T-Pack: Timed Network Security for Real Time Systems

Swastik Mittal, Frank Mueller
North Carolina State University, smittal6@ncsu.edu, mueller@cs.ncsu.edu

2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC) | 978-1-6654-0414-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISORC52013.2021.00014

***Abstract*—Network communication between real-time control systems raises system vulnerability to malware attacks over the network. Such attacks not only result in alteration of system behavior but also incur timing dilation due to executing injected code or, in case of network attacks, to dropped, added, rerouted, or modified packets.**

This work proposes to detect intrusion based on time dilation induced by time delays within the network potentially resulting in system malfunctioning due to missed deadlines. A new method of timed packet protection, T-Pack, analyzes end-to-end transmission times of packets and detects a compromised system or network based on deviation of observed time from the expected time on end nodes, well in advance of a task's deadline. First, the Linux network stack is extended with timing information maintained within the kernel and further embedded within packets for TCP and UDP communication. Second, real-time application scenarios are analyzed in terms of their susceptibility to malware attacks. Results are evaluated on a distributed system of embedded platforms running a Preempt RT Linux kernel to demonstrate its real-time capabilities.

I. Introduction

Computer security is a critical requirement for any networked application nowadays. Main components of computer security include: (1) System level security, securing end system's code at different levels and layers from attacks leveraging memory (e.g., buffer overflow attacks including stack smashing [28], heap overflows [10]) or non-memory related attacks (e.g., value range overflows [17], shell shock [24], port smashing); and (2) network security, securing systems on the network from being attacked by malicious users. For the latter, a wide range of attacks are common, including relay, replay, phishing, spoofing, man-in-the-middle, denial of service, eavesdropping etc. [7], [14], [15], [33]. One of the common attacks in real-time systems is the delay attack. The objective of this attack is to stall execution/packets of a time-sensitive event (e.g., a code section within the system or some client/server request) causing excessive delays and resulting in performance degradation [23]. Real-time systems are particularly susceptible to such attacks as system behavior is compromised when deadlines are missed, i.e., time dilation not only results in performance penalties or reduced network throughput but may cause a control system to malfunction, which can result in damage to the controlled environment or even loss of life. Past work shows how delay attacks have affected cyber-physical systems (CPS) [23] and network control systems [31] subject to real-time constraints.

Significant work has been invested in analyzing and mitigating the impact of these attacks [23], [31]. Real-time systems offer a unique opportunity for intrusion detection besides traditional, general-purpose cyber-security techniques: Their inherent knowledge of worst-case execution times (WCET) [35], an upper bound on a task's execution budget required for real-time scheduling, opens up opportunities for additional monitoring and protection. The same techniques for establishing timing bounds on the execution of the real-time task may be applied to bound execution of any code section within the application [16], [36]. Past works have used this model for security in real-time systems for detecting memory attacks [37], securing clock synchronizations over the network [27] and protecting smart grid systems [29]. Timed analysis is not just restricted to security, it can also be used to design attack models, e.g., in the context of hardware security tokens such as Smartcards [13], [19], [32], exported secret decryption keys [19] and remote timing attacks [5]. The novelty of our work is to utilize time-based security to detect delay attacks on a network subject to communication with real-time constraints. This complements methods monitoring execution times for intrusion detection by providing a similar mechanism for packet transmissions.

In safety-critical distributed real-time systems, missed deadlines due to slower or missing packets could result in significant environmental damage or even in loss of life. System restarts often cannot be instantaneous due to an unstable physical system state. This research focuses on detecting intrusions due to such attacks at the packet level, i.e., before malware in one subsystem can enter another subsystem or even result in a deadline miss within the yet unharmed subsystems. The earlier intrusion is detected, the easier it is to resort to a safe operational mode with reduced or even without communication to another subsystem that has been compromised, e.g., using the Simplex design [3], [11], thereby avoiding any significant damage. Our work focuses on early intrusion detection on end nodes (as opposed to routers/switches as that would incur additional time to notify nodes), and while it relies on established methods to transition to a safe state (e.g., Simplex), the safe methods are beyond the scope of this paper.

Our work contributes:

- T-Pack is developed, a novel method for packet protection combining simplicity in design and implementation with integrity, negligible performance cost and no hardware modifications.
- T-Pack is compatible with other security protocols and utilizes IPSEC to establish data integrity alongside early detection of delay attacks.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

• Malware intrusion is detected by monitoring end-to-end packet deadlines at the time of packet reception instead of conventional detection at a task's deadline.
• Experiments with real-time applications under attack scenarios assess potential and limitations.
• Results indicate that T-Pack has low overhead per packet round-trip time ($\approx$ 0.09 milliseconds) and detected 95%-100% of the delays during ping flooding and distributed denial of service (DDOS) attacks in a number of experimental systems.

II. Attack Model

This work assumes a distributed environment with end nodes connected by a network with end-to-end real-time guarantees of message transmission, e.g., via packet prioritization. This can be accomplished by (expensive and proprietary) hardware solutions like TTEthernet [20], protocol extensions for time-based traffic shaping (e.g., 802.1Qbv [34], if supported), or via enhancements on top of (less expensive) software-defined network (SDN) equipment [30].

Each subsystem (node) within this CPS architecture is assumed to provide its own execution environment (processor). Inter-node communication is prioritized for real-time traffic in a statically constructed schedule. By this design, a sending subsystem puts a message on the wire via the network stack such that it is received prior to an end-to-end message deadline at the receiving subsystem, but the exact time of the send/receive activity may vary within deterministic bounds as they are determined by execution times of prior tasks.

An attacker may compromise a given subsystem and subsequently monitor network traffic and inject packets arbitrarily to either execute replay attacks (injecting duplicate packets), use IP spoofing (injecting packets with an incorrectly rewritten source address expecting lost reply packets from the destination) or flooding (SYN or ICMP packet flooding). Other subsystems remain unaffected in computation and their ability to send/receive packets as packet communication is assumed to use public key encryption, i.e., receiving nodes can detect content modifications including packet headers. (Even a replay attack with modified source can be detected by encrypting all original headers in the message such that a source mismatch between received header and decrypted header in the message can be detected). Other subsystems may, however, be affected by altered network behavior due to packets originating from the compromised node (additional or duplicated packets at any priority, modified packets, dropped packets with respect to the original static schedule). Most significantly, the compromised system neither has the ability to alter packets sent by uncompromised systems, nor may it change any router/switch functionality. A delayed packet reception results in an intrusion notification, just as an omitted packet, as a timeout will be raised at expected arrival time if the packet has not arrived yet. The benefit of our method is an early notification upon expected message arrival rather than a late timeout upon deadline violation, which leaves more time for exception handling / transitioning to a safe mode.

Fig. 1 illustrates T-Pack capabilities for both (a) TCP and (b) UDP, for a packet sent at t_S and received at t_R under real-time constraints with a task's absolute deadline of t_O (in line ①), which will trigger a timeout and exception if the packet has not been received by then. This deadline is present with or without T-Pack. The figure further depicts the expected arrival of the packet in range Δt_R. ②,③ & ④ illustrate the different distributions of the real-time events at times t_S, t_R & t_E for the packet sent, packet received and worst case end-to-end time (with time duration as Δ), respectively. For TCP, t_E is also the T-Pack timeout relative to the time when the packet is sent allowing earlier exception irrespective of packet arrival at t_R. Thunderbolt and cross (X) show triggered and canceled timers, respectively, plus exception (if triggered).

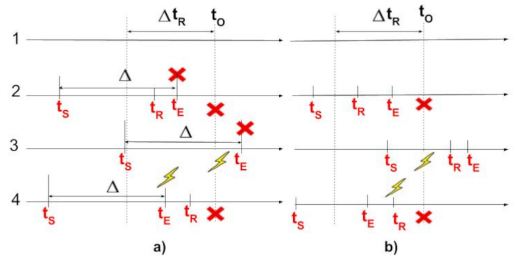


Fig. 1: T-Pack and global timeout scenarios for tasks with real-time deadlines under (a) TCP (b) UDP

② depicts a T-Pack scenario accepting an incoming packet at t_R followed by validation of the embedded timestamp. In Fig.1(a) under TCP, the packet is received before timeout t_E, i.e., the timeout at t_E is canceled. In Fig.1(b) under UDP, the duration for transmission, $t_R - t_S$, matches the expected transmission costs so that (i) *no* T-Pack exception needs to be raised as the duration is less than t_E; and (ii) timeout t_O can be canceled since the packet was received (both indicated by an X).

③ depicts a lost packet or a long delay before the packet is sent at t_S — both triggering a timeout at t_O resulting in an exception. As the packet is received at t_R, it will neither raise another exception at t_E (TCP Fig.1(a)), nor will it indicate a delay at t_R (UDP Fig.1(b)) as a late packet is simply ignored in hard real-time systems. In (m, k)-firm or soft real-time environments, where $m - k$ deadlines may be missed, the packet would not be ignored so that t_E can raise an exception if the transmission took too long, see next case.

④ depicts a packet sent early (t_S) as the sending real-time task takes less time than the budgeted WCET. A Timeout is triggered by T-Pack at t_E (TCP Fig.1(a)) as the packet has yet to be received; or, upon packet reception, the receiver notices that the transmission exceeded the expected duration (UDP Fig.1(b)) based on the embedded timestamp (i.e., $t_R - t_S > t_E$ is too long), indicating a delay at t_R, which cancels timeout t_O due to the early exception. This indicates a possible intrusion, which subsequentl triggers mitigation techniques. This illustrates the benefit of T-Pack over the global timeout as the prior exception due to timestamp validation leaves more time ($t_O - t_E$ under TCP and $t_O - t_R$ under UDP) for mitigation, i.e., to transition into a safe mode. Most of

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

all, this scenario would *not* result in any exception without T-Pack, i.e., (a) the triggered timeout at t_E or (b) reception at time t_R would cancel the timer at t_O even though the packet was delayed significantly.

III. Design

T-Pack is a novel methodology to verify end-to-end timing of each packet on the network of a real-time system during message transfer between subsystems. Fig 2 depicts a high-level timing model for message transfer between two subsystems for unidirectional UDP (left) and bidirectional TCP transfers (right) using the notation established by Table I. A message from sender S to receiver R is analyzed at packet level, considering packet P being sent at time t_S from S to R, where it is received at time t_R. The observed end-to-end time, T_{obs}, is compared with the expected time, T_{exp}, to detect malware intrusion in the network. The work assumes loosely synchronized clocks with a constant time difference, Δt_{rs}, between any two subsystems, which may be dynamically updated due to clock drift, as in our later implementation.

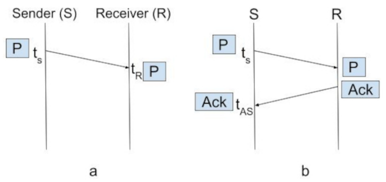


Fig. 2: a. One-Way (UDP)   b. Two-Way (TCP)

TABLE I. ABBREVIATIONS

t_S	Time recorded by T-Pack at which packet is sent from S
t_R	Time recorded by T-Pack at which packet is received at R
ACK	Acknowledgment packet from R to S in a 2-way message
t_{AS}	Time at which ACK is received at S (At Network Layer)
RTT	Round-trip time (TCP)
ETT	End-to-end time (UDP)
Δt_{rs}	Constant clock difference between S and R
T_{obs}	Observed end-to-end or round-trip time
T_{exp}	Expected end-to-end or round-trip time
T_d	Mean internal delay on the uncompromised network
ΔT_d	Deviation of internal delay from the mean internal delay
T_c	Added delay due to a compromised network
ΔT_c	Deviation of added delay from the mean added delay
T_{WCET}	Expected worst-case end-to-end time of packet

One Way Message Transfer: The UDP transport protocol is a suitable protocol assuming point-to-point full-duplex switch connectivity between network endpoints. Under UDP, a message is transferred from S to R (Fig. 2a) without any acknowledgment from R. In this scenario, the observed end-to-end time (ETT) is the time it takes for the packet to reach R having been sent by S.

$$ETT = t_R - t_S + \Delta t_{rs} \tag{1}$$

So, the expected end-to-end time is:

$$T_{exp} = ETT \tag{2}$$

Two-Way Message Transfer: TCP is a two-way message transfer transport protocol based on a handshaking protocol that acknowledges packets sent from the sender, S, to achieve reliable communication over the network (Figure 2b). In this scenario, a transfer is said to be complete once the acknowledgment (ACK packet) is received at S, again under point-to-point full-duplex switch connectivity. Here, we assume a constant clock difference between sender and receiver for the duration of the packet communication. Clock drift requires this difference to be updated from time to time, which is typical for distributed systems and beyond the scope of the paper [25].

Round-trip-time in TCP can be monitored without embedding time information within the packet. Instead, it suffices to measure the round-trip time of the packet (from the send to receipt of the acknowledgment at the sender).

$$RTT = t_{AS} - t_S \tag{3}$$

So, the expected end-to-end time is:

$$T_{exp} = RTT \tag{4}$$

TCP optimizes network traffic by consolidating multiple small bytes packet into one (reducing header overhead) at the sender [1] and sending cumulative acknowledgments at the receiver (reducing multiple ACK packets). This creates non-deterministic execution behavior in real-time system due to delays for sending and receiving. [6] shows how this affects the performance of a client-server application using the same communication pattern as distributed real-time systems. With controlled flow of packets in distributed real-time systems, additional data due to headers/trailers hardly reduce network bandwidth; instead, timely delivery is more important. We ensure timely delivery via socket options “TCP NODELAY“ (sender) and “TCP QUICKACK“ (receiver), which prevents packet consolidation.

The impact is analyzed for our experimental model (Paparazzi UAV, see Sec. V) by monitoring the average bandwidth and number of packets flowing in and out of the interface for a period of time with and without these socket options. Without socket options, we observe an average rate of data observed at the receiving (rx) end of 353.50 Kbps and at the transmitting (tx) end of 779.67 Kbps over 60 seconds. With socket options, this decreases only slightly to 347.98 Kbps and 753.80 Kbps at receiver and sender, respectively. In and out flows of packet also decreased slightly, i.e. from 486 to 473 packets/sec (receiver) and 742 to 713 packets/sec (sender). More significantly, a packet sent with socket options was instantly sent, whereas it was non-deterministically buffered at times without options. Clearly, the latter is not acceptable for real-time system, whereas a small decrease in bandwidth is.

Relation to the Attack Model: Using our T-Pack model, we aim to detect delay attacks within uncompromised subsystems. Such an attack may originate from a compromised subsystem that maliciously induces time overheads by injecting packets arbitrarily. This may cause delays at the switch due to ingress queue processing, even if packets are prioritized, in part because the compromised subsystem can prioritize packets as well. As a result, any (non-malicious) packet that is forwarded through such a switch may be delayed before it can reach the other uncompromised subsystem. For a message transfer (UDP & TCP), we have:

Using T_{exp} from Eq. 4,

$$T_{obs} = T_{exp} + T_c \pm \Delta T_c. \tag{5}$$

The objective of T-Pack is not to prevent intrusion but

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

rather to **detect** it within uncompromised subsystems due to incorrect timing on network behavior. This can prevent these other subsystems from becoming compromised as well — by a timely transitioning into a safe mode, e.g., via Simplex [3], [11] or other mode transitions depending on the application scenario, which is beyond the scope of the paper. Furthermore, intelligent switches could assist in blocking high priority packets that are non-compliant with a statically established end-to-end real-time message schedule under our attack model, but we do not expect such switches to actively notify end nodes of a compromised subsystem. This would violate the existing real-time schedule and induce sporadic messages, which may dilate latencies to where deadlines could be missed. Instead, T-Pack provides a means for uncompromised subsystems to autonomously detect intrusions by monitoring their own communication with other nodes.

Vulnerability Of T-Pack: Encryption prevents third party packet modifications by attackers as the private key of the receiver is unknown, even with access to the wire. This includes timestamp values of T-Pack within packets.

The WCET bound of a packet is determined by the end-to-end transfer in our model as follows. T_{exp} in T-Pack includes delays in an uncompromised system (expected delays on the network or internal delays). Let this delay be of magnitude $T_d \pm \Delta T_d$ (Table I). Hence, the WCET bound includes a maximum internal delay of $T_d + \Delta T_d$, which signifies the maximum positive deviation of the WCET.

$$T_{WCET} = T_{exp} \tag{6}$$

For a compromised network, we obtain T_{obs} from Eq. 5. For some values of $T_c \pm \Delta T_c$, where the attacker delays the packet transfer by a small value, we may find that $T_{obs} \leq T_{WCET}$, i.e., short delays may remain undetected. In other words, our model is probabilistic and may result in missed intrusion detection (false negatives), where our model does not identify an attacker in the network. This illustrates two points: (1) Our model complements existing cyber security measures and (2) the objective of T-Pack is to make the attack window that remains undetected as small (short) as possible, but only if attack traffic affects control functionality (deadlines). As long as packets are received in time, subsystems remain intact, i.e., additional background traffic may be tolerated.

Time Information: To support UDP, T-Pack embeds timing information within each packet to verify that end-to-end times of a packet are within given WCET bounds. A custom header with timing information is added just above the packet payload within the kernel comprising the lower level of the network stack (instead of higher networking layers). This establishes tighter and more deterministic bounds on the elapsed network delay.

To support TCP, time information of a sent packet is analyzed within the kernel with round-trip-time calculated using the corresponding acknowledgment received without the need to embed additional information within the packet. A lookup table is maintained on each subsystem to store sent time information of the packet to other subsystems.

Inclusion of TCP_NODELAY and TCP_QUICKACK ensures that only one outstanding packet from the same subsystem exists before an ACK is sent for the respective packet at any time.

Due to the static size of the table, the execution time of a table lookup is constant, which makes bounds under T-Pack highly predictable.

IV. Implementation

Linux: T-Pack is implemented in a PREEMPT-RT patched Linux kernel that provides real time capabilities to the operating system. This provides the flexibility of utilizing Linux network APIs such as socket buffers and netfilter to implement T-Pack.

Netfilter: Netfilter is a framework provided by the Linux kernel to implement customized handlers on events in the network layer (for pre-routing, post-routing, etc.). T-Pack utilizes this framework to implement callback functions to time packets (Fig. 3).

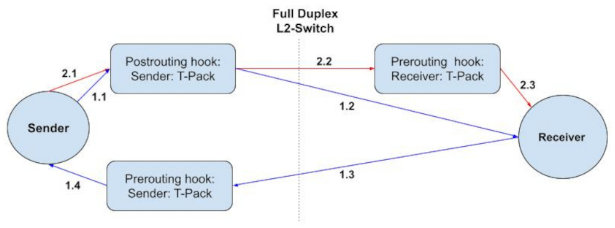


Fig. 3: Framework 1: Netfilter Hooks to Record Time Information for TCP (blue) and UDP (red) using T-Pack

Socket Buffers: Socket buffers are data structures provided by Linux as a common reference to packets in all layers of the network stack within the kernel. The T-Pack prototype utilizes the data types and the helper functions in the socket buffer API to record time (TCP) or manipulate packet memory in order to create additional space for the custom header (UDP) as depicted in Fig. 4.

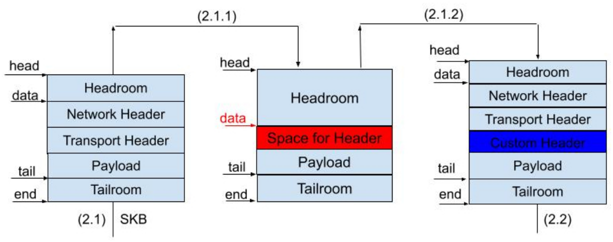


Fig. 4: Framework 2: Custom Header Insertion/Removal using Socket Buffers for T-Pack in UDP

Implementation Framework: 1) T-Pack for TCP (Fig. 3 with network path in blue):

1.1) At the sender, the netfilter post routing hook is utilized to call a handler, where the socket buffer references the packet as an argument.

1.1.1) The current time at the sender is recorded and stored in a lookup table for the corresponding subsystem (IP address, port) as a key. The expected acknowledgment sequence number is also recorded for the same key, which is the sequence number in the packet plus the data payload.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

1.2) According to TCP (with socket options as indicated in Section III), a packet is received at the receiver (1.2) and a corresponding ACK is sent (1.3) immediately.

1.3) At the sender, the netfilter pre-routing hook is again utilized to call a handler, where the socket buffer references the ACK packet as an argument.

1.3.1) The received ACK is matched with the destination IP address and port. If the expected sequence number matches the number in the ACK packet, the stored time is subtracted from the current time to determine the round-trip-time of the packet, which is then compared with the worst case round-trip-time to detect potential intrusions.

2) T-Pack for UDP (Fig. 3 with network path in red):

2.1) At the sender, the netfilter post routing hook is utilized to call a handler, where the socket buffer references the packet as an argument.

2.1.1) The $skb\_pull(sizeof(network\_header))$ and $skb\_push(sizeof(custom\_header))$ function is used to create additional space just before the packet payload to attach the custom header (see Fig 4).

2.1.2) $memcpy(ptr, custom\_trailer)$ encapsulates the custom header by copying it to the packet. The custom header includes fields for *ktime_t timestamp* and *long sendtime*, where timestamp contains the current time (at header creation).

2.2) At the receiver, the netfilter pre-routing hook is utilized to call a handler, where the socket buffer references the packet as an argument.

2.2.1) Upon UDP reception, the difference of the current time and the timestamp embedded within the packet represents the end-to-end processing time of lower layer UDP activities, subject to validation against an expected upper bound for the exchange. If validation fails, an intrusion is signaled.

V. Experimental Setup And Applications

Experiment 1: Client Server Model: A client sends periodic messages (interval of 10ms) to the server according to the network activity of a time triggered real-time system. TCP messages that fit in a single packet are sent with an explicit reply packet from the server. Measuring this ping-pong message transfer derives the round-trip time (RTT) at the client. We measure RTT at both application and network layers to assess the benefits of implementing T-PACK at the network layer. We also measure the RTT at the application layer with and without T-Pack to analyze T-Pack's performance overhead.

Experiment 2: Paparazzi UAV Model: The Paparazzi UAV [4], [38] models a real-time control system utilizing shared memory, which we transformed into a peer-to-peer network of 3 subsystems: an auto pilot (AP), a fly by wire (FBW) control and a ground station communicator (GSC) to relay information from subsystems to ground or vice-versa. We prototyped a model constrained to only Paparazzi's periodic messages scheduled between the above three subsystems. Each subsystem is connected via the (1) UDP and (2) TCP protocols with a persistent connection. The subsystems communicate with each other periodically transferring necessary information for flying by wire autonomously with T-Pack integration. The RTT is measured between AP and GSC communicator to monitor T-Pack. A delay attack as a Distributed Denial of Service (ICMP packet flooding / ping-of-death) [12] is induced at the GSC using other nodes in the network as attackers. This resembles code injection by the attacker on the compromised subsystem to inject packets arbitrarily. We implemented this Paparazzi model on a network of Raspberry Pi systems with a Preempt RT patched Linux kernel to provide real-time capabilities.

Experiment 3: Waters Workshop Challenge 2018, a Drone-like Multi-System: A drone-like multi-system [22] within a peer-to-peer network of seven subsystems is implemented consisting of a Mission management system (MMS), Electrical Propulsion System (EPS), Hydraulic Braking System(HBS), Sensors (communicating with other sensors in the Waters model), Ground Station and Maintenance System, connected via a hub and spoke topology within the same subnet. We again model functions and communication patterns in each subsystem, including periodic calls to the functions. Each subsystem is connected via TCP (persistently), sending messages to other subsystems in parallel under random network congestion with T-Pack support. The RTT is measured between EPS and MMS to monitor T-Pack functionality with and without a delay attack on Raspberry Pis with Preempt RT-patched Linux.

VI. Results

Experiment 1: Results for different server client configurations (x-axis) are depicted in Fig. 5a with RTT (y-axis) of messages shown as box plots indicating maximum, top quartile, median, bottom quartile and minimum times as well as outliers (dots). Min-max values or variability outside the upper and lower quartiles are denoted by whiskers with a range 3.5 times that of the inter-quartile range (constant for all other measurements within the paper). The outliers in this graph are only 0.5% of the total data values. We report all values from the experiments, even the first iteration of execution, which may be subject to additional cache misses resulting in an outlier.

We observe that the time measured for T-Pack (box plot 1 in Fig. 5a)for the reduced network stack is much lower than the one measured by the application, both with and without T-Pack (plots 2+3). By monitoring time within the kernel, T-Pack eliminates the cost of upper kernel layers both for sender and receiver resulting in an earlier intrusion detection at the network layer than at the application layer (baseline). Measuring RTT at the application layer would also require the applications to have explicit replies to every sent request, i.e., up to twice the number of messages are required in contrast to an implementation within the kernel. This could lead to unnecessary saturation of the write buffer of the receiver, which might be due to a send causing corresponding receivers to delay their communication. This increase in traffic would also result in higher RTTs for all the packets,in turn resulting in delayed intrusion detection because of larger timeouts. False negatives for timeouts at the network layer were measured by introducing a DDOS attack in Experiment 1 using a

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

single attacker with 10 attack threads, each sending 100 bytes of ICMP packets at an interval of 0.001 seconds. Over a range of 300 packets sent, $\sim$170 detected in intrusion due to timeouts at the kernel-level network layer of which 130 were false negatives (F1 score of 0.723) compared to $\sim$289 false negatives at the user-level application layer (F1 score of 0.071). This illustrates the benefits of our approach with T-pack within the kernel at the network layer. The attack introduced above is of intensity between A5 and A6 5b.

The results also reveal the overhead without the T-Pack module. Fig. 5a indicates that T-Pack incurs a modest performance cost as the overall mean RTT of the client request-reply increases by a marginal amount of approximately 0.09 msecs.

We analyze consistency, flexibility and integrity of T-Pack over secure communication between the client and the server by implementing the client-server model on top of a communication channel protected by the IP-Security (IPSec) protocol at the transport layer with RSA key authentication and encryption.

Any data packet is encrypted by IPSec after T-Pack in UDP adds its custom header. Should an attacker (who does not hold the private key) modify the packet, this would be detected as data then becomes corrupted after decryption, including T-Pack's timestamps.

IPSec provides security against session hijacking, man-in-the-middle attacks etc. This cannot prevent attackers imposing delays by transmitting unwanted packets, but T-Pack will detect such delays. Of course, data integrity comes at the price of increased RTT (TCP) and ETT (UDP), as assessed in the next experiment.

Experiment 2a: UAV Paparazzi Subsystems under TCP: We monitor the RTT under TCP from AP to GSC (i.e., send and acknowledgment) under a delay attack. Each attacker features a multi-threaded program to send large ICMP ping packets in quick intervals to the GSC. This causes a buffer overflow at the receiving interface, which is handled but results in performance degradation. The sensitivity to attack intensity is investigated dependent on a tuple, $P(n, t, b, i)$, by varying the number of attackers, n, the number of threads within an attacker, t, the ping packet size, b (in bytes), and the time interval, i (in seconds), between packets for the tuples indicated in Figures 5b and 5c. By modifying parameters, the intensity gradually increases to a level where the DDOS attack results in a noticeable impact due to buffer overflows on the network devices, which eventually causes deadline misses due to excessive RTT (TCP)/ETT (UDP).

Fig. 5b depicts RTT as box plots again (y-axis) over different intensities of DDOS attack (x-axis). The outliers in this graph are only 0.5% of the total data values (eliminating only 1% of the extreme outliers including the first iterations) As attack intensity increases, the RTT increases slightly. Compared to attack A1, A2 hardly effects the average RTT,

A6 increases the RTT on average by ≈ 0.65 msecs, and A7 by ≈ 2.6 msecs. For A7, all the measured RTTs exceed those of A1 without any false negative (no overlapping values). In other words, T-Pack accurately captures the "attack vector", since the WCET bound of RTT without intrusion is lower than the minimum RTT on a compromised network. Recall that to take over an entire kernel, millions of instructions are typically required. We can limit an attack to 15k instructions here assuming, e.g., a CPU clock of 1GHz at an instruction per cycle rate of one, which restricts undetected intrusions under T-Pack to a ≈ 0.2 msec window. Thus, a chain of 5 attack instances with 200K instructions each would be required (adding up to 1M instructions) to take over the system without being detected.

We also observe a sudden increase in RTT in Fig. 5b when as attack intensity increases. DDOS prevents effective resource utilization by consuming most of the resources (network and receiver buffer here) within the attacker [26]. A low intensity of such an attack does not affect the performance of a low traffic network, which is typical for a number of distributed real-time systems. However, intensifying the attack can cause sudden spikes that instantly degrade the network latency leading to packets arriving after a deadline, if at all. T-Pack detects this, which allows a system to transition to a safe mode while continuing to operate. Transitioning back online requires the attack source to be removed in a DDOS attack, both for real-time or commodity computing environments, i.e., counter measurements addressing the root cause remain unchanged and are beyond the scope of this paper.

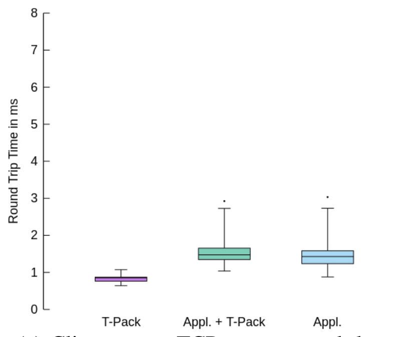
We further analyze the results of frequency distributions in Figures 6, 7 and 8, which depict the number times (y-axis) a certain RTT (x-axis) was measured in experiments. Figure 6 indicates the distribution for without intrusion (A1 in blue) and with high-intensity intrusion (A7 in red). An empty intersection between the distributions indicates that both the cases can be discretely distinguished, i.e., the attach vector is accurately identified by T-Pack.

Fig. 7 depicts distribution results without intrusion (A1) and a relatively intense attack (A6). We observe a slight overlap between the blue (no attack) and red (A6) curves ranging from 0.9-0.95 msecs, a data range covering less than 1% of the samples, i.e., more than 99% of the attacks are detected (F1 score of 0.993).

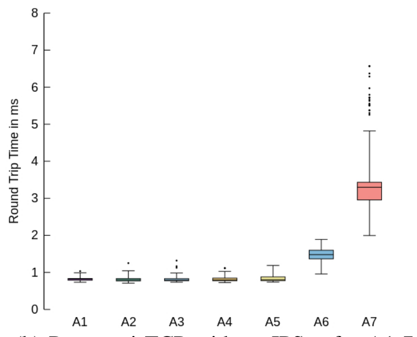
Fig. 8 depicts distributions without intrusion (A1) and a mild attack (A2). Results indicate a significant overlap between the blue (no attack) and red (A2) curves ranging from $\approx$ 0.65-0.85 msecs, a range with over 99% of the samples (F1 score of 0.014). This illustrates the limitations of T-Pack. Any attack with similar delays does not cause deadline misses. This illustrates why T-Pack *complements, but does not substitute* other security methods. In fact, T-Pack functions as expected: When deadlines are met, system functionality remains intact as sufficient network bandwidth remains available, and no intrusion is detected, but when attack intensity increases, intrusion is flagged requiring, e.g., Simplex mode changes.

Results in Fig. 5c reinforce our qualitative analysis of consistency, flexibility and integrity of T-Pack with IPSec from Experiment 1. Just as in Fig. 5b, we observe a similar trend in Fig. 5c in the distribution of RTT values when the Paparazzi model is subjected to different intensities of DDOS attacks.

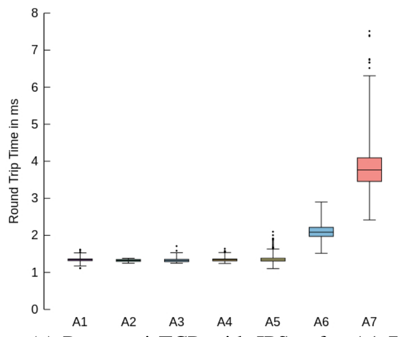
Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.



(a) Client-server TCP over network layers



(b) Paparazzi TCP without IPSec for A1-7



(c) Paparazzi TCP with IPSec for A1-7

Fig. 5: RTT in ms with/without T-Pack; Configurations for 5b and 5c: No attack A1:P(0,0,0,0), attacks A2:P(1,10,500,0.5), A3:P(1,10,500,0.1), A4:P(2,10,500,0.1), A5:P(2,30,500,0.05):, A6:P(2,10,500,0), A7:P(2,30,1000,0.001)

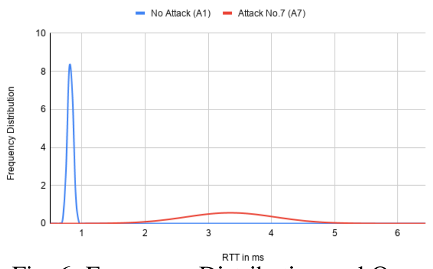


Fig. 6: Frequency Distribution and Overlapping Region of Attack 1 vs. 7

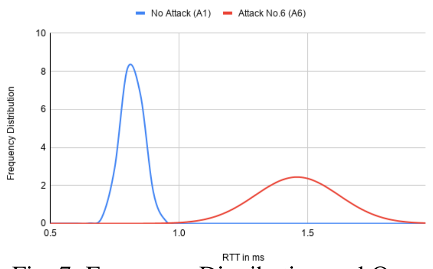


Fig. 7: Frequency Distribution and Overlapping Region of Attack 1 vs. 6

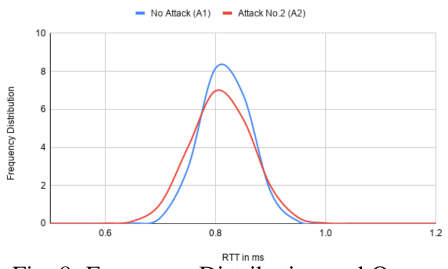


Fig. 8: Frequency Distribution and Overlapping Region of Attack 1 vs. 2

Absolute round-trip times are elevated by about 0.65ms. We observe that for higher intensity attacks (A7 in red), the RTT increases such that the entire box plot+whiskers range lies above the maximum RTT without attacks (A1 in purple). Also, ≈99% of the RTT values under A6 (blue) lie above the maximum RTT of A1.

Experiment 2b: UAV Paparazzi Subsystems under UDP: We next assess T-Pack under UDP for the UAV system by assessing packet ETT from one subsystem (AP) to another (GSC). Recall that unidirectional message transfer (UDP) requires a clock offset value between the two subsystems to calculate the ETT, which we implemented in analogy to the Network Time Protocol (NTP) within T-Pack. At system initialization, T-Pack measured RTT values of a UDP packet from AP to GSC. AP embeds timestamp information in the packet via T-Pack and sends it to GSC. GSC replies with another timestamped UDP packet plus its calculated time difference from the prior reception from AP. AP uses GSC's reply to calculate its offset to GSC using GSC's timestamp. In addition, the RTT is calculated just as for TCP using the time difference at AP between the reply from GSC and original send to GSC using AP's local clock. We experimentally verify that the RTT value closely approximates the one-way send overhead considering the clock offset between AP and GSC plus the one-way overhead plus offset for the reply. Subsequently, the clock offset is used to calculate ETT according to Eq. 1. Assuming loosely synchronized clocks, we expect a linear increase in the clock offset due to clock drift. This is handled by periodic re-synchronization of clock offsets in our protocol.

Fig. 9 shows that the ETT increases slightly as we keep increasing the attack intensity from A1-A6. Compared to Fig. 5b, ETT values are close to half the values of the RTT for the corresponding attack vectors. T-Pack can detect 100% of the delay for attacks with A6 intensity as values of ETT under A6 are slightly higher than half the RTT in Fig. 5b. We were able to determine that this is due to a higher RTT value obtained due to increasing clock drift not reflected in offsets (before the next clock synchronization), which results in perceived increases in ETT. (Since A6 already shows 100% coverage for its attacks, A7 is omitted.) With intrusions as intense as A6 or more, attacks can be identified whereas low intensity attacks such A2 do not result in intrusion detection as the system meets all deadlines, just as for A2 in Fig. 5b. We also observe similar overhead when enabling IPSec for Paparazzi under UDP (Fig. 10) as seen previously under TCP (in Fig. 5c).

Discussion: IPSec encrypts the packet at the network layer including the T-Pack header. This cannot be achieved with SSL as encryption is realized above network layer. Additionally, T-Pack can incorporate SSL encryption alongside IPSec without any modification. T-Pack's ability to mesh with other security protocols underlines the advantages of its simplicity without any reliance on specialized hardware.

Experiment 3: For the drone-like multi-system, subsystems EPS and MMS are selected as sender and receiver, respectively. An attack model with the same attack parameters as in Experiment 2 (A2-A7) (see Fig. 5b) assesses performance and vulnerability of T-Pack.

Fig 11 depicts the RTT as box plots (y-axis) over DDOS attack intensities (y-axis). The outliers correspond to 0.5% of

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

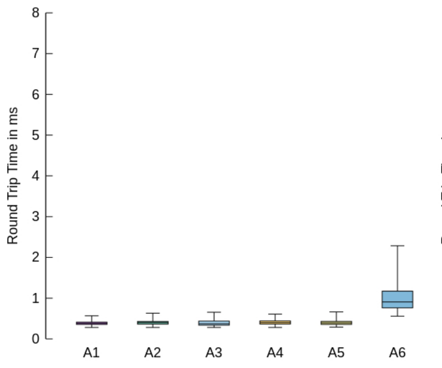


Fig. 9: T-Pack measured ETT in ms for Paparazzi over UDP without IPSec under different attack scenarios.

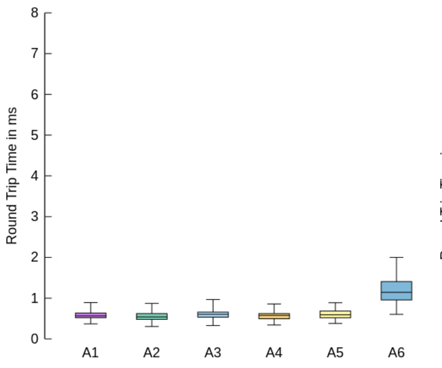


Fig. 10: T-Pack measured ETT in ms for Paparazzi over UDP with IPSec under different attack scenarios.

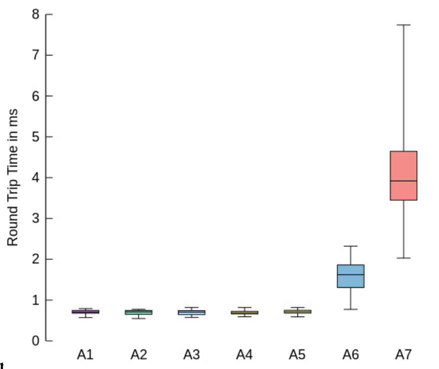


Fig. 11: T-Pack measured RTT in ms for Drone under different attack scenarios.

the total data. As attack intensity increases, a slight increase in RTT is seen — until a sudden and significant increase (A6 and A7) under more intense attacks. Under increasing attack, all RTT values eventually exceed those without attack, at which point 100% of attacks are detected with T-Pack at A7.

In summary, we experimentally demonstrated consist behavior of T-Pack for different real-time applications and varying attack intensities for DDOS ping-of-death scenarios that affect network delays, ranging from tolerated low intensity attacks without missed deadlines to high intensity attacks resulting in all deadlines being missed. Notably, deadline misses are detected early, at packet reception time (or, if a packet is omitted, at the latest scheduled reception time), which is well before a task's deadline — the earliest point of intrusion detection in the absence of T-Pack.

VII. Related Work

Prior work exploited timing bounds derived from timing analysis of code to detect malware intrusion. Zimmer et al. [37] developed techniques to provide micro-timings for multiple granularity levels of application code. They implemented a set of timed analysis methods, T-Rex, T-Prot and T-Axt, which demonstrated an advantage of timed analysis of code execution in constraining the window of vulnerability for code injections, from usually tens of millions of cycles down to tens, hundreds, or thousands of cycles, depending on the respective protection technique. In contrast, our work focuses on network protection.

Cyber-physical control systems subject to real-time constraints are vulnerable to malware intrusion over the network. Prior work [8], [9], [21] demonstrated the viability of attacks on the network of a real-time system and uncovered potential damages. Our work proposes to mitigate damages by detecting intrusion prior to such attacks using timed analysis of packets on the network, which establishes end-to-end packet delivery times allowing intrusions to be detected in a hard real-time systems, where the size of messages and time of data transmission can be bound a priori.

Time sensitive networking systems can be implemented by scheduling and traffic shaping using the IEEE standards for bridges (or switches), e.g., IEEE 802.1Qbv extensions, which could detect any abnormal flow of packets due to attacks on the network but lack any means to notify end nodes. T-Pack fills this gap as intrusion is detected on end systems, which can instantaneously transition into another mode using the Simplex architecture.

Both [2] and [27] detect attacks by analyzing time delays under secure clock synchronization. However, they propose to embed time information in packets measured at the time of transmission and received right at the end of the propagation using hardware timestamping (see [2]), which requires such support in switches/routers thereby raising cost. Instead, T-Pack embeds time inside the packet within the kernel relying on hardware support.

The IEEE 1588 standard for a precision time protocol [18] calculates end-to-end time for clock synchronization by sending prior transmission time as a message payload in follow-up packets and sends burst of such packets, which — in contrast to T-Pack — would increases the number of packets and thereby reduce network bandwidth. The 1-step method of the IEEE 1588 standard reduces the packet burst duration on the network, however, unlike T-Pack, it utilizes hardware timestamping, which raises implementation cost.

VIII. Conclusion

This work contributes the design and implementation of a novel network timed security method, T-Pack, that monitors end-to-end response times of packet delivery in the Linux network stack for early detection of malware intrusion, before a task's deadline, if time delays of packets are discovered in relative to a real-time schedule of expected packet reception times. Results indicate successful detection of malware intrusion in 95%-100% of the cases during distributed denial of service attacks that induce time delays under three application scenarios. T-Pack was further shown to incur a small overhead to the overall network performance of the system relative to the range of delays for the attacks it protects against. T-Pack combines efficiency and simplicity because of its implementation with low performance overhead without relying on specialized hardware.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.

Acknowledgment

This work was supported in part by NSF grants 1813004 and 1525609.

References

[1] Congestion Control in IP/TCP Internetworks. RFC 896, January 1984. URL: https://rfc-editor.org/rfc/rfc896.txt, `doi:10.17487/RFC0896`.

[2] Robert Annessi, Joachim Fabini, and Tanja Zseby. Securetime: Secure multicast time synchronization. *arXiv preprint arXiv:1705.10669*, 2017.

[3] S. Bak, D.K. Chivukula, O. Adekunle, Mu Sun, M. Caccamo, and Lui Sha. The system-level simplex architecture for improved real-time embedded system safety. In *IEEE Real-Time Embedded Technology and Applications Symposium*, pages 99–107, 2009.

[4] Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard, and Jeremy Tyler. The paparazzi solution. In *2nd US-European Competition and Workshop on Micro Air Vehicles*, 2006.

[5] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[6] Robert Buff and Arthur Goldberg. Web servers should turn off nagle to avoid unnecessary 200 ms delays. May 1999.

[7] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[8] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco, 2011.

[9] Chien-Ying Chen, Monowar Hasan, and Sibin Mohan. Securing real-time internet-of-things. *Sensors*, 18(12):4356, 2018.

[10] Matt Conover. w00w00 on heap overflows, 1999.

[11] T.L. Crenshaw, E. Gunter, C.L. Robinson, Lui Sha, and P.R. Kumar. The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In *IEEE Real-Time Systems Symposium*, pages 400–412, 2007.

[12] Scott A Crosby and Dan S Wallach. Denial of service via algorithmic complexity attacks. In *USENIX Security Symposium*, pages 29–44, 2003.

[13] Jean-Francois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, pages 167–182. Springer, 1998.

[14] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011.

[15] Lishoy Francis, Gerhard P Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical relay attack on contactless transactions by using nfc mobile phones. *IACR Cryptology ePrint Archive*, 2011:618, 2011.

[16] Roberto Gorrieri, Ruggero Lanotte, Andrea Maggiolo-Schettini, Fabio Martinelli, Simone Tini, and Enrico Tronci. Automated analysis of timed security: a case study on web privacy. *International Journal of Information Security*, 2(3-4):168–186, 2004.

[17] Oded Horovitz. Big loop integer protection. *Phrack Inc., Dec*, 2002.

[18] IEEE. 1588-2019 - ieee approved draft standard for a precision clock synchronization protocol for networked measurement and control systems. https://standards.ieee.org/content/ieee-standards/en/standard/1588-2019.html, month = feb, year = 2020,.

[19] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[20] H. Kopetz. The rationale for time-triggered ethernet. In *2008 Real-Time Systems Symposium*, pages 3–11, 2008.

[21] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[22] Emmanuel Ledinot. Workshop on Analysis Tools and Methodology for Embedded Systems, The 2018 Industrial Challenge. https://w3.onera.fr/waters2018/node/13, 2018.

[23] Xin Lou, Cuong Tran, Rui Tan, David KY Yau, and Zbigniew T Kalbarczyk. Assessing and mitigating impact of time delay attack: a case study for power grid frequency control. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 207–216. ACM, 2019.

[24] C Mary. Shellshock attack on linux systems–bash. *International Research Journal of Engineering and Technology*, 2(8):1322–1325, 2015.

[25] David L Mills. Internet time synchronization: the network time protocol. *Communications, IEEE Transactions on*, 39(10):1482–1493, 1991.

[26] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[27] Lakshay Narula and Todd E Humphreys. Requirements for secure clock synchronization. *IEEE Journal of Selected Topics in Signal Processing*, 12(4):749–762, 2018.

[28] Aleph One. Smashing the stack for fun and profit. *Phrack magazine*, 7(49):14–16, 1996.

[29] Gabriel Pedroza, Pascale Le Gall, Christophe Gaston, and Fabrice Bersey. Timed-model-based method for security analysis and testing of smart grid systems. In *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 35–42. IEEE, 2016.

[30] Tao Qian, Frank Mueller, and Yufeng Xin. A linux real-time packet scheduler for reliable static sdn routing. In *Euromicro Conference on Real-Time Systems*, pages 25:1–25:22, July 2017. `doi:10.4230/LIPIcs.ECRTS.2017.25`.

[31] Arman Sargolzaei, Kang K Yen, Mohamed N Abdelghani, Abolfazl Mehbodniya, and Saman Sargolzaei. A novel technique for detection of time delay switch attack on load frequency control. *Intelligent Control and Automation*, 6(04):205, 2015.

[32] Werner Schindler. A timing attack against rsa with the chinese remainder theorem. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 109–124. Springer, 2000.

[33] Fadi SHAAR and Ahmet EFE. Ddos attacks and impacts on various cloud computing components. *Int. Journal of Information Security Science*, 7:26–48, 2018.

[34] Wikipedia. Time-Sensitive Networking. https://en.wikipedia.org/wiki/Time-Sensitive_Networking.

[35] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom. The worst-case execution time problem — overview of methods and survey of tools. Technical Report MRTC report ISSN 1404-3041 ISRN MDH-MRTC-209/2007-1-SE, Maelardalen Real-Time Research Centre, Maelardalen University, March 2007.

[36] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom. The worst-case execution time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, April 2008.

[37] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan. Time-based intrusion dectection in cyber-physical systems. In *International Conference on Cyber-Physical Systems*, pages 109–118, April 2010.

[38] Christopher Zimmer and Frank Mueller. Fault resilient real-time design for noc architectures. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 75–84. IEEE, 2012.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on July 01,2025 at 03:46:26 UTC from IEEE Xplore. Restrictions apply.