundamentally alter TCP performance in our simulated environment. At the same time, analysis of the distinct-timestamp ratio demonstrates that covert flows leave a measurable difference that can be used to detect this covert channel.

Beyond the experimental perspective, these findings also carry implications for network security operations (SecOps). First, they demonstrate that commonly deployed TCP extensions such as timestamps can be misused for covert data exfiltration, bypassing conventional security middleboxes. Since the timestamp option is critical for round-trip time estimation and Protection Against Wrapped Sequence numbers (PAWS), operators cannot simply disable or strip this field without degrading legitimate performance. This makes detection and mitigation particularly challenging for security operations centers (SOCs), especially in environments where insider threats or compromised endpoints may execute eBPF programs directly in the kernel. Second, the detection methodology explored here offers a practical tool for operational defense. The distinct-timestamp ratio feature can be integrated into traffic monitoring pipelines or anomaly-based intrusion detection systems to flag suspicious flows with minimal processing overhead. Moreover, because the per-packet eBPF overhead of the channel is measured at only a few hundred nanoseconds, similar kernel- or SmartNIC-based monitoring mechanisms could also operate at line rate. This opens the possibility of deploying eBPF not only offensively for covert channels, but also defensively, either to monitor timestamp behavior in real time or to enforce active wardens that normalize anomalous values. In this way, the study highlights both an attack vector and potential operational strategies for detection and proactive defense in modern SecOps workflows. The channel suffers from limited reliability, particularly duplicate block reception at high bandwidths. Future work can explore error-control mechanisms to improve robustness under varying channel conditions.

References

[1] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, p. 613–615, Oct. 1973.

[2] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: design and detection," in *Proc. of the 11th ACM Conference on Computer and Communications Security, CCS '04*. New York, NY, USA: Association for Computing Machinery, 2004, p. 178–187.

[3] N. Hou, X. Xia, and Y. Zheng, "CloakLoRa: A covert channel over LoRa PHY," *IEEE/ACM Transactions on Networking*, vol. 31, no. 3, pp. 1159–1172, 2023.

[4] G. Teca and M. Natkaniec, "A novel covert channel for IEEE 802.11 networks utilizing MAC address randomization," *Applied Sciences*, vol. 13, no. 14, 2023.

[5] G. Costa, F. Pinelli, S. Soderi, and G. Tolomei, "Turning federated learning systems into covert channels," *IEEE Access*, vol. 10, pp. 130 642–130 656, 2022.

[6] K. Harris, W. Henry, and R. Dill, "A network-based IoT covert channel," in *Proc. of the 4th International Conference on Computer Communication and the Internet (ICCCI)*, 2022, pp. 91–99.

[7] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in internet traffic with active wardens," in *Information Hiding*, F. A. P. Petitcolas, Ed. Berlin, Heidelberg: Springer, 2003, pp. 18–35.

[8] S. R. A. O. Filho, C. A. S. Lelis, E. T. E. Soares, and C. A. C. Marcondes, "Exploiting temporal variability in TCP timestamps for covert channel design," *IEEE Access*, vol. 13, pp. 115 909–115 923, 2025.

[9] J. Rutkowska, "The implementation of passive covert channels in the linux kernel," in *Proc. of the Chaos Communication Congress*, Dec 2004.

[10] M. Repetto, L. Caviglione, and M. Zuppelli, "bccstego: A framework for investigating network covert channels," in *Proc. of the 16th International Conference on Availability, Reliability and Security, ARES '21*, 2021.

[11] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Proc. of the International Workshop on Privacy Enhancing Technologies*. Springer, 2002, pp. 194–208.

[12] S. Sundberg, A. Brunstrom, S. Ferlin-Reiter, T. Høiland-Jørgensen, and R. Chacón, "Measuring network latency from a wireless ISP: Variations within and across subnets," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2024, pp. 29–43.

[13] S. Hemminger *et al.*, "Network emulation with netem," in *Linux.conf.au*, vol. 5, 2005, p. 2005.

[14] M. Husák, D. Manoj, and P. Kumar, "Machine learning in intrusion detection: An operational perspective," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024, pp. 1–7.

[15] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Information Hiding*, M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, Eds. Berlin, Heidelberg: Springer, 2005, pp. 247–261.