Journal of *Cybersecurity and Privacy*

MDPI

Article

Tracking Real-Time Anomalies in Cyber–Physical Systems Through Dynamic Behavioral Analysis

Prashanth Krishnamurthy *, Ali Rasteh, Ramesh Karri and Farshad Khorrami

Department of Electrical and Computer Engineering, NYU Tandon School of Engineering, 5 MetroTech Center, Brooklyn, NY 11201, USA; ar7655@nyu.edu (A.R.); rkarri@nyu.edu (R.K.); khorrami@nyu.edu (F.K.)
* Correspondence: prashanth.krishnamurthy@nyu.edu

Abstract

Embedded devices in modern power systems offer increased connectivity and remote reprogrammability/reconfigurability. These features along with interconnections between Information Technology (IT) and Operational Technology (OT) networks enable greater agility, reduced operator workload, and enhanced power system performance and capabilities, as well as expanding the cyber-attack surface. This increased cyber-attack surface, as well as increasingly complex, diverse, and potentially untrustworthy software/hardware supply chains, increases the need for robust real-time monitoring in power systems, and more generally in cyber–physical systems (CPS). We propose a novel framework for real-time monitoring and anomaly detection in CPS, specifically smart grid substations and SCADA systems. The proposed framework enables real-time signal temporal logic condition-based anomaly monitoring by processing raw captured packets from the communication network through a hierarchical semantic extraction and tag processing pipeline into a time series of semantic events and observations, that are then evaluated against expected temporal properties to detect and localize anomalies. We demonstrate the efficacy of our methodology on a hardware in the loop (HITL) testbed under several attack scenarios. The HITL testbed includes multiple physical power system devices (real-time automation controllers and relays) and simulated devices (Phasor Measurement Units—PMUs, relays, Phasor Data Concentrators—PDCs), all interfaced to a dynamic power system simulator.

Keywords: integrity verification; anomaly detection; Intrusion Detection System; cyber–physical systems; smart grid; power systems; OT networks

1. Introduction

The smart grid is the next generation of power systems offering promises of a wide variety of benefits in efficiency, reliability, and safety. Some of the key features of smart grids are agile reconfigurability and dynamic optimization of grid operations, rapid detection and response to faults in the system, integration of renewable power sources with conventional fossil fuels, and providing of pervasive monitoring facilities for power systems. An important step in Industrial Revolution 4.0 is the digitization of Industry 3.0 and bringing together the Information and Communication Technology (ICT) and Operational Technology (OT) for controlling the physical processes, their monitoring, and maintenance [1–3]. In the case of power systems, smart grids are the emerging point of Industrial Revolution 4.0. IT systems in smart grid technology include different ICT servers, Communication Technology, supervisory and monitoring infrastructure, etc. On the other hand, OT comprises Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), Intelligent

Academic Editor: Martin G. Jaatun

Received: 12 December 2025
Revised: 25 February 2026
Accepted: 13 March 2026
Published: 23 March 2026

Electronic Devices (IEDs), Phasor Measurement Units (PMUs), relays, Human–Machine Interfaces (HMIs), etc. [3,4]. Seamlessly combining ICT and OT can provide efficient methods to augment the capabilities of smart grids. However, the resulting expanded connectivity and remote programmability/reconfigurability can broaden the attack surface and increase cybersecurity vulnerabilities [5,6]. Recent examples of attacks on power grids and industrial control systems (ICS) show the crucial importance of these systems and the extensive impacts that can result if the security of these systems is compromised. For example, the coordinated cyber attack on the Ukrainian power grid in 2015 caused a power loss for 6 h, affecting about 225,000 customers in Ukraine [7]. As another example, the well-known Stuxnet malware, which is known to use zero-day covert attacks, was used to attack nuclear industrial systems in 2010 [8]. Other recent examples of sophisticated attacks on ICS/CPS include TRITON [9] (also known as Trisis and HatMan), which targeted safety instrumented systems in a petrochemical plant in 2017; Industroyer2/Sandworm [10], which targeted IEC-104 SCADA communications in Ukrainian power infrastructure in 2022; the Oldsmar water treatment facility attack [11] in Florida, where an attacker gained remote access and attempted to alter the sodium hydroxide (lye) levels to dangerous values; FrostyGoop [12], which caused heating system service disruptions in Ukraine in 2024 by injecting unauthorized Modbus TCP commands; Fuxnet/Blackjack [13], which performed flood attacks on sensor networks in a Russian underground water and sewage and communications infrastructure company in 2024; ELECTRUM-linked attack [14] on the power grid in Poland in 2025; and the emergence of modular "malware frameworks" such as Pipedream [15] (often described as a "Swiss army knife" of malware) that are designed to be able to target a wide range of industrial control devices. These incidents underscore the evolving ICS/CPS threat landscape and the urgent need for robust security frameworks for smart grids and other ICS/CPS.

Industrial control systems, including smart grids, are complex systems consisting of embedded device nodes interconnected by communication networks and interfaced to physical processes. Relevant devices for smart grids include, for example, MTUs (Master Terminal Units), RTUs, RTACs (Real-Time Automation Controllers), PLCs, relays, PMUs, PDCs (Phasor Data Concentrators), and HMI. These devices communicate using various protocols such as DNP3, IEEE C37.118, Modbus, IEC61850, SEL Fast Msg, OPC-UA (Open Platform Communications Unified Architecture), IEC 60870-5, etc., which are all industrial communication standards primarily developed several decades back without specific focus on security. The temporal evolution of the cyber–physical system is governed by the device behaviors (e.g., logic/rules programmed on the devices), the communications/interactions between the devices, and the physical dynamics of the system. A malicious manipulation of a device behavior or of the communication network (e.g., device spoofing, packet injection or manipulation, etc.) by an intruder/adversary can lead to catastrophic consequences in the power grid, including destabilization of the grid and damaging physical components in the grid. Therefore, techniques to monitor these interactions and processes in real time and flag any anomalies with low latency and high accuracy would be vitally beneficial to the security of the grid. Such a technology should not only consider the basic communication specifications between the grid devices but also whether the observed temporal processes are consistent with the expected behaviors and dynamics of the grid. Since the smart grid is a composite of the controller devices, power system dynamics, network communication channels, and interplay between these components, a comprehensive monitoring system should be able to track the temporal behaviors in real time and detect any abnormalities. Analogously to anomaly monitoring systems for other cyber–physical systems [16] and autonomous vehicles [17], such a comprehensive monitoring system should span controller-focused anomaly monitoring (CFAM) for validating behaviors of controllers and other

devices in the grid, network-focused anomaly monitoring (NFAM) for validating network-level transactions and statistics, system-focused anomaly monitoring (SFAM) for validating temporal process dynamics, and cross-domain anomaly monitoring (CDAM) for validating the interplay between controller/system/network components (Figure 1).

To address this crucial need, we develop a real-time integrity verification methodology (TRAPS—Tracking Real-time Anomalies in Cyber-Physical Systems) in this paper to detect abnormal behaviors in the power system by continuous dynamic behavioral analysis of the cyber–physical system. The proposed TRAPS approach is based on a Signal Temporal Logic (STL) condition-based anomaly monitoring and Intrusion Detection System that processes real-time observations from communication network packet captures through a hierarchical semantic extraction and Directed Acyclic Graph (DAG)-based tag processing pipeline to transform them into a time series of semantic events and observations, collectively referred to as semantic tags. These semantic tag time series are then evaluated against expected temporal properties to detect and localize anomalies and visualize them in a dashboard graphical user interface (GUI). The contributions of this paper are as follows:

- Development of a flexible and scalable end-to-end framework that can directly operate on streaming raw network traffic and map back in real-time to the spatio-temporal operation of the CPS and evaluate integrity relative to high-level semantic behavioral properties of the CPS
- Development of a DAG-based hierarchical tag processing framework enabling iterative transformations of time series of semantic tags and an integrity verification framework that enables real-time monitoring of configurable STL-based behavioral specifications defined over the hierarchically computed time series of semantic tags.
- Demonstration of the efficacy of the proposed methodology with several attack scenarios on a hardware-in-the-loop (HIL) testbed, which includes both physical and virtual power devices, interfaced to a dynamic power system simulator.

The proposed TRAPS approach enables essentially a distributed sequence-of-events monitor that outputs time-series observations of semantic variables that can be iteratively processed according to configurable DAG-based definitions and dynamically queried for integrity verification against configurable STL-based behavioral specifications. Furthermore, a key aspect of the approach is that the set of behavioral specifications to be monitored is open and extensible so as to be customizable to meet the needs of particular CPS. The generality and flexibility of the behavioral specification structure facilitates the encoding of various types of expected semantic properties of the CPS (e.g., device control logic that implies that some events should be followed by some other events, physics models that indicate changes in physical signals based on the operation of devices such as relays, communication configuration implying that separate communications between certain device pairs should have either temporal or value-based dependencies, etc.) to any desired level of detail. The hierarchical tag processing and STL-based integrity verification engine provide a real-time semantic view of the CPS operation on top of which any STL-based behavioral specifications can be configured for monitoring and the proposed framework enables a fully automated data flow from raw traffic to real-time semantic integrity verification and alert generation based on the configured specifications.

This paper is organized as follows. The related literature is reviewed in Section 2. We describe the proposed methodology in Section 3, including the threat model and problem formulation (Section 3.1), network packet parsing (Section 3.2), observation set extraction and processing (Sections 3.3 and 3.4), integrity verification (Section 3.5), anomaly localization (Section 3.6), and the visualization dashboard (Section 3.7). Section 4 presents experimental results on a hardware-in-the-loop testbed demonstrating the efficacy of

the proposed framework under various attack scenarios. Section 5 provides concluding remarks with a summary and directions for future work.

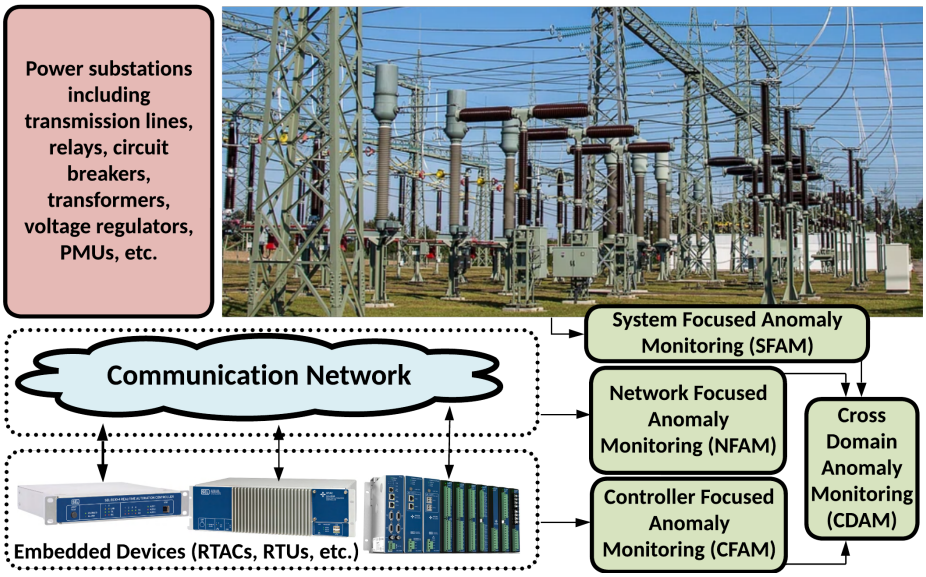

Figure 1. Proposed TRAPS multi-domain monitoring approach with a unified framework for network-focused anomaly monitoring (NFAM), controller-focused anomaly monitoring (CFAM), system-focused anomaly monitoring (SFAM), and cross-domain anomaly monitoring (CDAM).

2. Related Works

Several types of attacks on smart grid systems have been considered in the literature (e.g., [5,6,18–20]) including measurement integrity attacks, false data injection (FDI), false command injection (FCI), control logic modification, and denial of service (DoS) attacks including time-delay and jamming attacks. Coordinated attacks, which use multi-stage complicated patterns to increase attack efficacy, and cascading attacks exploiting a single point of failure to propagate the effect to the other points of the system have also been considered [3,21]. To defend against the various attacks, defenses (termed in general as Intrusion Detection Systems or Anomaly Detection Systems) have been developed using a variety of approaches and underlying techniques as discussed below.

Signature-based methods use a "blacklist" of signatures of prior attack/anomaly events to detect intrusions of the same category, but cannot detect unknown/zero-day attacks with new signatures. Examples of methods of this type include [18] which detected machine-in-the-middle (MITM) attacks on the DNP3 protocol through Snort rules [22], which applied the ML-based fusion of cyber and physical sensors to detect FCI/FDI attacks on DNP3 protocols, and [23] which applied Suricata, an open-source network IDS, to detect anomalies for network protocols, including IEC61850, based on software rules.

Specification-based methods model the system's behavior using its specifications, especially at the network level, and analyze the observed behavior such as the communication protocol details to detect abnormalities. Typical limitations in the available methods of this type include support for only specific protocols, limited scalability, and the ability to monitor only specific types of behaviors/events. Specification/behavior-based IDS have been developed considering various protocols such as IEEE C37.118 [24,25], DNP3 [26], IEC60870 [27], IEC61850 [28,29], and Modbus/TCP [30]. Combinations of multiple IDS approaches have also been studied such as: the combination of signature-based and model-based methods using Snort in [27]; combination of access-control, protocol whitelisting, model-based, and multi-parameter-based detection methods in [31]; combinations of host-based and network-based detection methods in [32]; and the combination of access-control,

protocol-based, and behavioral whitelists [33]. Process-aware monitoring methods based on knowledge of the underlying CPS behavior have been studied [21]. Other specification-based approaches in the literature include the monitoring of values of process variables in terms of rules defined in a specific description language in [34], state tracking methods [35,36], and sequence of events monitoring using a Discrete-Time Markov Chain model in [37]. Moving-target defense methods against false data injection attacks in the context of state estimators have been studied [38–41] based on dynamically altering some aspects of the system configuration, such as changing line impedances using distributed flexible AC transmission system (D-FACTS) devices. In the broad context of CPS across different domains, STL-based methods have been developed for monitoring and analysis for both continuous-time and discrete-time signals. Recent advances in this direction include cumulative-time extensions to STL and associated monitoring algorithms [42] based on evaluating the sum of all timesteps for which an STL formula is true, informative online monitoring for STL [43] based on causation and relevance evaluations to provide more informative context for STL violations, and the formally proved compilation of STL fragments into synchronous observer implementations [44]. STL methods have also been developed for hybrid systems with both discrete and continuous components using SMT-based robust model-checking techniques [45]. Formal control system approaches have been leveraged to enable model predictive monitoring of dynamical systems under STL specifications [46] by assuming that the observed state signal traces are generated by a dynamical system with a known model but unknown control signal. Tool support for STL-based monitoring is also maturing such as RTAMT [47], which provides online/offline monitor implementations designed to integrate with the Robot Operating System (ROS) and with Matlab/Simulink. Recent work has also begun to address privacy/security aspects of monitoring itself, such as oblivious monitoring for discrete-time STL using fully homomorphic encryption [48]. The proposed TRAPS framework is synergistic and complementary with these works, which address monitors over explicit system signals/models, while the primary focus of TRAPS is on the complementary problem of the extraction of semantically meaningful time-series observations from heterogeneous OT network traffic and then monitoring open and configurable STL behavioral specifications over those derived tags for real-time integrity verification of power grid CPS.

Learning-based methods use data-driven machine learning to detect anomalous or abnormal patterns in the system's traffic/signals. Challenges when applying these methods include difficulty in obtaining extensive training datasets, lack of explainability of ML prediction results that can also make it difficult to localize underlying causes of detected anomalies and guide appropriate remediations, and limitations in generalizability under changes in data distribution (domain shift). Learning-based methods [49–51] have been applied, for example, to the detection of false data injection attacks [52–55], jamming [56], and time-delay attacks [57] and anomaly detection in transmission protective relays [58], wide-area protection systems [59], distribution systems [60,61], and Modbus communications [62]. Host-based anomaly detectors using analog/digital side channels such as system calls and Hardware Performance Counters (HPCs) have been developed (e.g., [63–65]). Recent work has also explored *hybrid* approaches that combine formal/specification-based monitoring with data-driven learning, aiming to combine the benefits of interpretability and well-definedness provided by the specifications approach while improving adaptivity and robustness in complex CPS environments by leveraging data-driven methods. For example, hybrid knowledge-driven and data-driven techniques have been proposed to synthesize run-time monitors for CPS by combining prior domain knowledge with learned models from data in [66]. Learning-based time-series anomaly detection methods have also been applied to extract informative representations from raw signals. Approaches in

this direction include self-supervised disentangled reconstruction-based representation learning for time-series anomaly detection by learning both recurrent/consistent patterns and irregular variations in the latent space [67] and transformer-based architectures combined with probabilistic filtering to identify anomalous CPS signals [68] by capturing the dynamics and temporal dependencies in CPS within a dynamic state-space model. In related research, methods have also been developed to make monitoring more efficient and adaptive at run-time, e.g., self-triggered strategies for STL monitoring tasks that reduce monitoring effort (and thereby computational burden and energy expenditure) when the system appears to be behaving nominally [69].

Real-time processing methods operating on streaming data have been addressed in recent work to transform raw telemetry and event streams into structured higher-level representations (*semantic streaming*) to improve interpretability and facilitate downstream applications to CPS monitoring and intrusion/anomaly detection. For example, in [70], a semantic analysis approach combined with self-supervised embeddings and geospatial context features was proposed to enhance intrusion detection for IoT and sensor networks by extracting more meaningful representations from streaming observations. In a broader streaming analytics context under varying data distributions (concept drift), comparative evaluations and benchmark-driven analyses have been addressed in [71,72], studying practical trade-offs among different anomaly detection methods in online settings. Also, in a broader CPS context, semantic event-handling architectures aimed at building explainable CPS (ExpCPS) have been developed in [73] by structuring event processing pipelines around semantic abstractions rather than raw signals based on a semantic event-handling module that is designed to be integrated into ExpCPS architectures across different domains.

In contrast to prior methods discussed above, the key benefits of TRAPS are: a unified framework for the monitoring of at-scale heterogeneous communication traffic against an open and extensible set of behavioral properties using hierarchical semantic tag processing and STL-based monitoring in a protocol-agnostic and extensible framework for real-time validation of the entire cyber–physical loop; end-to-end pipeline from raw network packet capture to protocol-agnostic semantic parsing, semantic tag extraction, situational awareness, integrity verification, anomaly detection, localization, and visualization; and computational simplicity and scalability enabling real-time processing of high-bandwidth traffic and simultaneous monitoring of several hundreds of tags and STL conditions. Unlike approaches that focus on specific attack types (e.g., delays, DoS, false data injection), TRAPS verifies semantic event sequences across multiple devices and domains enabling dynamic end-to-end auditing of behavioral specifications that can span correlations, causations, and other CPS behavioral properties.

TRAPS is synergistic and complementary with emerging CPS/OT security trends such as the incorporation of verifiable data flow and data query techniques. In particular, blockchain-based mechanisms [74], such as verifiable decentralized identities and data integrity, can enable secure and verifiable data flows for the cyber–physical Web 3.0 [74]. Also, advanced data query systems, such as VQL (Verifiable Query Layer) [75] and TeLEx (Two-Level Learned Index for Secure Queries) [76], offer enhanced efficiency and security for querying large-scale distributed and blockchain systems. While VQL provides cloud-deployable, efficient, and cryptographically verifiable data query services for blockchain systems, TeLEx introduces a two-level learned indexing methodology for enabling rich query functionalities on enclave-based blockchain systems by leveraging Trusted Execution Environment (TEE) and oblivious RAM techniques. These emerging techniques which utilize the immutable and decentralized nature of blockchain systems to facilitate robust and secure data management/queries within CPS can provide vital benefits synergistic with CPS monitoring solutions, such as TRAPS, by ensuring trust in

the data shared/queried across distributed components. Real-time monitoring frameworks such as TRAPS complement blockchain technologies by enabling the dynamic validation of behavioral properties to flag anomalies, intruders, or other compromises of the CPS. These technologies collectively strengthen defenses against dynamic threat actors with emerging adversarial tactics, thereby offering robust defense-in-depth solutions.

3. The Proposed Method

The proposed framework is based on the pipeline architecture shown in Figure 2, where raw data is processed through a sequence of layers to extract time-series observations of hierarchically defined semantic tags, that are then used for anomaly detection relative to a set of STL-based behavioral specifications and visualization in an operator dashboard. The threat model and problem formulation are summarized in Section 3.1. The individual components of the proposed framework are then discussed in the following subsections.

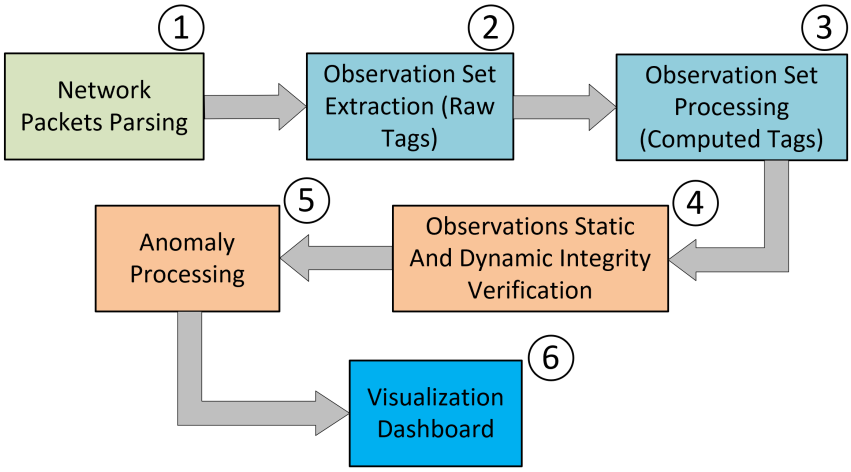

Figure 2. Overall structure of the proposed method.

3.1. Threat Model and Problem Formulation

The threat model addressed is formally defined below along with the key elements of the considered problem formulation.

Protected Assets and Security Objectives: The protected assets in a CPS such as the smart grid include: (i) the correct operation of devices (e.g., relays, RTACs, PMUs, PDCs), (ii) the integrity of network communications between devices, and (iii) the integrity of physical process behavior. The security objective of the defender is to detect deviations from expected CPS behavior in real-time with low latency and high accuracy, so as to enable rapid response to mitigate potential damage.

System State and Observations: Let $x(t) = (x_d(t), x_p(t), x_c(t))$ denote the system state at time t, where $x_d(t)$ represents the internal states of devices (e.g., relay logic states, controller variables), $x_p(t)$ represents the physical system states (e.g., voltages, currents, power flows), and $x_c(t)$ models a representation of the "communication states" (e.g., message sequences, timing). The defender does not have direct access to $x(t)$, but instead observes network traffic through a monitoring point (e.g., an RSPAN port). Let $\mathcal{P} = \{P_i = (t_i, s_i, d_i, p_i, m_i)\}_{i=1}^{n}$ denote the time series of observed network packets, where t_i is the timestamp, s_i and d_i are source and destination identifiers, p_i is the protocol/message type, and m_i is the payload content. Through the semantic extraction pipeline described in Sections 3.3 and 3.4, the raw packets are transformed into an observation set $\mathcal{O} = \{(t_j, st_j, v_j)\}_{j=1}^{m}$ of semantic tags, where st_j is a tag identifier and v_j is the corresponding value.

Adversary Model: We consider an adversary who gains unauthorized access to the OT network of the CPS and introduces perturbations δ to either device behavior or network

communications. Formally, the adversary can modify the effective system state to add a perturbation as $\tilde{x}(t) = x(t) + \delta(t)$, where $\delta(t)$ represents adversarial perturbations such as:

- Firmware/logic modifications on devices such as relays or RTACs (thereby affecting x_d), e.g., altering relay control logic or masking/delaying commands (and therefore possibly indirectly affecting also x_p and x_c);
- Insertion of MITM devices to manipulate communications between devices (thereby affecting x_c and possibly indirectly x_p and x_d), e.g., modifying, delaying, replaying, or dropping messages;
- Injection of unauthorized network traffic, e.g., false commands or flood attacks for denial of service (thereby affecting x_c and possibly indirectly x_p and x_d).

Adversary Capability Bounds: The adversary may compromise one or more devices or communication links, potentially simultaneously (e.g., modifying firmware/logic of a device while also simultaneously injecting network traffic, masking sensor messages from multiple devices, etc.). However, we assume that the adversary cannot manipulate *all* observations relevant to a given behavioral specification so as to completely mask the existence of an anomaly. Formally, for each STL-based behavioral specification ϕ defined over a subset of tags $\mathcal{T}_\phi \subseteq \mathcal{O}$, we assume there exists at least one tag in $\mathcal{T}_\phi$ whose observations remain uncompromised. This assumption is consistent with typical CPS attack vectors where adversarial access originates from specific entry points (e.g., compromised firmware on specific devices, MITM on specific communication links, intruder device sending spurious commands). Hence, adversarial effects are localized to parts of the CPS that the adversary has gained access to and cannot feasibly affect all observable network communications. Furthermore, the heterogeneity of devices and protocols in real-world CPS deployments typically makes it infeasible for an attacker to simultaneously control all observable communications. Also, note that there is no assumption that any specific measurements or communication channels are trusted *a priori*. Rather, the proposed framework's robustness derives from the ability to define behavioral specifications that span multiple independent observation sources, requiring attackers to compromise multiple independent parts of the CPS to evade detection (e.g., both the command path and the measurement path). As with any anomaly detection approach, we assume that the adversary is not so powerful that they can manipulate all relevant measurements so as to completely mask the existence of an anomaly since such an adversary of unlimited capacity who can manipulate all observations can always elude detection.

Attack Classification: A broad categorization of attack types relevant to power grid CPS/OT environments is summarized in Table 1 along with representative examples of the attack types and their effects on system behavior and corresponding STL-based specifications, deviations from which are aimed to be detected by the TRAPS framework. These attack categories map naturally to the MITRE ATT&CK Matrix for ICS [77] and MITRE EMB3D [78] threat model frameworks, which provide detailed taxonomies of cyber kill chain elements (tactics, techniques, and procedures—TTPs) and device vulnerabilities, respectively, in the ICS/CPS context. MITRE ATT&CK lists TTPs across the several stages of a cyber-attack lifecyle, ranging from initial access to eventual impact. Components of various stages such as network connection enumeration (in Discovery stage), adversary-in-the-middle (in Collection stage), denial of service (in Inhibit Response Function stage), unauthorized command message (in Impair Process Control stage), and loss of control (in Impact stage) map directly to the attack categories in Table 1. The MITRE EMB3D framework organizes embedded device vulnerabilities into a threat heat map across networking, hardware, system software, and application software domains. The attack types in Table 1 primarily draw from scenarios modeled from networking (e.g., TID-404—Remotely Triggerable Deadlock/DoS, TID-406—Unauthorized Messages or Connections, TID-407—Missing

Message Replay Protection, TID-412—Network Routing Capability Abuse), system software (e.g., TID-202—Exploitable System Network Stack Component, TID-204—Untrusted Programs Can Access Privileged OS Functions, TID-205—Existing OS Tools Maliciously Used for Device Manipulation, TID-211—Device Allows Unauthenticated Firmware Installation, TID-213—Faulty FW/SW Update Integrity Verification, TID-215—Unencrypted SW/FW Updates), and application software (e.g., TID-301—Applications Binaries Modified, TID-304—Manipulate Run-Time Environment, TID-309—Device Exploits Engineering Workstation, TID-311—Default Credentials, TID-328—Hardcoded Credentials) device vulnerability categories in the MITRE EMB3D framework.

Table 1. Classification of attack types and their effects on CPS behavior and STL-based specifications.

Attack Type	Examples	Effects on CPS Behavior and STL-Based Specifications
Network Attacks	DoS (denial of service), MITM (machine-in-the-middle), replay attacks, unauthorized nodes, port scanning, covert channels, data exfiltration	Disrupts communication availability, integrity, or expected inter-node traffic patterns. Violates protocol-level timing specifications (e.g., max latency constraints) or sequence specifications (e.g., request-response patterns).
Data Manipulation	False data injection (FDI), command spoofing, sensor manipulation, measurement scaling, status readings modifications	Alters semantic values of sensors or commands. Violates physics-based value specifications (e.g., voltage/current limits) or consistency specifications (e.g., correlation between redundant/related sensors).
Device Compromise	Logic modifications, firmware tampering, supply chain attacks, multi-device attacks, parameter manipulation, HMI attacks	Modifies internal device control logic. Violates input–output logic specifications (e.g., relay trip logic) or state transition specifications (e.g., unexpected device status changes).
Communication Disruption	Time delays, data gaps, protocol modifications, packet manipulation	Introduces timing irregularities or protocol deviations. Violates timing specifications (e.g., periodic reporting rates) or real-time delivery constraints.

System and Trust Assumptions:

- *Network Observability:* The defender has access to a network monitoring point (e.g., RSPAN) that provides visibility into all relevant OT network traffic in the CPS.
- *Timing:* Observations are timestamped at the monitoring point (e.g., an RSPAN port) based on packet arrival times, thereby providing a common reference clock, which is not derived from device-local clocks. The framework does not require clock synchronization across distributed CPS devices since all timing is relative to the monitoring point's clock, thereby avoiding issues with clock desynchronization or drift across devices. Furthermore, since the timing thresholds in timing-based behavioral specifications (e.g., time windows in pre-/post-conditions) are configurable, they can be set to accommodate typical network latencies and timestamp jitter in the specific deployments.
- *System Behavioral Specification:* The behavioral specifications of the CPS are defined as a set of STL properties based on the expected behavior of the CPS (e.g., device control logic, physics constraints, communication configurations). These properties are configured based on CPS design documentation (e.g., DNP3 point map lists, relay control logic documentation) and formal specifications and verified through historical "golden" traces. However, the behavioral specifications can, in general, be incomplete (e.g., missing characterizations of control logic of some devices) in which case the proposed framework enables the detection of deviations from the specifications that are included. To handle potential incompleteness, the framework adopts a "safety envelope" approach, enforcing the defined subset of critical properties (e.g., safety constraints) rather than requiring a complete model of all CPS behaviors.

This structure supports incremental maintenance, allowing operators to refine or add behavioral specifications over time without system downtime. Additionally, the framework facilitates robustness by allowing the definition of behavioral properties that span diverse, independent domains (e.g., physical states, network timings, control logic), thereby enabling robust and sensitive anomaly detection that is not reliant on overly constrained or brittle specifications of any particular single-point/single-sensor behavioral properties.

Defender Objective: The task for the defender is to enable real-time mapping from the raw network traffic $\mathcal{P}$ (which may comprise multiple OT communication protocols) to the higher-level semantic observation set $\mathcal{O}$, and to enable continuous evaluation of $\mathcal{O}$ against a set of STL-based behavioral specifications $\Phi = \{\phi_1, \ldots, \phi_K\}$ to detect and localize any deviations as anomalies.

Defender and Attacker Success Criteria: The success criteria for the defender and attacker are defined as follows:

- *Defender success:* The defender succeeds when the framework achieves (1) high detection rate, i.e., all adversarial actions that cause a violation of at least one behavioral specification $\phi \in \Phi$ are detected; (2) low false positive rate, i.e., normal CPS operation that satisfies all behavioral specifications does not trigger anomaly alerts; and (3) low detection latency, i.e., anomalies are flagged within a short time window after the occurrence of the violating observation.
- *Attacker success:* The attacker succeeds if they achieve their operational objective (e.g., manipulating physical process behavior, injecting false commands) while evading detection by the anomaly monitoring framework.

3.2. Network Packets Parsing

The raw network traffic (either live or as a pcap) that is the input to TRAPS comprises of the communications between the various devices in the smart grid using several different protocols such as the following supported by our current implementation of our system: DNP3, IEEE C37.118, Modbus, IEC61850 GOOSE, IEC61850 MMS, and SEL Fast Msg (a proprietary protocol by Schweitzer Engineering Laboratories Company), IEC60870-5, OPC-UA, and Telnet protocols. Two versions of our framework were implemented, as discussed further in Sections 3.8 and 4.2.4. In the first version, which was primarily Python-based (with some computational hot spots implemented in C++), the network packet parsers were implemented as a set of scripts based on open-source libraries such as Scapy [79], Pyshark [80], and the Hammer library [81] to process the network traffic to parse and extract the payload contents using methodologies analogous to [82,83]. For parallel processing, the parser components were structured as a set of separate Docker containers for each communication protocol with a front-end ingest module to detect the application layer protocol for each incoming packet and forward it to the appropriate protocol-specific parser for extracting the payload contents. The outputs of the protocol-specific parsers were combined into an MQTT streaming feed that is then used by the semantic tag processing component. In addition to the MQTT feed, the combined output stream from the parsers was also exposed via a REST API interface, with both the push (streaming) and pull (REST API) interfaces to the parser outputs supporting filtering on properties such as IP addresses, protocols, and message types. In the second version of our framework, which is primarily Go-based with the architecture discussed in Section 3.8, the network packet parsers are instead generated via declarative specifications using the Kaitai framework, which yields highly efficient binary payload parsers. These Kaitai-generated codes are then run in parallel using lightweight Go green threads (goroutines) with channel-based message passing and synchronization. This architecture yields significantly higher performance as

discussed further in Section 4.2.4. The overall TRAPS prototype is structured such that the downstream components including the semantic tag processing can run as separate threads in the same process (obtaining data from the parsers via in-process channels) or as a separate process (in which case the data is passed via MQTT as before). When running as a separate process, the downstream components can run on a separate machine for even more efficient parallelization (as well as potentially enabling a distributed network of network capture nodes with local parser components feeding to a centralized machine for semantic tag processing and anomaly detection).

3.3. Observation Set Extraction

The output of the protocol-specific parsers is a time series $\mathcal{P}$ of records of the form $P_i = (t_i, s_i, d_i, p_i, m_i), i \in [1, n]$ where

- t_i is the packet's timestamp;
- s_i and d_i are the source and destination of the packet, respectively, which could be IP and/or MAC addresses depending on the protocol;
- p_i is the protocol and message type of the packet;
- m_i is the set of measurements/values in the packet's payload such as analog and digital values in IEEE C37.118, Modbus coils, Modbus holding registers, DNP3 analog inputs and outputs, etc.

The specific information mapping (i.e., which fields in a DNP3 message correspond to what physical quantities) are installation-specific and can vary widely. Hence, after the parsing of raw fields in the network packets, a key step is mapping the fields to semantic variables. For this purpose, TRAPS uses a flexible query set structure wherein functions defined over the raw fields are used to populate the values of semantic variables as appropriate for the particular installation and the particular network communication protocol. For example, in IEEE C37.118, the constituent fields are typically phasors, analogs (e.g., currents and voltages), and digitals (e.g., status values). The queries to extract semantic variables ("raw tags") from the time series $\mathcal{P}$ is defined as a set of packet filtering rules of the form $R_i = (s_i, d_i, p_i, a_i, st_i), i \in [1, m]$ where s_i, d_i, and p_i have the same meaning as in $\mathcal{P}$, a_i is an attribute address specifier (e.g., index of the data in DNP3 binary inputs/outputs, address of an input register in Modbus, etc.), and st_i is a tag identifier to be raised (along with the corresponding timestamp) whenever the filtering rule is triggered due to a matching (s_i, d_i, p_i, a_i). Note that multiple packet filtering rules could be triggered by a single packet. The algorithmic structure of this component is shown in Algorithm 1.

Algorithm 1 Filtering time series of parsed packets to generate time series of raw tags

```
for packet P_i in parsed packets do
    for j = 1 to m do
        R_j = (s_j, d_j, p_j, a_j, st_j)
        if (s_i, d_i, p_i) == (s_j, d_j, p_j) then
            if attributes addressed by a_j exist in P_i then
                Extract attributes addressed by a_j from P_i
                Push st_j into output time series (with corresponding timestamp)
            end if
        end if
    end for
end for
```

To reduce the manual configuration effort in configuring the packet filtering rules which map raw fields in network traffic packets to semantic variables, TRAPS includes utility scripts to automatically ingest substation-specific configuration files in standard formats

(such as CSV-based point lists for DNP3/Modbus, CSV-based phasor and analog/digital element lists for C37, and Substation Configuration Language or SCL files for IEC 61850) and generate the packet processing and semantic tag extraction rules. These scripts internally use the Python-based API of our framework to enter the rules into the underlying system. The utilization of automated ingestion scripts ensures that the installation-specific mappings are derived directly from the power system design documents, thereby streamlining deployment and reducing the risk of configuration errors.

3.4. Observation Set Processing

Since the behavioral properties of the CPS might be most naturally described not in terms of raw tags but in terms of variables that are computed as functions of multiple tags over multiple time instants. Hence, TRAPS includes a hierarchical tag processing engine that allows definitions of computed tags as functions of other raw/computed tags. To facilitate a flexible structure for defining hierarchical dependencies of computed tags, a DAG $\mathcal{D}$ is used in which each node represents a time series of a particular tag and is constructed as a function of the previously extracted time series of tags in the node's dependency list. The functional dependency structure is represented as a set of filtering rules of the form $C_k = (Dep_k, f_k, st_k), k \in [1, s]$ where Dep_k denotes the dependency list (of raw/computed tags), f_k is a function encoding the calculations required to obtain updated values of C_k (along with their corresponding timestamps) from the time-series values of the tags in the dependency list, and st_k is a tag identifier to be raised whenever the filtering rule is triggered, similar to the corresponding designator for raw tags. The algorithmic structure of this component is shown in Algorithm 2 where the input queue $\mathcal{Q}$ holds both raw tags raised from Algorithm 1 and computed tags pushed as part of Algorithm 2 (to iteratively process downstream dependencies in the DAG). The time series of observations for each semantically extracted tag is of form $P = \{(t_i, v_i)\}_{i=1,\ldots,n}$ with t_i, v_i being the timestamp and value, respectively, of that tag.

Algorithm 2 Extracting time series of computed tags

```
while True do
    Get next tag q from queue Q (or wait until there is one).
    for each child node k of q in DAG D do
        Compute f_k using time-series values of tags from Dep_k and add computed value to time series of observations for tag st_k.
        Push st_k to Q.
    end for
end while
```

3.5. Observation Set Static & Temporal Integrity Verification

A crucial property of the CPS is that its expected behavior (as defined by device logic, system dynamics, etc.) implies various correlations/causalities among the time series of tags. These include both dependencies/relationships between values of two time series (e.g., expected relationships between relay open/closed status and voltage values) and temporal properties (e.g., an event in one time series expected to happen before or after an event in a second time series). These relations could stem from physics-based and behavior-based properties of the CPS. For example, physics-based properties result from power system physical laws such as the Kirchhoff's voltage and current laws (e.g., interdependencies between PMU measurements at different locations) while behavior-based properties relate to device configurations, network characteristics, etc. For example, in the case of devices such as proxies, protocol converters, or command forwarders in the smart grid, the values of corresponding time series from before-proxy and after-proxy traffic

have expected relationships, both in terms of the numerical values of payload contents and temporal relationships between messages (e.g., time delays before retransmission). Deviations from expected correlation/causality relations indicate anomalies or abnormal behavior that could stem from cyber attacks, physical attacks, or physical malfunctions.

To enable flexible monitoring spanning these various types of correlation/causality relations, we define several types of condition structures discussed below that could hold between time series of different tags. To show examples of the condition structures, we use the notations $P_1 = \{(t_{1,i}, v_{1,i})\}_{i=1,\ldots,n}$, $P_2 = \{(t_{2,j}, v_{2,j})\}_{j=1,\ldots,m}, \ldots, P_r = \{(t_{r,k}, v_{r,k})\}_{k=1,\ldots,q}$ to denote the time series of observations of various tags.

- *Threshold conditions* such as

$$|v_{1,i} - v_{1,i-1}| \leq V_{th} \tag{1}$$

$$\underline{T}_{th} \leq |t_{1,i} - t_{1,i-1}| \leq \bar{T}_{th} \tag{2}$$

  where V_{th}, $\underline{T}_{th}$, and $\bar{T}_{th}$ denote the value threshold, lower timing threshold, and upper timing threshold, respectively, for the observations.
- *Match conditions* such as

$$|v_{1,i'} - v_{2,j'}| \leq V_{th} \tag{3}$$

  where i' and j' denote matching time instants of the time series of observations P_1 and P_2 (e.g., time values such that $t_{1,i'} \approx t_{2,j'}$). More generally, functional match conditions (with an arbitrary function f) across multiple time series of observations can be defined by the form

$$f(v_{1,i'}, v_{2,j'}, \cdots v_{r,k'}) = 0 \tag{4}$$

  where $i', j', \ldots, k'$ denote matching time instants across the different time series of observations (e.g., time values such that $t_{1,i'} \approx t_{2,j'} \approx \ldots \approx t_{r,k'}$).
- *Pre-conditions* are conditions that an event (defined in general in terms of values from one or more time series of observations) should have been preceded by some other defined event within some time interval; for example,

$$f_1(v_{1,i}) = 0 \Longrightarrow \exists t_{2,j} \in [t_{1,i} - T_{th}, t_{1,i}) \quad s.t. f_2(v_{1,i}, v_{2,j}) = 0 \tag{5}$$

  where f_1 and f_2 are arbitrary functions and T_{th} is a threshold on timing. For example, a condition that the time series of observations P_1 should track (possibly with a delay) the time series of observations P_2 would be represented with $f_1(v_{1,i}) \equiv 0$ and $f_2(v_{1,i}, v_{2,j}) = \max\{|v_{1,i} - v_{2,j}| - V_{th}, 0\}$ where V_{th} is a threshold for the matching of $v_{1,i}$ and $v_{2,j}$.
- *Post-conditions* are conditions that some specified event should be followed by some other defined event within some time interval; for example,

$$f_1(v_{1,i}) = 0 \Longrightarrow \exists t_{2,j} \in [t_{1,i}, t_{1,i} + T_{th}) \quad s.t. f_2(v_{1,i}, v_{2,j}) = 0 \tag{6}$$

  where f_1 and f_2 are arbitrary functions and T_{th} is a threshold on timing.

The algorithmic structure of the integrity verification component for flagging condition violations is shown in Algorithm 3, where $\mathcal{C}$ denotes the set of all conditions defined in a particular deployment configuration. Besides the example structures above, the conditions can involve dependencies on arbitrary numbers of tags as well as the time history of the

tags. Also, the conditions can involve other similarity measures between time series such as L_p norms/distances, dynamic time warping, correlation measures, etc.

Algorithm 3 Algorithm for flagging condition violations

1: **for** condition $c \in \mathcal{C}$ **do**
2: **if** deviation from condition check as in (1)–(6) **then**
3: Push anomaly detection flag on c to anomaly queue $\mathcal{M}$ with metadata on timestamp and variables used in computation of condition c
4: **end if**
5: **end for**

As discussed above, the integrity verification engine operates by evaluating time series of semantic tags against the configurable set of STL-based specifications, encompassing both static constraints (such as instantaneous value thresholds, allowed enumerated states, or enforcing invariants across tags) and temporal properties (such as event orderings, delay ranges, periodicities, and temporal correlations). Each STL specification encodes a behavioral property that, when violated, triggers an anomaly alert. The verification process is multi-stage; initially, individual tag values are checked for compliance with their defined invariants (e.g., within physical safety bounds or protocol value constraints), followed by the evaluation of temporal patterns across tags (e.g., verifying event sequences such as a command issuance resulting in a corresponding state change within an expected time window). To ensure real-time performance effectively scales to large-scale CPS, the integrity verification algorithm utilizes efficient data structures specifically picked for the purpose. Specifically, match and threshold conditions are evaluated in constant time ($O(1)$) for each incoming observation using direct hash-table-based lookups. Temporal conditions (pre- and post-conditions) utilize time-indexed queues to efficiently manage the active time windows, ensuring that the processing complexity remains bounded by the number of active temporal dependencies and does not increase with operating time. This streaming time-window-based processing architecture ensures bounded resource usage (e.g., memory utilization) since incoming observations are processed in a streaming fashion without requiring any growing buffers/states that could cause cumulative errors or degradation over time, and thereby facilitates the robust operation of the integrity verification pipeline over long time periods. The satisfaction or violation of each STL formula is tracked in real-time, and violation events are logged together with contextual information such as which tags were involved and which property was violated (as discussed further in Section 3.6). This approach allows for fine-grained distinction between transient anomalies and persistent specification violations, and also facilitates further analysis by operators. Furthermore, this condition-based verification provides a unified mechanism to enforce cross-domain semantic consistency. For instance, a physical state change (e.g., a breaker opening observed via PMU measurements) can be correlated with a cyber command (e.g., a DNP3 operate command) and the corresponding network traffic patterns (e.g., the underlying packets from the relevant RTAC to the relay), with these diverse observations abstracted into tags and their relationships encoded as conditions. This abstraction enables the framework to efficiently track semantic behavior across the cyber–physical loop to detect anomalies that might be semantically consistent within a single domain (e.g., a valid relay open command) but violate cross-domain consistency (e.g., missing corresponding PMU voltage drop or abnormal network traffic).

3.6. Anomaly Localization

Each raw tag and computed tag maintains a provenance information as to which specific underlying communication observation was involved in the observed value of the

particular tag. Hence, considering the devices in the CPS and the observed communications between devices as a communication graph $\mathcal{G}$, the flagging of a condition violation directly indicates potential physical locations of the anomaly based on the edges (communication links) related to the constituent underlying tags in the condition check and the corresponding adjoining nodes. The DAG structure of raw and computed tags enables efficient retrieval of underlying raw tags corresponding to any flagged anomaly. Hence, anomaly scores are maintained for each node and edge and these scores are incremented each time a related anomaly is flagged. To enable the operator to rapidly see the most likely anomalous nodes/edges, the anomaly scores are normalized over the graph $\mathcal{G}$ and used for color coding in a graphical visualization (Figure 3). The algorithmic structure of this component is shown in Algorithm 4.

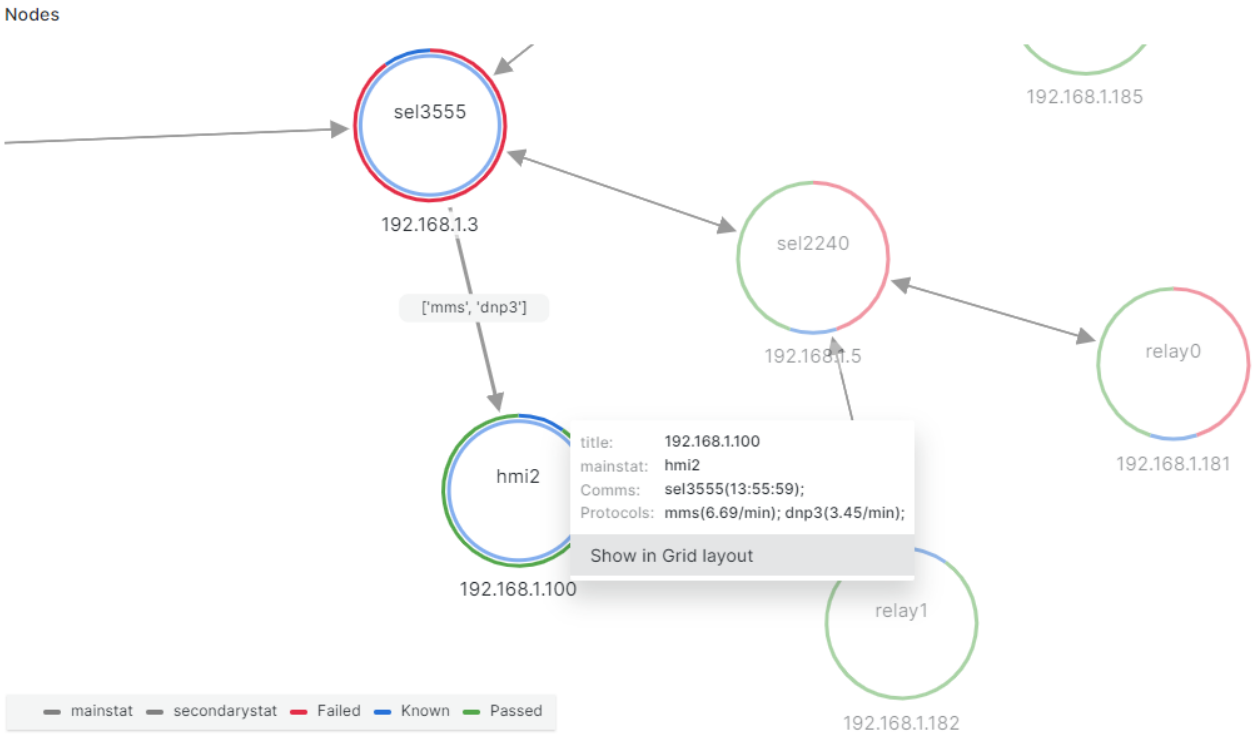

Figure 3. Sample screenshot of automatically generated nodes and edges graph in our Grafana-based GUI dashboard.

Algorithm 4 Algorithm for anomaly provenance scoring

1: Set anomaly scores to 0 $\forall$ nodes and edges in graph $\mathcal{G}$.
2: **for** M_i in anomaly queue $\mathcal{M}$ **do**
3: Look up all underlying raw tags R_j for M_i using DAG.
4: **for** R_j in effective raw tags **do**
5: Look up corresponding nodes and edge for R_j and increment their anomaly scores.
6: **end for**
7: **end for**
8: Normalize anomaly scores for nodes and edges so that $\sum_{n \in \mathcal{N}} a(n) = 1$ and $\sum_{e \in \mathcal{E}} a(e) = 1$ where $\mathcal{N}$ and $\mathcal{E}$ are the sets of nodes and edges in graph $\mathcal{G}$ and $a(.)$ is the anomaly score for a node/edge.

3.7. Visualization Dashboard

The user front-end of TRAPS is a dashboard GUI implemented using Grafana to visualize summaries of detected anomalies and overall semantically parsed observations

with a hierarchical interactive interface providing an easy-to-use top-level summary as a broad overview and on-demand interactive mechanisms to access additional details when desired. The visualization dashboard shows various elements of situational awareness and anomaly detection, such as observed nodes and communications (along with salient communication properties such as request/response timing), tag values and tag histories, and detection of anomalies in expected match/pre-/post-conditions and the provenance of detected anomalies. Also, a graph of the network architecture with a color-coded visualization of detected anomalies is embedded in the GUI (sample screenshot in Figure 3). The dashboard also provides plots of tag histories and tabular views of communications and tag values (screenshots omitted for brevity).

3.8. Implementation Architecture

The algorithmic structure of TRAPS offers multiple avenues for parallelization. Leveraging the modular structure of the TRAPS pipeline from the initial raw network ingest to the semantic processing and anomaly detection/localization components, the implementation architecture of the current prototype is shown in Figure 4.

At the network ingest front-end, the packet parser component uses a producer–consumer worker pool architecture where one thread reads from the source (either a PCAP or live network traffic) and dispatches packets to a set of worker threads via buffered channels. A consistent hashing strategy based on connection tuples (using source and destination identifiers) is used which guarantees that all packets belonging to the same flow are processed by the same worker. This flow affinity enables each worker thread to maintain an isolated parser state for protocols like DNP3 (which require fragment reassembly) without requiring complex locks or shared memory, significantly reducing contention. To ensure that the aggregated output from the threads is accurately time-ordered, a decoupled writer pattern is used where workers accumulate results into reusable batch buffers and send them to a dedicated output thread of the packet parser component. The output thread implements a resequencing buffer using a min-heap, enabling reconstruction of the original packet order from the asynchronously processed batches before writing the JSON stream output from the packet parser component.

The JSON intermediate representation (IR) is protocol-agnostic, enabling the unified processing of heterogeneous protocols (DNP3, Modbus, IEC61850, C37.118, etc.) by the following stages of the TRAPS pipeline. The JSON IR stream is ingested by the semantic tag processor, which is based on a DAG computation model in which both the incoming JSON messages and the tags emitted during computation are processed through a pool of worker threads based on dependency queues to allow efficient recursive computation of dependent tags while maintaining time-consistent ordering. The semantic tag processor outputs a time series of tags, which are then ingested by the anomaly monitor component, which utilizes a thread pool to process groups of STL property verifications over the incoming tag time series. The anomaly monitor implements match/threshold conditions using constant-time hash-table lookups and temporal conditions (pre-/post-conditions) using time-indexed queues whose memory usage is bounded by the number of active STL conditions rather than operating time. The anomaly monitor outputs a time series of violation events with timestamps and contextual metadata (tags involved, violated property). This time series is then used by the anomaly localizer, which runs as a separate thread, to track provenance information (node and edge anomaly scores) for anomaly indicators by referring to the DAG structure to recursively identify the underlying raw tags that contributed to any flagged computed tag or condition violation. The REST API server runs as a separate thread to handle on-demand requests from the dashboard (Grafana-based in a browser) and/or third-party systems by fetching information as needed from

the other components. The REST API server thread maintains local data caches to reduce queries to the other components.

While the initial prototype version of the pipeline was implemented primarily in Python 3 (with some hot spots in C++), an optimized implementation was then developed in Go (while keeping only the user-facing configuration API in Python). Although C++ can typically offer slightly higher single-threaded performance, the choice of Go over C/C++ was primarily based on the more lightweight multi-threading (green threads, i.e., goroutines) and more efficient inter-thread communication primitives (channels) in Go, which were found in tests of some of the more computationally intensive, but parallelizable, parts (packet payload parsing, tag computations) to yield around 10–15% higher throughput compared to C++.

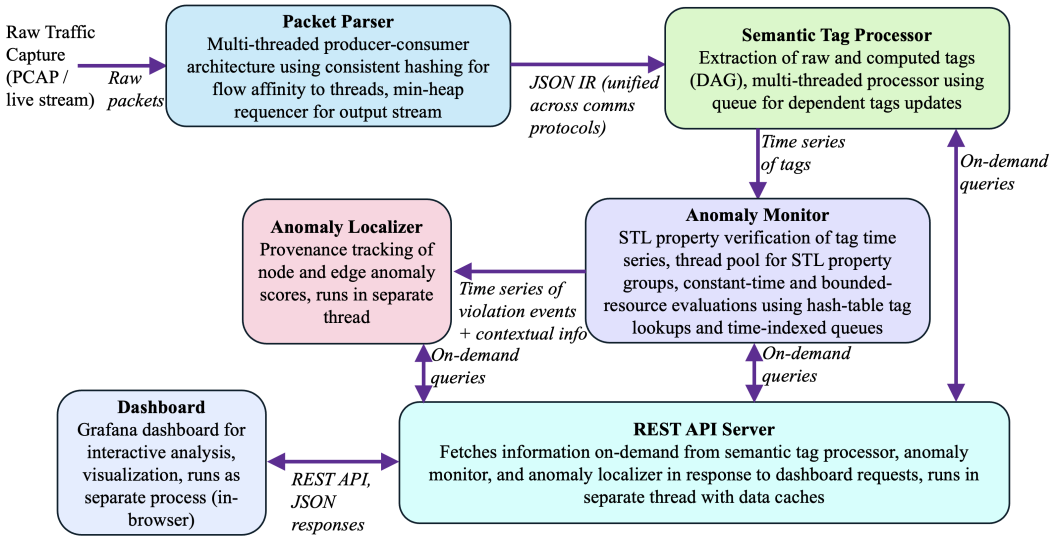

Figure 4. Implementation architecture of TRAPS with parallelization in multiple components to increase throughput and scalability.

4. Experimental Results

4.1. Experimental Setup and Behavioral Modeling

To evaluate the efficacy of the proposed framework, we developed a HIL testbed (Figure 5) with the architecture shown in Figure 6. The power grid simulator implemented using Matlab and Simulink running on a Linux server is interfaced with a network emulator based on the open-source CORE (Common Open Research Emulator) tool for building virtual networks. The virtual network emulator transparently routes traffic between both physical nodes and virtual nodes in the testbed as illustrated in Figure 7. The physical devices in the testbed are SEL (Schweitzer Engineering Laboratories) RTACs, which have been configured to have different roles including a Human–Machine Interface (HMI), data concentrator, and relay control logic devices. CORE allows creation of virtual nodes, each of which runs in a separate Linux namespace. Using this functionality, virtual devices were defined to simulate virtual IEDs like Relays, PMUs, and PDC, each with a corresponding script running in their Linux namespace to define their behavior and role. The MATLAB-based simulator for the power system dynamics and the virtual nodes interfaced to the network emulator communicate using FIFO (first-in-first-out) special files, via which the status of the relays and PMU measurements are transferred to/from the Matlab-based simulator. Note that all the components of the HITL simulator (including the physical SEL devices, virtual nodes running in Linux namespaces, and MATLAB-based power dynamics simulator) run online and interact in real-time. The interaction between the physical and virtual nodes is via the communication network (the bridging of the network

traffic between the physical and virtual nodes is handled by the CORE network emulator). As mentioned above, the interaction between the virtual nodes and the MATLAB-based power dynamics simulator is via FIFO files created in the Linux filesystem wherein each input value into the Matlab power dynamics simulator (e.g., relay status) is read in at the start of each simulation timestep from the corresponding FIFO file and each output value from the Matlab power dynamics simulator (e.g., voltage and current values to be used by the virtual PMUs) is written out to the corresponding FIFO file at the end of each simulation timestep. This architecture for integration of virtual and physical nodes and the power dynamics simulator presents a high-realism environment from the perspective of the physical devices and the TRAPS anomaly detection system, from whose viewpoint the network traffic communications/protocols and observed semantic behaviors are completely analogous to a fully real-world system with all physical devices and real power system interconnections. The HMI designed on the SEL-3555 can be accessed through the web interface and includes graphical functionalities designed to monitor values, control relays, and trigger attacks in the system for testing TRAPS. All physical devices and the simulation computer are connected using an L2/L3 Netgear switch. The switch's SPAN (Switched Port Analyzer) port is used to mirror all the traffic data in the network and send a copy to the monitoring machine (a Linux workstation), which then analyzes the traffic using the TRAPS framework described in this paper to detect anomalies.

Figure 5. Our HIL experimental testbed.

As a sample power system scenario for experimental testing, we defined a simple four-bus topology shown in Figure 8. The substation considered for monitoring in TRAPS is the set of components on the right side of Bus 1 in the figure. Area 1 is considered a remote area connected via a three-phase transmission line. There are five relays in the substation. Two loads are being fed (Loads 1 and 2) with breakers in a 1.5 breaker scheme. There are four PMUs (one for each bus). The relays, PMUs, and PDC are implemented as virtual nodes (Figure 9) while the RTACs and HMI are physical devices. These physical and virtual nodes communicate using a variety of protocols including DNP3, Modbus/TCP, IEEE C37.118, IEC61850 Goose and MMS, and SEL Fast Msg, as shown in Figure 9. To model the behavior of the overall CPS, a set of raw tags was defined as summarized in Table 2 in terms of which several match/pre-/post-/threshold conditions were listed as illustrated in Table 3 based on the defined roles and communications of the physical and

virtual roles in the system. Each tag filtering rule and condition are related to a specific part of the CPS design shown in Figure 9. For example, the tag at Index 4 keeps track of the Modbus Write Single Register Requests transmitted from SEL-2240 to Relay 1 and 2, which are holding registers with Integer type. As examples of conditions, observe that Condition 3 is a match requirement between values of Tags 1 and 2 (faithful forwarding of PMU measurements by a PDC). On the other hand, Condition 8 is a post-condition between Tags 12 and 4 (correct relaying of a HMI relay command by the RTAC). Several more tags and conditions could be defined analogously to capture characteristics such as power flows in different parts of the system (as computed tags), time-averaged or low pass filtered signals calculated from power measurements, temporal patterns of tag observations, etc.

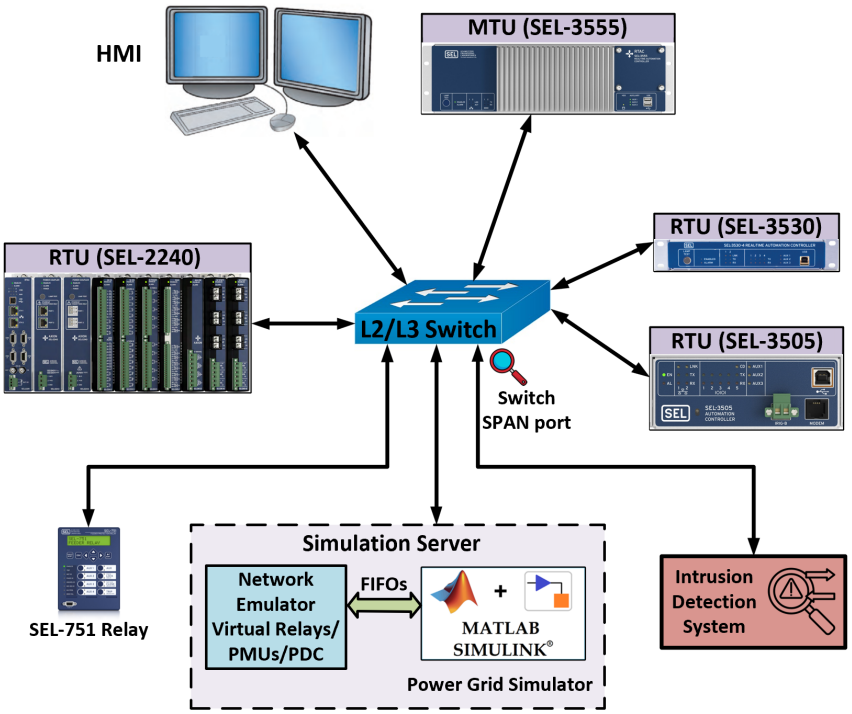

Figure 6. Architecture of HIL testbed.

Table 2. Sample raw tags for extracting semantic observations from network traffic in the HIL simulation setup. Register address details are omitted for brevity.

Index	Source	Destination	Protocol and Msg Type	Data Type
1	PMU_{1-4}	PDC	IEEE C37.118 Data Frame	Phasor, Analog and Digital data
2	PDC	SEL-3505	Modbus Read Holding Register Response	Holding Registers
3	SEL-3505	SEL-3555	SEL Fast Msg Unsolicited Write	Float Registers
4	SEL-2240	$Relay_{1-2}$	Modbus Write Single Register Request	Int Holding Registers
5	SEL-2240	$Relay_{1-2}$	Modbus Read Holding Register Request	-
6	$Relay_{1-2}$	SEL-2240	Modbus Write Single Register Response	Int Holding Registers
7	$Relay_{1-2}$	SEL-2240	Modbus Read Holding Register Response	Int and Float Holding Registers
8	SEL-3530	$Relay_{3-5}$	Modbus Write Single Register Request	Int Holding Registers
9	SEL-3530	$Relay_{3-5}$	Modbus Read Holding Register Request	-
10	$Relay_{3-5}$	SEL-3530	Modbus Write Single Register Response	Int Holding Registers
11	$Relay_{3-5}$	SEL-3530	Modbus Read Holding Register Response	Int and Float Holding Registers
12	SEL-3555	SEL-2240	DNP3 Operate Request	Group 41 all variations Int Analog Output
13	SEL-3555	SEL-2240	DNP3 Read Request	Group 60 all variations Int Analog Input
14	SEL-2240	SEL-3555	DNP3 Operate Response	Group 41 all variations Int Analog Output
15	SEL-2240	SEL-3555	DNP3 Read Response	Group 30 all variations Int and Float Analog Inputs
16	SEL-3555	SEL-3530	SEL Fast Msg Unsolicited Write	Int Registers
17	SEL-3530	SEL-3555	SEL Fast Msg Unsolicited Write	Int and Float Registers
18	Any Device	$Relay_{1-2}$	Modbus Write Single Register Request	Int Holding Registers
19	Any Device	$Relay_{3-5}$	Modbus Write Single Register Request	Int Holding Registers
20	SEL-2240	SEL-751	IEC61850 GOOSE Multicast	Float Analog Registers
21	SEL-2240	SEL-751	IEC61850 MMS Report Unbuffered	Float Analog Registers
22	SEL-751	SEL-2240	IEC61850 GOOSE Multicast	Float Analog Registers
23	SEL-751	SEL-2240	IEC61850 MMS Report Unbuffered	Float Analog Registers

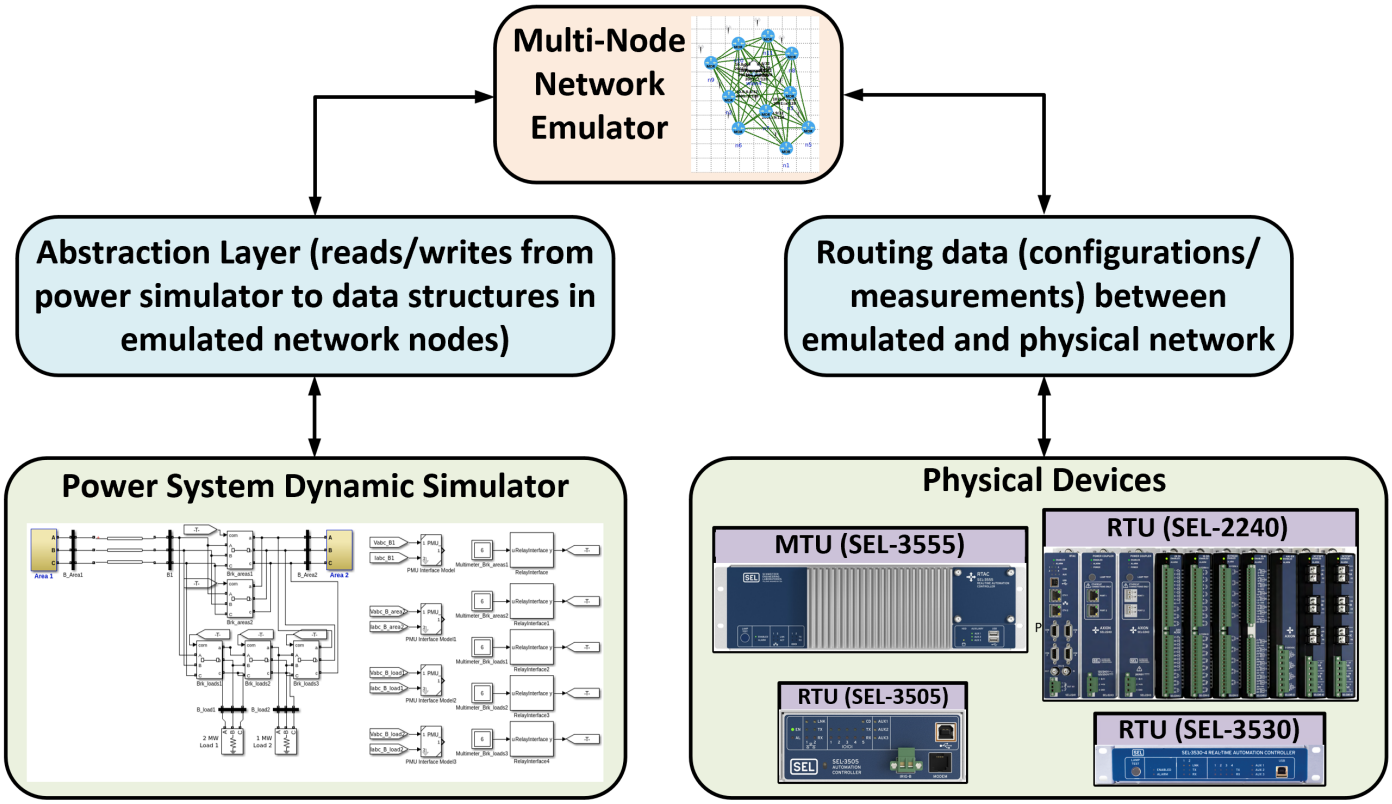

Figure 7. Architecture of communications between physical devices (SEL), emulated devices, and the power system dynamic simulator. The power system simulator uses an abstraction layer to interact with emulated nodes (including Relays, PMUs, and PDC). On the other hand, emulated nodes interact directly with physical nodes in an L2/L3 network.

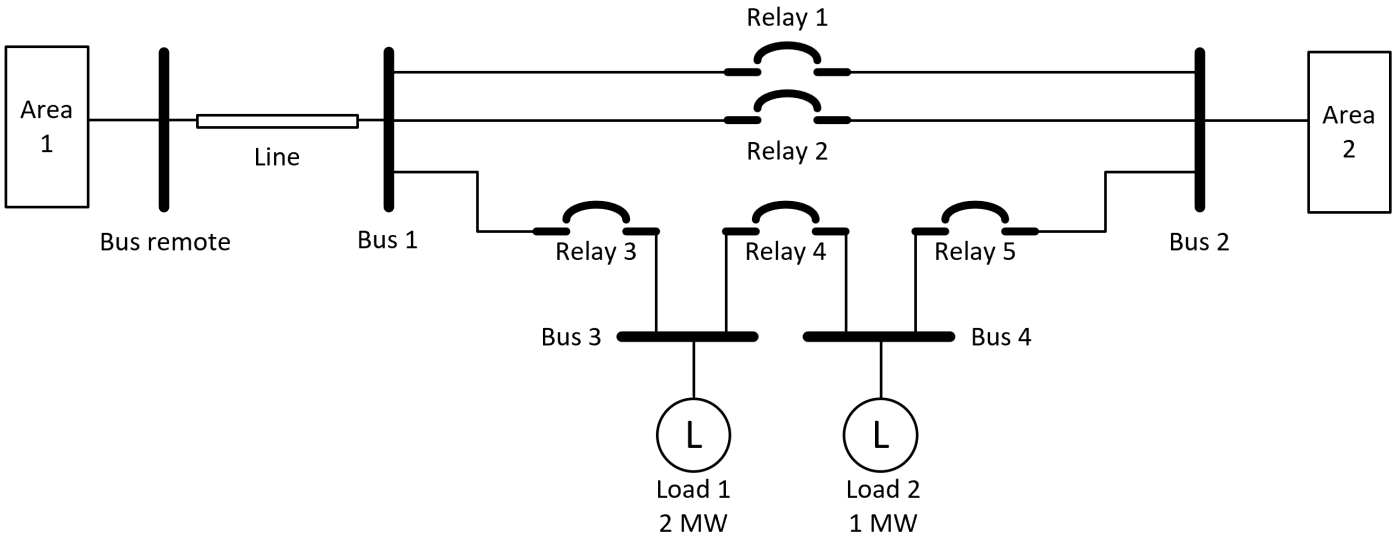

Figure 8. Simple power system topology defined for HIL testing.

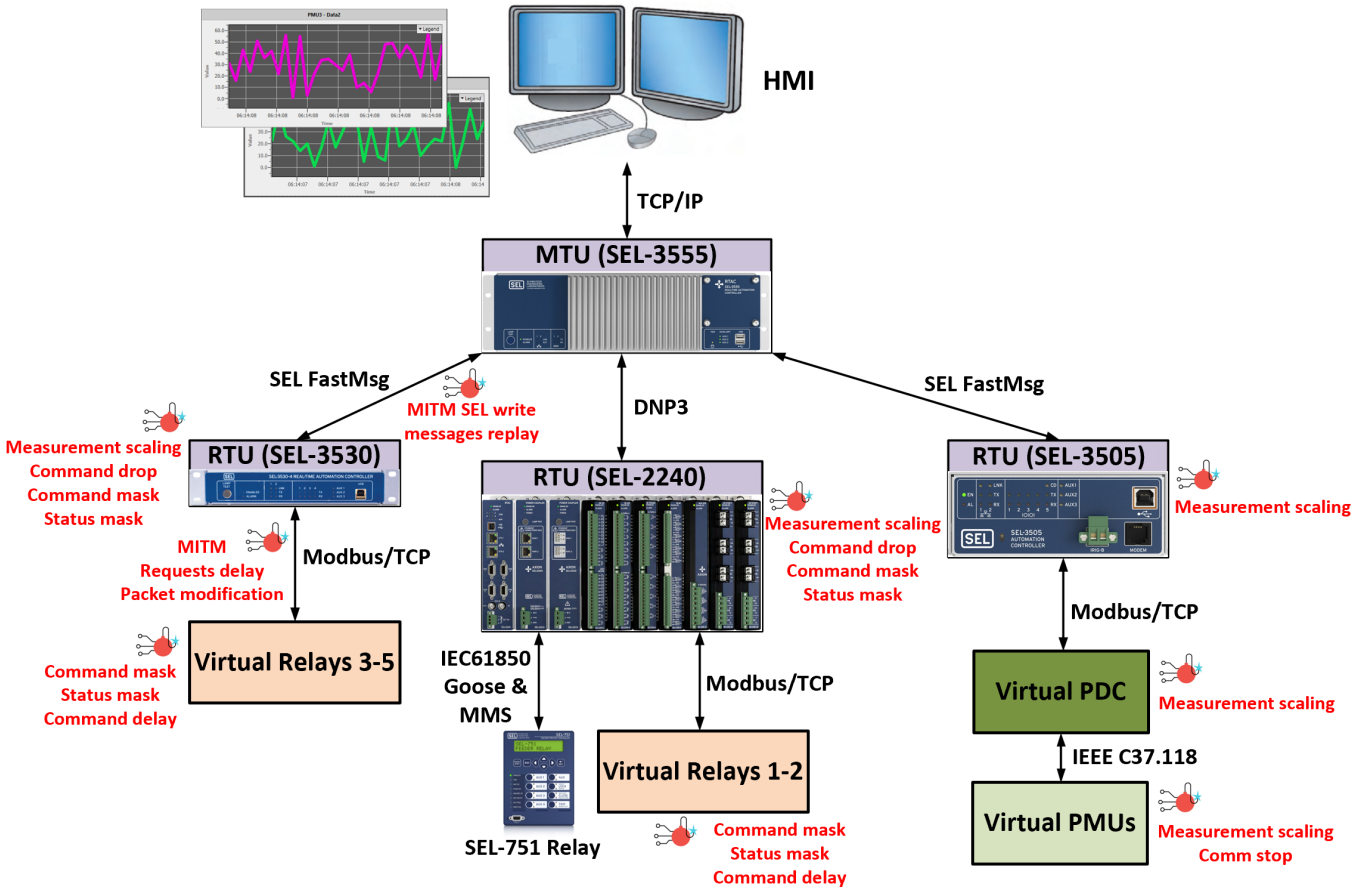

Figure 9. Physical and virtual nodes and roles in HIL testbed. SEL RTACs are used along with simulated Relays, PMUs, and PDC communicating with a variety of different protocols. Various attacks that were tested are shown in red between devices.

Table 3. Sample conditions to model the CPS temporal behavior. The defined thresholds are based on the nominal communication characteristics and power system dynamics. For the threshold conditions on value and match conditions, the defined threshold shows the allowed deviation from nominal values. For pre-/post-conditions, the defined threshold shows the allowed time window and deviation from nominal values, respectively.

Index	Condition Type	Tag 1 ID	Tag 2 ID	Threshold
1	Threshold condition on value	1	-	1%
2	Threshold condition on time	1	-	0.05 s–0.15 s
3	Match condition	1	2	1%
4	Match condition	2	3	1%
5	Post-condition	4, 8	7, 11	1.1 s, 1%
6	Match condition	7	15	1%
7	Post-condition	12	15	6.0 s, 1%
8	Post-condition	12	4	0.3 s, 1%
9	Match condition	11	17	1%
10	Post-condition	16	17	1.1 s, 1%
11	Post-condition	16	8	0.3 s, 1%
12	Threshold condition on time	9	-	0.98 s–1.02 s
13	Threshold condition on time	17	-	0.95 s–1.05 s
14	Pre-condition	18	12	0.3 s, 1%
15	Pre-condition	19	16	0.3 s, 1%
16	Match condition	22, 23	15	1%
17	Post-condition	12	20, 21	1.0 s, 1%

4.2. Evaluation of Attack Detection

4.2.1. Attack Scenarios and Detection Analyses

To test the attack detection performance of the proposed framework, a wide range of attacks were implemented in our HIL setup as shown in red in Figure 9. Our experiments mainly focus on adversarial manipulations of the grid's devices' logic and behavior, which are among the most potent and stealthy attacks that could be done via supply chain, firmware tampering, and advanced persistent threat (APT) lateral movement attacks. The considered attacks summarized in Table 4 cover a vast range of real-world attacks. The attacks on the PMUs, PDC, RTACs, and relays are simulated as malicious firmware delivered via supply chain attacks. On the other hand, the MITM attacks on Modbus/TCP and the SEL Fast Msg protocols are performed by injecting an external device in the path of communication between devices. An ODROID-XU4 was used as the MITM device and an MITM interception was implemented using the Python Scapy library and Scapy Modbus package (for intercepting and modifying Modbus requests and response packets). MITM attacks could alternatively be performed using ARP spoofing, which results in ARP cache poisoning on the victim devices and enables the attacker to intercept communications. In our MITM implementation, an intruder device is physically placed in the communication path, therefore removing the need for ARP spoofing and making the attack even more effective and stealthy.

We captured 5 min of network communication traffic from the HIL simulator for the normal mode and when triggering each of the attacks listed in Table 4. We fed each of the captured traffic data to our proposed TRAPS system and observed that all the considered attack scenarios are reliably detected after transitioning from the normal mode to the attack mode since at least one of the conditions in Table 3 is violated under each scenario. For example, attack scenario 9 (command dropping attack on SEL-2240) results in violation of Post-Condition 8 between Tags 12 and 4, which states that the value of Tag 12 (DNP3 Operate Requests from SEL-3555 to SEL-2240) should match with Tag 4 (Modbus Write Single Register Request from SEL-2240 to Relay 1–2) within a short time window. As another example, attack scenario 15, which is an FCI attack to Relays 3 and 4 in the form of sending false open/close commands to relays from SEL-3530 (or any other intruder devices) is depicted in Figure 10. This attack is detected using Pre-Condition 15 since it requires that if a Modbus Write Single Register Request is being sent to Relays 3–5 in Figure 9, then within a short time window before that, a corresponding command should have been sent from SEL-3555 to SEL-3530 (i.e., an authorized command by the HMI) requesting the relays to be opened or closed. This condition is not satisfied when the attacker sends false messages, leading to detection of an anomaly.

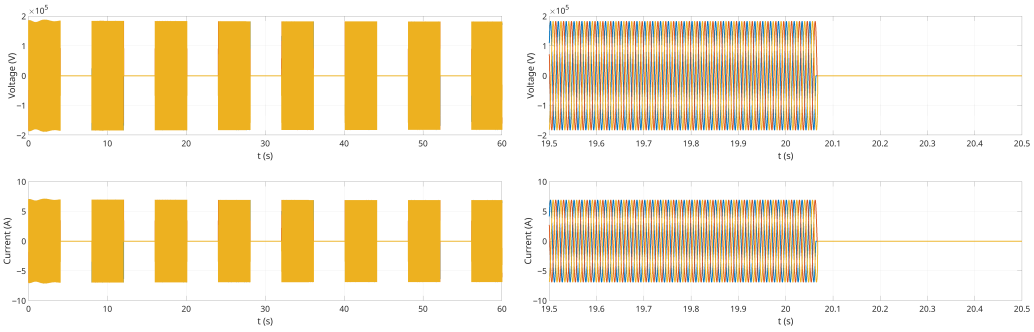

Figure 10. Voltage (phase to ground) and current plots for Load 1 in Figure 8 (as measured at Bus 3) when the attacker sends commands to Relays 3 and 4 in Figure 9 to cycle them on and off with a time interval of 4 s between commands. This has the effect of cycling power to Load 1. Voltage and current plots for other buses are omitted for brevity. The right side of the figure is a zoomed-in view of the plot in the left side of the figure.

Table 4. Various attack scenarios considered in HIL testing.

Index	Scenario
1	FDI attack in virtual PMUs through measurement scaling
2	DoS attack in virtual PMUs through communication disruption
3	FDI attack on the PDC in the form of measurements scaling
4	FDI attack in SEL-3505 using measurements scaling
5	FDI and FCI attacks in Virtual relays through commands and status masking
6	DoS attack in virtual relays in the form of command delaying
7	FDI attack on SEL-2240 in the form of measurement scaling
8	FDI and FCI attacks on SEL-2240 through status and command masking
9	DoS attack on SEL-2240 by dropping commands
10	FDI attack on SEL-3530 through measurement scaling
11	FDI and FCI attacks on SEL-3530 through status and command masking
12	DoS attack on SEL-3530 by dropping commands
13	MITM attack on Modbus/TCP protocol between the virtual relays and SEL-3530, which includes DoS in the form of Read Holding Registers request delaying and FDI through packet modification
14	MITM attack on SEL Fast Msg protocol between the SEL-3530 and SEL-3555, which includes replay attack on SEL Unsolicited Write messages
15	FCI attack on Relay3-5 by sending false commands from SEL-3530 (or any other intruder devices in the network) to open/close the relay without the request of HMI

As another illustration of anomaly detection by the proposed framework, the timing plots under attack scenario 13, which is an MITM attack between SEL-3530 and Relays 3–5, are shown in Figure 11. The attacker randomly adds delays in the communication of Modbus Read Holding Registers Request messages sent from SEL-3530 to Relays 3–5 in Figure 9. As seen in Figure 11, the time intervals between subsequent Modbus request packets under the MITM attack have an anomalous temporal pattern triggering the raising of an alert based on condition 12 in Table 3.

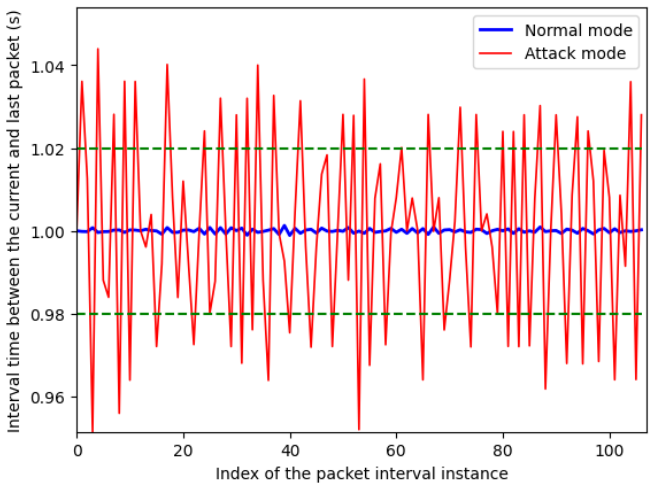

Figure 11. Plot of interval times between subsequent Modbus Read Holding Register Requests from SEL-3530 to Relay 3 in Figure 9 in normal mode and MITM attack mode. The interval times have a relatively large variance in the attack case, enabling detection of the anomaly/attack using the defined threshold conditions. The horizontal dashed lines show the thresholds defined on the value of time intervals in the conditions of Table 3.

Another example attack is illustrated in Figure 12, which shows the timing plots under attack scenario 14 (MITM replay attack between SEL-3530 and SEL-3555). In this case, the attacker retransmits some SEL Fast Msg packets from SEL-3530 to SEL-3555. The figure shows time intervals between subsequent SEL Unsolicited Write messages during normal and attack modes. These timings are detected as anomalous under the attack mode based on Condition 13 of Table 3.

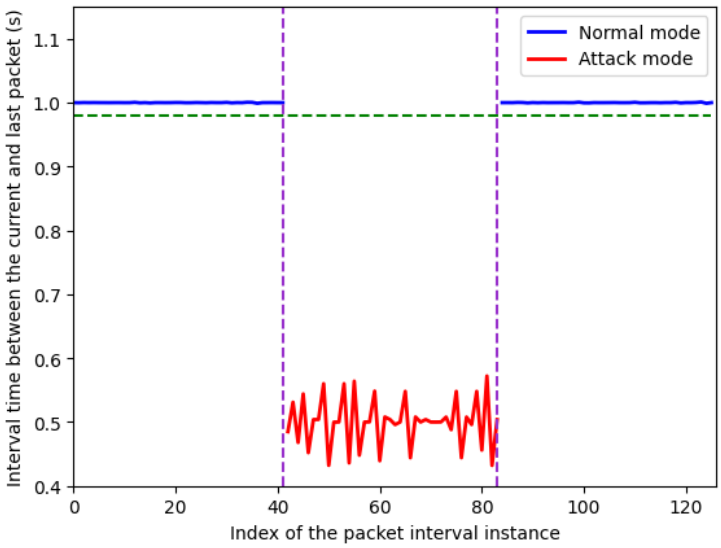

Figure 12. Plot of interval times between subsequent SEL Fast Msg Unsolicited Write messages from SEL-3530 to SEL-3555 in Figure 9 in normal mode and MITM replay attack mode. In the case of attack mode (during time interval demarcated with purple lines in the plot), the delay value drops lower than the defined threshold (green line) in the conditions of Table 3, resulting in detection of an anomaly.

4.2.2. Anomaly Detection Performance Comparison Against Several Baselines

Now, to evaluate TRAPS performance against several prior anomaly detection approaches, we consider multiple representative baselines. Since we are evaluating end-to-end anomaly detection from raw PCAPs to anomaly alerts, to obtain a range of baselines, we consider a standard pipeline of (i) feature extraction from raw PCAP and (ii) anomaly detection on the resulting feature vectors to classify the observed traffic as anomalous or benign based on the extracted feature vectors.

We consider two widely used feature extraction engines: Zeek [84] and nfstream [85,86]. In the case of Zeek (formerly known as Bro), which is an industry-standard tool for network traffic analysis, we used a configuration in which it extracts a multi-layered feature vector that spans both statistical and payload inspection-based features in a unified log. Specifically, this Zeek-based feature vector log includes raw packet-level features (e.g., packet lengths, source/destination IPs and ports, TCP flags, protocol), enriched flow-level connection context (e.g., flow duration, packets and bytes in both directions, timing statistics), deep-packet semantic values for supported OT communications protocols (using ICSNPP extension packages for Zeek) including Modbus, DNP3, and C37.118 (e.g., Modbus function codes and register references, DNP3 application control and object data, C37.118 phasor values), and behavioral statistical features like payload entropy and timing-based features. As an alternative to Zeek, TShark [87] could also be used, which is also industry-standard and is the command-line version of the almost universally used Wireshark network protocol analyzer and enables extraction of payload inspection-based features using protocol-specific dissectors. However, Zeek provides two advantages motivating its choice in this baseline instead of TShark. The first advantage of Zeek over TShark is greater flexibility and

versatility of feature extraction; while TShark by itself only provides largely stateless, field-by-field extraction of raw packet data (limited to the specific features available from the command-line options in Tshark), Zeek provides a flexible stateful parsing engine that enables enriching each record with cumulative flow-level context by tracking connection state, behavioral entropy metrics, and deep real-time protocol aggregations such as C37.118 phasor and analog value statistics. The second advantage of Zeek over TShark is significantly higher speed (as discussed further later as part of Section 4.2.4). In the case of nfstream, which is a popular and actively maintained feature extraction framework, we used its "early" and "post-mortem" statistical flow feature analysis capabilities to generate detailed flow-level features, including bidirectional packet-level statistics, flow duration features, and statistical distributions (minimum, maximum, mean, standard deviation) of packet sizes and inter-packet arrival times. However, nfstream does not perform deep payload parsing and therefore cannot directly represent most types of protocol semantics.

For the anomaly detection algorithms, we use the state-of-the-art PyOD 2 library [88,89], which provides a unified framework with optimized implementations of several anomaly/outlier detection algorithms ranging from classical algorithms such as LOF (Local Outlier Factor) [90] and OCSVM (One-Class Support Vector Machine) [91] to newer variants such as CBLOF (Clustering-Based Local Outlier Factor) [92] and AE1SVM (Autoencoder-Based One-Class Support Vector Machine) [93], as well as several recent algorithms such as LUNAR (Learnable Unified Neighborhood-based Anomaly Ranking) [94], ECOD (Empirical Cumulative Distribution-Based Outlier Detection) [95], DeepSVDD (Deep Support Vector Data Description) [96], and Deep Isolation Forest (DIF) [97].

The anomaly detection performance for each combination of the two feature extractors (Zeek, nfstream) and each of six anomaly detection algorithms (CBLOF, AE1SVM, LUNAR, ECOD, DeepSVDD, DIF) is shown in Table 5. For each of these anomaly detection algorithms, we used the default parameters from PyOD 2. The anomaly detection performance metrics of TRAPS are shown in the last row of Table 5. All numbers in the table are based on averages over five runs to account for small variations of performance in successive runs for some methods. In all cases, only data from normal operation is used for training the anomaly detection algorithms. For testing, we use separate datasets of normal operation and of several attack scenarios (Table 4). Also rather than overly coarse "one label per PCAP" classification and to reflect streaming detection requirements, we consider 5 s time windows to obtain a sufficient number of samples for the statistical evaluation of metrics including accuracy, precision, recall, F1, and Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) curve.

There are two key observations that can be seen from Table 5. Firstly, it is seen that anomaly detectors when using the Zeek-based feature extractor perform better than with nfstream-based feature extraction. This is expected since nfstream provides purely statistical, flow-based features. Since the anomalies being addressed here are primarily semantic anomalies that manipulate CPS/OT signals without affecting traffic statistics, it makes sense that nfstream features cannot capture much of the underlying meaning needed for reliable detection. On the other hand, Zeek-extracted features include payload inspection-based features which capture the underlying OT semantics and therefore enable reasonable accuracies. Secondly, while when using the Zeek-based feature extraction, there is some variation among the different algorithms and the best scores among these do give reasonably good anomaly detection, TRAPS provides significant improvements with near-perfect end-to-end performance by capturing the underlying semantics structures and context dependence. In particular, precision is 1.0 with TRAPS, indicating that the TRAPS approach of analyzing explicit semantic behavioral properties tends to yield very low false positives under normal operation. More broadly, TRAPS' approach of validating

semantic sequences of events across multiple devices in a CPS, thereby essentially auditing the entire "cyber–physical loop", enables accurate flagging of deviations from expected behaviors. However, it is to be noted that in general, in real-world deployments, a defense-in-depth approach with multi-layered defenses would be the most suitable for a robust security solution. These different monitoring and anomaly detection approaches are, in fact, complementary and synergistic. For example, while TRAPS enables detection of variations from subtle semantic behaviors across different devices in a CPS, general-purpose network monitoring can facilitate robust detection of more general patterns of malicious behaviors (port scans, denial of service attacks, lateral movements, etc.), and signature-based detectors can enable rapid detection of markers of known attack types. In general, these different approaches should ideally be combined together to obtain a robust solution in real-world CPS.

Table 5. End-to-end anomaly detection performance comparison (raw PCAP → alerts) of TRAPS versus several baseline pipelines that combine feature extraction (Zeek [84] with OT protocol packages such as ICSNPP, and nfstream [85] flow-statistical features) with multiple anomaly detectors from PyOD 2 [89] (CBLOF, AE1SVM, LUNAR, ECOD, DeepSVDD, DIF). Models are trained only on normal-operation data and tested on separate datasets of normal operation and of several attack scenarios (Table 4). Metrics (accuracy, precision, recall, F1, ROC AUC) are averaged over five runs to account for the small variability across runs of some anomaly detection methods. The highest score in each column is shown in bold and the second-highest score is underlined.

Method	Acc.	Prec.	Rec.	F1	ROC AUC
Zeek + CBLOF	0.731	0.657	<u>0.974</u>	0.784	0.891
Zeek + AE1SVM	0.805	0.730	0.970	0.833	0.895
Zeek + LUNAR	<u>0.899</u>	<u>0.910</u>	0.886	<u>0.898</u>	<u>0.904</u>
Zeek + ECOD	0.516	0.565	0.144	0.230	0.658
Zeek + DeepSVDD	0.734	0.731	0.745	0.736	0.790
Zeek + DIF	0.503	0.498	0.129	0.203	0.549
nfstream + CBLOF	0.547	0.527	0.931	0.673	0.377
nfstream + AE1SVM	0.603	0.568	0.862	0.685	0.405
nfstream + LUNAR	0.521	0.513	0.697	0.590	0.428
nfstream + ECOD	0.540	0.615	0.213	0.317	0.527
nfstream + DeepSVDD	0.597	0.564	0.848	0.678	0.427
nfstream + DIF	0.452	0.408	0.203	0.270	0.435
TRAPS	**0.992**	**1.0**	**0.984**	**0.992**	**0.992**

4.2.3. Threshold Selections for Anomaly Detection

The thresholds for value-based and timing-based checks in TRAPS (e.g., Table 3) are specified directly as parameters in the STL conditions monitored by the system (including match conditions, threshold conditions, and temporal pre-/post-conditions). In practice, these thresholds are chosen based on typical system operation and network characteristics as captured by statistical characterization under normal operation. For example, for timing-based conditions that enforce bounds on inter-message intervals or end-to-end response delays; for instance, thresholds can be set using the empirical distribution of the measured intervals during normal operation to ensure no/low false positives under normal conditions by selecting a bound based on the tail of the observed distribution and incorporating a safety margin to tolerate typical levels of jitter. This approach for picking thresholds is standard in anomaly monitoring because, as with any detection methodology, there is an inherent trade-off between tighter thresholds and false positives: tighter thresholds can increase sensitivity to small deviations but may also increase false positives when benign variability (noise, jitter, transient variations) causes occasional apparent threshold

violations. In fact, TRAPS is less susceptible to this trade-off than approaches that monitor low-level packet features or purely statistical deviations, because the monitored conditions encode higher-level semantic behaviors that are robustly satisfied in normal operation in the absence of genuine anomalies and are meaningfully violated only when the system behavior is inconsistent with the intended cyber–physical loop. For instance, if a command issued from the HMI to an RTAC is expected to be forwarded to a relay (and subsequently reflected in the relay's state and/or corresponding measurements), then failure of this semantic sequence indicates an actionable anomaly regardless of whether the root cause is an attack, malfunction, or misconfiguration; thus, the monitoring objective is aligned with meaningful semantic deviations rather than low-level patterns.

To explicitly study sensitivity to threshold selection and the effects of unexpected noise, we consider the example scenario illustrated in Figure 11 where an MITM attack is inserted between SEL-3530 and Relays 3–5 and we intentionally inject additional synthetic timing variations between the SEL-3530 and the Relays 3–5 under both the normal case and when the MITM attack (illustrated in Figure 11 when the additional timing variations were not included) was launched. In this example scenario, focusing on just this single timing threshold condition to study the sensitivity to noise and threshold selection, we evaluate precision and recall under a range of candidate threshold values. The first row of Figure 13 shows the precision and recall as the threshold is varied for the case when the additional synthetic timing noise is not included. Under this same case, the second row shows the corresponding ROC and precision–recall curves. The third and fourth rows show the analogous plots when synthetic timing variations are injected into the measured interval times to emulate unexpected network noise/jitter. Specifically, we add an i.i.d. Gaussian perturbation with mean 5 ms and standard deviation 5 ms to the inter-message intervals (i.e., interval times between subsequent Modbus Read Holding Register Requests from SEL-3530 to each of Relays 3–5). Without the additional timing noise, the accuracy, precision, recall, and F1 score (measured based on anomaly detections over 5 s time windows) are all 1.0 for the threshold 0.98–1.02 (i.e., threshold on diff = 0.02) shown in Table 3. With injected timing noise, these metrics become the following(averaged over five runs to account for variability introduced when adding random noise): accuracy $= 0.994$, precision $= 0.992$, recall $= 0.996$, and F1 $= 0.994$, with a small precision reduction attributable to a small number of false positives induced by the added timing variability. However, it is seen that the ROC curve remains essentially ideal (ROC AUC $= 1.0$), indicating that a small increase in the threshold can eliminate these false positives and restore accuracy of 1.0; such a threshold adjustment is naturally guided by the empirical distribution observed under normal conditions. Note that threshold selection in this process relies only on measurements from normal operation and does not require any data from attack/anomalous conditions, consistent with TRAPS' specification-based monitoring approach. Also, note that the precision and recall curves as functions of threshold are relatively flat around the threshold setting, indicating that the anomaly detection is not very sensitive to the exact threshold value.

4.2.4. Scalability and Throughput

While the above examples illustrate some types of attacks detected by our proposed TRAPS framework, an important point to note is that TRAPS is not designed specifically to detect these (or any other) particular attacks. Rather, TRAPS is intended as a flexible behavioral modeling and integrity verification framework in which the semantic temporal characteristics or behavioral specifications of the CPS based on device control logic, CPS control flow designs, physics, communication configuration, etc., can be defined to any desired level of detail and thereafter *automatically* monitored in real-time by TRAPS and any deviations flagged as anomalies. Since anomalies are flagged as deviations from expected

semantic behavioral properties, an anomaly detection is accompanied by the indication of the particular semantic properties that were violated, thereby providing interpretability and anomaly localization in terms of specific nodes/edges that were involved in the flagging of the particular anomaly. Furthermore, since TRAPS validates the real-time observations against semantic properties, it was noted that the false positive rate was practically zero, i.e., under normal behavior of the devices, the expected semantic properties are indeed satisfied resulting in no anomalies being flagged. Also, since TRAPS continuously audits the entire cyber–physical closed loop for the configured semantic properties, multiple simultaneous adversarial modifications if present are each individually detected and localized to the corresponding anomalous devices or communication links. To evaluate this, combinations of attack examples from Table 4 were tested and it was seen that even when multiple attacks were simultaneously deployed, the anomalies and the separate underlying condition violations were accurately detected. However, it is to be noted that as with any anomaly detection system, if the attacker is allowed to coordinate an arbitrary number of attacks simultaneously so as to mask each other's presence, then detection of such an attack would not be possible. For example, as noted in Section 3.1, an adversary with unlimited power (e.g., changing firmware of a relay to cause a delayed response while simultaneously attacking all PMUs to show the expected voltage and current signals if the relay had operated correctly) would be able to elude detection by ensuring that attacks on different parts of a system mask each other's effects so as to present a consistent view to the anomaly monitor. However, as noted in Section 3.1, the typical attack vectors and entry points of adversaries in a CPS such as the smart grid (also taking into account the heterogeneity of devices and protocols) tend to make it difficult for an attacker to feasibly affect all observable network communications. These observations are as with any anomaly detection approach since such an adversary of unlimited capacity as to corrupt all observations can definitely elude detection.

While the experiments presented above use 5 min traffic captures for focused analysis of each attack scenario, the TRAPS framework has also been validated in longer-duration experiments, including hours-long runs on our HIL testbed as well as day-long tests in real-world operational settings during exercises sponsored by the US Department of Energy. In these longer experiments, we have observed consistent performance of the system without unbounded resource requirements growth, cumulative errors, or any degradation in system behavior due to network conditions/jitter. This stability is primarily due to the streaming-based architecture of the framework, where observations are processed through the semantic extraction and anomaly detection pipeline with the state maintained only for the sliding time windows required by behavioral specifications (typically on the order of seconds, as shown in Table 3). This streaming architecture results in bounded memory usage and also implies that there is no accumulation of errors over time since packets are processed in a streaming fashion and the system does not maintain growing buffers or state that could degrade over extended operation.

Another key part of enabling a flexible framework for semantic mapping and real-time integrity verification is scalability and computational tractability. While OT communications traffic is typically lower-bandwidth than general-purpose IT traffic, performance remains important for scalability, particularly in larger substations and during transient bursts. This is true even though very high sustained OT traffic rates are rare, and even when network links support 1 Gbps, the actual OT communications traffic in substation-like environments rarely exceeds a few hundred Mbps. In addition, OT communications are often sparse in the sense that packets relevant to semantic tag parsing and STL condition checking constitute only a fraction of the overall traffic, which further enables efficient parallelization by focusing deep parsing and verification effort on semantically relevant message types.

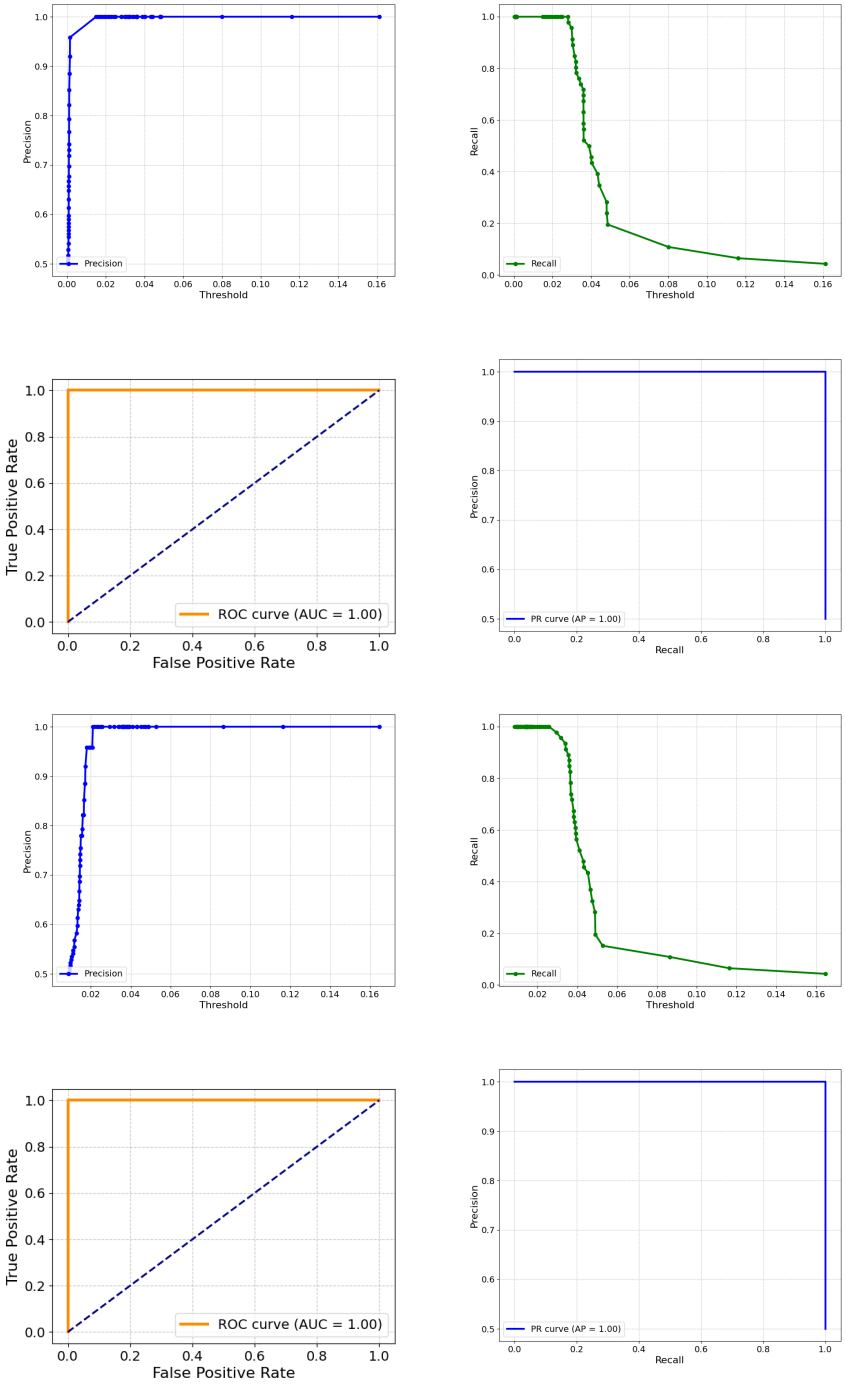

Figure 13. Analysis of the effects of noise on threshold selection focusing on a representative timing threshold STL condition in an MITM attack scenario on the SEL-3530 → Relays 3–5 communication path. The timing threshold from Table 3 is set to 0.98–1.02 in this case, i.e., the threshold on difference from the nominal value of 1.0 is set to 0.02. Rows 1–2: baseline case (no additional injected timing noise) showing precision/recall versus range of thresholds (i.e., thresholds in terms of diff with nominal threshold based on Table 3 set to 0.02) and the corresponding ROC and precision–recall curves. Rows 3–4: analogous plots with synthetic timing variations added to the inter-message intervals (additive Gaussian noise with mean 5 ms and standard deviation 5 ms). Note that due to the randomness of injected additional synthetic timing noise, there is a small variability of the plots across runs; averaging over 5 runs yields average precision of 0.992 and recall of 0.996. The plots illustrate the trade-off between threshold tightness and false positives and show that a small threshold increase (guided by the normal-operation distribution tail) will recover an accuracy of 1.0.

As noted in Section 3.8, our initial implementation was primarily Python-based, while our current updated implementation is in the Go language and is heavily parallelized using the lightweight Go goroutine and channel functionalities. Our initial Python-based implementation, which already incorporated parallelization (Docker containers per protocol and multi-threaded tag processing and anomaly detection), achieved approximately 25 Mbps sustained throughput for processing raw network traffic, with per-item computational processing times of around 1–2 µs per semantic tag and 0.5–1 µs per STL condition check. The optimized Go implementation (with the architecture discussed in Section 3.8) provides significantly higher performance, reaching around 250 Mbps throughput, with per-item computational processing times of around 0.1 µs per tag and 0.1–0.2 µs per STL condition check. Additionally, the end-to-end anomaly detection latency (from arrival of raw packets violating an STL condition to flagging of the anomaly) in streaming mode is typically 2–3 ms, and due to the parallelized pipeline architecture, this latency remains low even under high traffic rates and concurrent dashboard interactions. The primary bottleneck in the overall pipeline is the packet parsing for deep payload inspection, which is intrinsic to TRAPS' end-to-end semantic monitoring approach. Real-time dissection of protocol payloads is inherently a computationally demanding task. Our initial Python implementation utilized parsers built using scapy [79], pyshark [80], and the Hammer [81] library (which provides a parser combinator interface in which grammars can be written as inline domain-specific languages). The updated Go implementation system instead uses custom parsers generated using the Kaitai framework [98], which is based on a declarative language to describe the binary layout of the data structures and a code generator that transforms the declarative specifications into highly efficient implementation code, yielding significant throughput gains. To compare the attained throughput of the packet parsing implementation in TRAPS against industry-standard tools, we measured processing times for a sample PCAP of 32 MB size with contained TCP, UDP, and DHCP traffic with a mix of OT communications traffic including DNP3, C37, and GOOSE. For this PCAP, we measured feature extraction processing times using Zeek [84] and TShark [87], with two representative configurations for each: extraction of basic statistical features (e.g., packet lengths, protocol identifier, source/destination IPs and ports, TCP flags), extraction of payload inspection-based features (e.g., protocol-specific semantic fields for OT communications traffic such as Modbus function codes and register references, DNP3 application control and object data, synchrophasor phasor values) in addition to statistical features. Under these two configurations, feature extraction from the PCAP using Zeek used 1.31 s and 2.51 s, respectively, while the feature extraction using TShark used 4.35 s and 4.46 s, respectively. In comparison, the TRAPS Kaitai-based multi-threaded parser, which extracts both statistical features and deep payload features at a higher level of detail than the Zeek- and TShark-based parsers, took 1.04 s, providing significantly higher processing speed than these industry-standard tools. As another baseline for comparison, we also measured the processing time for feature extraction using nfstream [85,86], which is implemented with an optimized C-based packet parsing engine built on libpcap and computes a wide range of flow-based statistical features but does not perform deep payload parsing. For this PCAP, the feature extraction using nfstream required 0.79 s. It is notable that the feature extraction processing time using TRAPS approaches close to nfstream while providing deep payload inspection and significantly exceeds Zeek and TShark. All the timing measurements above were averaged over five runs (with two additional warm-up runs prior to measurement to reduce variability by ensuring binaries and pcap data were in cache). The experiments were performed on a laptop with an Intel Core i9 2.2 GHz (14th gen) processor and 32 GB memory running Ubuntu 25.10.

5. Conclusions

A novel anomaly monitoring and integrity verification framework for CPS based on dynamic behavioral analysis was developed and experimentally demonstrated on a HIL testbed emulating a smart grid SCADA system. The efficacy of the framework was shown under a wide range of attack scenarios that span several categories such as FDI, FCI, MITM, and DoS attacks. The proposed framework is designed to provide an end-to-end pipeline that is scalable and computationally lightweight to facilitate applicability to real-time monitoring of CPS and flexible for the customizability of specific sets of behavioral properties to be monitored. Future work will address the following directions to further develop and validate the proposed framework: (1) extending the underlying algorithmic components of the TRAPS framework to support automated and adaptive specification mining/refinement (e.g., tag definitions, conditions, baseline operating characteristics) along with discovery of latent dependencies; (2) extending the time-series processing and anomaly detection algorithmic components to enable flexible hybrid combinations of formal and data-driven approaches through the integration of STL-based monitoring with machine learning algorithms to address complex, stealthy, and evolving attack patterns; (3) extending the architecture from a centralized monitor to a distributed system of verifiers for enhanced scalability to large-scale grids; (4) leveraging synergies with blockchain-based data integrity mechanisms to enable robust data verification and efficient verifiable data queries; and (5) further increasing the scalability (including by using GPU acceleration and specialized hardware) throughput performance and the accuracy of the framework to facilitate deployment in large-scale CPS.

Author Contributions: Conceptualization, P.K., A.R., R.K. and F.K.; methodology, P.K., A.R., R.K. and F.K.; software, P.K. and A.R.; validation, P.K., A.R., R.K. and F.K.; investigation, P.K., A.R., R.K. and F.K.; writing—original draft preparation, P.K. and A.R.; writing—review and editing, P.K., A.R., R.K. and F.K.; visualization, P.K. and A.R.; supervision, P.K., R.K. and F.K.; funding acquisition, P.K., R.K. and F.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by DOE NETL (DE-CR0000017) and NSF SaTC (2039615).

Data Availability Statement: Data available on request from the authors.

Acknowledgments: The authors would like to thank collaborators from Narf (Prashant Anantharaman, Michael Locasto, and others) and SRI (Nick Boorman, Ulf Lindqvist) for their work on parsers for several network communication protocols used in experiments in this study as well as many helpful discussions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Faheem, M.; Shah, S.B.H.; Butt, R.A.; Raza, B.; Anwar, M.; Ashraf, M.W.; Ngadi, M.A.; Gungor, V.C. Smart grid communication and information technologies in the perspective of Industry 4.0: Opportunities and challenges. *Comput. Sci. Rev.* **2018**, *30*, 1–30. [CrossRef]
2. Tantawi, K.H.; Sokolov, A.; Tantawi, O. Advances in industrial robotics: From industry 3.0 automation to industry 4.0 collaboration. In Proceedings of the IEEE Technology Innovation Management and Engineering Science International Conference, Bangkok, Thailand, 11–13 December 2019; pp. 1–4.
3. Nafees, M.N.; Saxena, N.; Cardenas, A.; Grijalva, S.; Burnap, P. Smart grid cyber-physical situational awareness of complex operational technology attacks: A review. *ACM Comput. Surv.* **2023**, *55*, 1–36. [CrossRef]
4. Yadav, G.; Paul, K. Architecture and security of SCADA systems: A review. *Int. J. Crit. Infrastruct. Prot.* **2021**, *34*, 100433. [CrossRef]
5. Bhamare, D.; Zolanvari, M.; Erbad, A.; Jain, R.; Khan, K.; Meskin, N. Cybersecurity for industrial control systems: A survey. *Comput. Secur.* **2020**, *89*, 101677. [CrossRef]

6. Alanazi, M.; Mahmood, A.; Chowdhury, M.J.M. SCADA vulnerabilities and attacks: A review of the state-of-the-art and open issues. *Comput. Secur.* **2023**, *125*, 103028. [CrossRef]
7. ICS-CERT. Cyber-Attack Against Ukrainian Critical Infrastructure; ICS Alert (IR-ALERT-H-16-056-01); Cybersecurity and Infrastructure Security Agency. Available online: https://www.cisa.gov/news-events/ics-alerts/ir-alert-h-16-056-01 (accessed on 31 January 2026).
8. Singer, P.W. Stuxnet and its hidden lessons on the ethics of cyberweapons. *Case West. Reserve J. Int. Law* **2015**, *47*, 79.
9. TRITON Malware Remains Threat to Global Critical Infrastructure Industrial Control Systems (ICS). Available online: https://www.ic3.gov/CSA/2022/220325.pdf (accessed on 31 January 2026).
10. Industroyer2: Industroyer Reloaded. Available online: https://www.welivesecurity.com/2022/04/12/industroyer2-industroyer-reloaded/ (accessed on 31 January 2026).
11. Oldsmar Water Treatment Plant Incident Allegedly Caused by Human Error, Not Remote Access Cybersecurity Breach. Available online: https://industrialcyber.co/utilities-energy-power-water-waste/oldsmar-water-treatment-plant-incident-allegedly-caused-by-human-error-not-remote-access-cybersecurity-breach/ (accessed on 31 January 2026).
12. What's the Scoop on FrostyGoop: The Latest ICS Malware and ICS Controls Considerations. Available online: https://www.sans.org/blog/whats-the-scoop-on-frostygoop-the-latest-ics-malware-and-ics-controls-considerations (accessed on 31 January 2026).
13. Unpacking the Blackjack Group's Fuxnet Malware. Available online: https://claroty.com/team82/research/unpacking-the-blackjack-groups-fuxnet-malware (accessed on 31 January 2026).
14. ELECTRUM: Cyber Attack on Poland's Electric System 2025. Available online: https://hub.dragos.com/report/electrum-targeting-polands-electric-sector (accessed on 31 January 2026).
15. Detecting CHERNOVITE's PIPEDREAM with the Dragos Platform. Available online: https://www.dragos.com/blog/detecting-chernovites-pipedream-with-the-dragos-platform (accessed on 31 January 2026).
16. Khorrami, F.; Krishnamurthy, P.; Karri, R. Cybersecurity for Control Systems: A Process-Aware Perspective. *IEEE Des. Test* **2016**, *33*, 75–83. [CrossRef]
17. Patel, N.; Saridena, A.N.; Choromanska, A.; Krishnamurthy, P.; Khorrami, F. Learning-Based Real-Time Process-Aware Anomaly Monitoring for Assured Autonomy. *IEEE Trans. Intell. Veh.* **2020**, *5*, 659–669.
18. Wlazlo, P.; Sahu, A.; Mao, Z.; Huang, H.; Goulart, A.; Davis, K.; Zonouz, S. Man-in-the-middle attacks and defence in a power system cyber-physical testbed. *IET Cyber-Phys. Syst. Theory Appl.* **2021**, *6*, 164–177. [CrossRef]
19. Jin, M.; Lavaei, J.; Sojoudi, S.; Baldick, R. Boundary Defense Against Cyber Threat for Power System State Estimation. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 1752–1767. [CrossRef]
20. Bouramdane, A.A. Cyberattacks in Smart Grids: Challenges and Solving the Multi-Criteria Decision-Making for Cybersecurity Options, Including Ones That Incorporate Artificial Intelligence, Using an Analytical Hierarchy Process. *J. Cybersecur. Priv.* **2023**, *3*, 662–705. [CrossRef]
21. ur Rehman, M.; Bahşi, H. Process-aware security monitoring in industrial control systems: A systematic review and future directions. *Int. J. Crit. Infrastruct. Prot.* **2024**, *47*, 100719. [CrossRef]
22. Sahu, A.; Mao, Z.; Wlazlo, P.; Huang, H.; Davis, K.; Goulart, A.; Zonouz, S. Multi-source multi-domain data fusion for cyberattack detection in power systems. *IEEE Access* **2021**, *9*, 119118–119138. [CrossRef]
23. Kang, B.; McLaughlin, K.; Sezer, S. Towards a stateful analysis framework for smart grid network intrusion detection. In Proceedings of the International Symposium for ICS & SCADA Cyber Security Research, Belfast, UK, 23–25 August 2016; pp. 124–131.
24. Sprabery, R.; Morris, T.H.; Pan, S.; Adhikari, U.; Madani, V. Protocol mutation intrusion detection for synchrophasor communications. In Proceedings of the Cyber Security and Information Intelligence Research Workshop, Oak Ridge, TN, USA, 8–10 January 2013; pp. 1–4.
25. Pan, S.; Morris, T.H.; Adhikari, U. A Specification-based Intrusion Detection Framework for Cyber-physical Environment in Electric Power System. *Int. J. Netw. Secur.* **2015**, *17*, 174–188.
26. Lin, H.; Slagell, A.; Di Martino, C.; Kalbarczyk, Z.; Iyer, R.K. Adapting Bro into SCADA: Building a specification-based intrusion detection system for the DNP3 protocol. In Proceedings of the Cyber Security and Information Intelligence Research Workshop, Oak Ridge, TN, USA, 8–10 January 2013; pp. 1–4.
27. Yang, Y.; McLaughlin, K.; Littler, T.; Sezer, S.; Pranggono, B.; Wang, H. Intrusion detection system for IEC 60870-5-104 based SCADA networks. In Proceedings of the IEEE Power & Energy Society General Meeting, Vancouver, BC, Canada, 21–25 July 2013; pp. 1–5.
28. Premaratne, U.K.; Samarabandu, J.; Sidhu, T.S.; Beresh, R.; Tan, J.C. An intrusion detection system for IEC61850 automated substations. *IEEE Trans. Power Deliv.* **2010**, *25*, 2376–2383. [CrossRef]
29. Kwon, Y.; Kim, H.K.; Lim, Y.H.; Lim, J.I. A behavior-based intrusion detection technique for smart grid infrastructure. In Proceedings of the IEEE Eindhoven PowerTech, Eindhoven, The Netherlands, 29 June–2 July 2015; pp. 1–6.

30. Cheung, S.; Dutertre, B.; Fong, M.; Lindqvist, U.; Skinner, K.; Valdes, A. Using model-based intrusion detection for SCADA networks. In Proceedings of the SCADA Security Scientific Symposium, Miami Beach, FL, USA, 24–25 January 2007; pp. 127–134.
31. Yang, Y.; Xu, H.Q.; Gao, L.; Yuan, Y.B.; McLaughlin, K.; Sezer, S. Multidimensional intrusion detection system for IEC 61850-based SCADA networks. *IEEE Trans. Power Deliv.* **2017**, *32*, 1068–1078. [CrossRef]
32. Hong, J.; Liu, C.C.; Govindarasu, M. Integrated anomaly detection for cyber security of the substations. *IEEE Trans. Smart Grid* **2014**, *5*, 1643–1653. [CrossRef]
33. Yang, Y.; McLaughlin, K.; Sezer, S.; Littler, T.; Im, E.G.; Pranggono, B.; Wang, H. Multiattribute SCADA-specific intrusion detection system for power networks. *IEEE Trans. Power Deliv.* **2014**, *29*, 1092–1102. [CrossRef]
34. Nivethan, J.; Papa, M. A SCADA intrusion detection framework that incorporates process semantics. In Proceedings of the Cyber and Information Security Research Conference, Oak Ridge, TN, USA, 5–7 April 2016; pp. 1–5.
35. Fovino, I.N.; Carcano, A.; Murel, T.D.L.; Trombetta, A.; Masera, M. Modbus/DNP3 state-based intrusion detection system. In Proceedings of the IEEE International Conference on Advanced Information Networking and Applications, Perth, Australia, 20–23 April 2010; pp. 729–736.
36. Carcano, A.; Coletta, A.; Guglielmi, M.; Masera, M.; Fovino, I.N.; Trombetta, A. A multidimensional critical state analysis for detecting intrusions in SCADA systems. *IEEE Trans. Ind. Inform.* **2011**, *7*, 179–186. [CrossRef]
37. Caselli, M.; Zambon, E.; Kargl, F. Sequence-aware intrusion detection in industrial control systems. In Proceedings of the ACM Workshop on Cyber-Physical System Security, Singapore, 14 April 2015; pp. 13–24.
38. Tian, J.; Tan, R.; Guan, X.; Xu, Z.; Liu, T. Moving target defense approach to detecting Stuxnet-like attacks. *IEEE Trans. Smart Grid* **2019**, *11*, 291–300. [CrossRef]
39. Zhang, Z.; Deng, R.; Yau, D.K.Y.; Cheng, P.; Chen, J. Analysis of Moving Target Defense Against False Data Injection Attacks on Power Grid. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 2320–2335. [CrossRef]
40. Liu, B.; Wu, H. Optimal planning and operation of hidden moving target defense for maximal detection effectiveness. *IEEE Trans. Smart Grid* **2021**, *12*, 4447–4459. [CrossRef]
41. Xu, W.; Jaimoukha, I.M.; Teng, F. Robust Moving Target Defence Against False Data Injection Attacks in Power Grids. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 29–40. [CrossRef]
42. Chen, H.; Zhang, Z.; Roy, S.; Bartocci, E.; Smolka, S.A.; Stoller, S.; Lin, S. Cumulative-Time Signal Temporal Logic. *ACM Trans. Embed. Comput. Syst.* **2025**, *24*, 1–23. [CrossRef]
43. Zhang, Z.; An, J.; Arcaini, P.; Hasuo, I. CauMon: An Informative Online Monitor for Signal Temporal Logic. In Proceedings of the International Symposium on Formal Methods, Milan, Italy, 9–13 September 2024; pp. 286–304.
44. Bellanger, C.; Garoche, P.L.; Martel, M.; Picard, C. Formally proved specification of non-nested STL formulas as synchronous observers. *Sci. Comput. Program.* **2025**, *245*, 103315. [CrossRef]
45. Lee, J.; Yu, G.; Bae, K. SMT-based robust model checking for signal temporal logic. *Sci. Comput. Program.* **2025**, *246*, 103332. [CrossRef]
46. Yu, X.; Dong, W.; Li, S.; Yin, X. Model predictive monitoring of dynamical systems for signal temporal logic specifications. *Automatica* **2024**, *160*, 111445. [CrossRef]
47. Yamaguchi, T.; Hoxha, B.; Ničković, D. RTAMT—Runtime Robustness Monitors with Application to CPS and Robotics. *Int. J. Softw. Tools Technol. Transf.* **2024**, *26*, 79–99. [CrossRef]
48. Waga, M.; Matsuoka, K.; Suwa, T.; Matsumoto, N.; Banno, R.; Bian, S.; Suenaga, K. Oblivious Monitoring for Discrete-Time STL via Fully Homomorphic Encryption. In Proceedings of the International Conference on Runtime Verification, Istanbul, Turkey, 15–17 October 2024; pp. 59–69. [CrossRef]
49. Umer, M.A.; Junejo, K.N.; Jilani, M.T.; Mathur, A.P. Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations. *Int. J. Crit. Infrastruct. Prot.* **2022**, *38*, 100516. [CrossRef]
50. Vourganas, I.J.; Michala, A.L. Applications of Machine Learning in Cyber Security: A Review. *J. Cybersecur. Priv.* **2024**, *4*, 972–992. [CrossRef]
51. Nguyen, H.N.; Koo, J. Enhancing SCADA Security Using Generative Adversarial Network. *J. Cybersecur. Priv.* **2025**, *5*, 73. [CrossRef]
52. He, Y.; Mendis, G.J.; Wei, J. Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism. *IEEE Trans. Smart Grid* **2017**, *8*, 2505–2516. [CrossRef]
53. Ahmed, S.; Lee, Y.; Hyun, S.H.; Koo, I. Unsupervised Machine Learning-Based Detection of Covert Data Integrity Assault in Smart Grid Networks Utilizing Isolation Forest. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 2765–2777. [CrossRef]
54. Zhang, Y.; Wang, J.; Chen, B. Detecting false data injection attacks in smart grids: A semi-supervised deep learning approach. *IEEE Trans. Smart Grid* **2020**, *12*, 623–634. [CrossRef]
55. Hallaji, E.; Razavi-Far, R.; Wang, M.; Saif, M.; Fardanesh, B. A Stream Learning Approach for Real-Time Identification of False Data Injection Attacks in Cyber-Physical Power Systems. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 3934–3945. [CrossRef]

56. Irfan, M.; Omri, A.; Hernandez Fernandez, J.; Sciancalepore, S.; Oligeri, G. Detecting Jamming in Smart Grid Communications via Deep Learning. *J. Cybersecur. Priv.* **2025**, *5*, 46. [CrossRef]
57. Ganesh, P.; Lou, X.; Chen, Y.; Tan, R.; Yau, D.K.; Chen, D.; Winslett, M. Learning-based simultaneous detection and characterization of time delay attack in cyber-physical systems. *IEEE Trans. Smart Grid* **2021**, *12*, 3581–3593. [CrossRef]
58. Khaw, Y.M.; Jahromi, A.A.; Arani, M.F.; Sanner, S.; Kundur, D.; Kassouf, M. A deep learning-based cyberattack detection system for transmission protective relays. *IEEE Trans. Smart Grid* **2020**, *12*, 2554–2565. [CrossRef]
59. Singh, V.K.; Govindarasu, M. A cyber-physical anomaly detection for wide-area protection using machine learning. *IEEE Trans. Smart Grid* **2021**, *12*, 3514–3526. [CrossRef]
60. Cui, M.; Wang, J.; Chen, B. Flexible machine learning-based cyberattack detection using spatiotemporal patterns for distribution systems. *IEEE Trans. Smart Grid* **2020**, *11*, 1805–1808. [CrossRef]
61. Presekal, A.; Ştefanov, A.; Semertzis, I.; Palensky, P. Spatio-Temporal Advanced Persistent Threat Detection and Correlation for Cyber-Physical Power Systems Using Enhanced GC-LSTM. *IEEE Trans. Smart Grid* **2025**, *16*, 1654–1666. [CrossRef]
62. Shlomo, A.; Kalech, M.; Moskovitch, R. Temporal pattern-based malicious activity detection in SCADA systems. *Comput. Secur.* **2021**, *102*, 102153. [CrossRef]
63. Gao, D.; Reiter, M.; Song, D. On Gray-Box Program Tracking for Anomaly Detection. In Proceedings of the USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004; pp. 103–118.
64. Krishnamurthy, P.; Karri, R.; Khorrami, F. Anomaly Detection in Real-Time Multi-Threaded Processes Using Hardware Performance Counters. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 666–680. [CrossRef]
65. Konstantinou, C.; Wang, X.; Krishnamurthy, P.; Khorrami, F.; Maniatakos, M.; Karri, R. HPC-Based Malware Detectors Actually Work: Transition to Practice After a Decade of Research. *IEEE Des. Test* **2022**, *39*, 23–32. [CrossRef]
66. Zhou, X.; Ahmed, B.; Aylor, J.H.; Asare, P.; Alemzadeh, H. Hybrid Knowledge and Data Driven Synthesis of Runtime Monitors for Cyber-Physical Systems. *IEEE Trans. Dependable Secur. Comput.* **2024**, *21*, 12–30. [CrossRef]
67. Zhang, L.; Zhu, J.; Han, G.; Jin, B.; Wang, P.; Wei, X. Self-Supervised Disentangled Representation Learning for Time Series Anomaly Detection. *IEEE Internet Things J.* **2025**, *12*, 40259–40271. [CrossRef]
68. Zhang, S.; Hu, X.; Liu, J. TranBF: Deep Transformer Networks and Bayesian Filtering for Time Series Anomalous Signal Detection in Cyber-physical Systems. In Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), Niagara Falls, ON, Canada, 15–19 July 2024. [CrossRef]
69. Wang, C.; Yu, X.; Zhao, J.; Lindemann, L.; Yin, X. Sleep When Everything Looks Fine: Self-Triggered Monitoring for Signal Temporal Logic Tasks. *IEEE Robot. Autom. Lett.* **2024**, *9*, 8983–8990. [CrossRef]
70. Liu, Y.; Guo, Y. Enhancing Intrusion Detection for IoT and Sensor Networks Through Semantic Analysis and Self-Supervised Embeddings. *Sensors* **2025**, *25*, 7074. [CrossRef]
71. Iglesias Vázquez, F.; Hartl, A.; Zseby, T.; Zimek, A. Anomaly detection in streaming data: A comparison and evaluation study. *Expert Syst. Appl.* **2023**, *233*, 120994. [CrossRef]
72. Cao, Y.; Ma, Y.; Zhu, Y.; Ting, K.M. Revisiting streaming anomaly detection: Benchmark and evaluation. *Artif. Intell. Rev.* **2025**, *58*, 8. [CrossRef]
73. Steindl, G.; Schwarzinger, T.; Schreiberhuber, K.; Ekaputra, F.J. Toward Semantic Event-Handling for Building Explainable Cyber-Physical Systems. *IEEE Open J. Ind. Electron. Soc.* **2024**, *5*, 928–945. [CrossRef]
74. Qiu, C.; Deng, J.; Peng, T.; Peng, Z. Blockchain-Based Verifiable Decentralized Identities for Cyber-Physical Web 3.0. In Proceedings of the International Conference on Automation in Manufacturing, Transportation and Logistics (ICaMaL), Hong Kong, 7–9 August 2024; pp. 1–6.
75. Wu, H.; Peng, Z.; Guo, S.; Yang, Y.; Xiao, B. VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1393–1406. [CrossRef]
76. Wu, H.; Tang, Y.; Shen, Z.; Tao, J.; Lin, C.; Peng, Z. TELEX: Two-Level Learned Index for Rich Queries on Enclave-Based Blockchain Systems. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 4299–4313. [CrossRef]
77. MITRE ATT&CK Matrix for ICS. Available online: https://attack.mitre.org/matrices/ics/ (accessed on 31 January 2026).
78. MITRE EMB3D Threat Model. Available online: https://emb3d.mitre.org/ (accessed on 31 January 2026).
79. Scapy. Available online: https://scapy.net/ (accessed on 31 January 2026).
80. PyShark. Available online: https://kiminewt.github.io/pyshark/ (accessed on 31 January 2026).
81. Hammer Parsing Library. Available online: https://github.com/UpstandingHackers/hammer (accessed on 31 January 2026).
82. Anantharaman, P.; Palani, K.; Brantley, R.; Brown, G.; Bratus, S.; Smith, S.W. PhasorSec: Protocol Security Filters for Wide Area Measurement Systems. In Proceedings of the IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, Aalborg, Denmark, 29–31 October 2018; pp. 1–6.
83. Anantharaman, P.; Chachra, A.; Sinha, S.; Millian, M.; Copos, B.; Smith, S.; Locasto, M. A Communications Validity Detector for SCADA Networks. In Proceedings of the International Conference on Critical Infrastructure Protection, Virtual Event, 15–16 March 2021; pp. 155–183.

84. Zeek: An Open Source Network Security Monitoring Tool. Available online: https://zeek.org/ (accessed on 31 January 2026).
85. NFStream: Flexible Network Data Analysis Framework. Available online: https://www.nfstream.org/ (accessed on 31 January 2026).
86. Aouini, Z.; Pekar, A. NFStream: A flexible network data analysis framework. *Comput. Netw.* **2022**, *204*, 108719. [CrossRef]
87. Tshark. Available online: https://www.wireshark.org/docs/man-pages/tshark.html (accessed on 31 January 2026).
88. Chen, S.; Qian, Z.; Siu, W.; Hu, X.; Li, J.; Li, S.; Qin, Y.; Yang, T.; Xiao, Z.; Ye, W.; et al. Pyod 2: A python library for outlier detection with llm-powered model selection. In Proceedings of the Companion Proceedings of the ACM on Web Conference, Sydney, NSW, Australia, 28 April–2 May 2025; pp. 2807–2810.
89. PyOD: A Python Library for Outlier and Anomaly Detection. Available online: https://pyod.readthedocs.io/en/latest/ (accessed on 31 January 2026).
90. Breunig, M.M.; Kriegel, H.P.; Ng, R.T.; Sander, J. LOF: Identifying density-based local outliers. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2000; pp. 93–104.
91. Schölkopf, B.; Platt, J.C.; Shawe-Taylor, J.; Smola, A.J.; Williamson, R.C. Estimating the Support of a High-Dimensional Distribution. *Neural Comput.* **2001**, *13*, 1443–1471. [CrossRef] [PubMed]
92. He, Z.; Xu, X.; Deng, S. Discovering cluster-based local outliers. *Pattern Recognit. Lett.* **2003**, *24*, 1641–1650. [CrossRef]
93. Nguyen, M.N.; Vien, N.A. Scalable and Interpretable One-Class SVMs with Deep Learning and Random Fourier Features. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Würzburg, Germany, 16–20 September 2019.
94. Goodge, A.; Hooi, B.; Ng, S.K.; Ng, W.S. LUNAR: Unifying Local Outlier Detection Methods via Graph Neural Networks. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual Event, 22 February–1 March 2022.
95. Li, Z.; Zhao, Y.; Hu, X.; Botta, N.; Ionescu, C.; Chen, G.H. ECOD: Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 12181–12193. [CrossRef]
96. Ruff, L.; Vandermeulen, R.; Goernitz, N.; Deecke, L.; Siddiqui, S.A.; Binder, A.; Müller, E.; Kloft, M. Deep One-Class Classification. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018.
97. Xu, H.; Pang, G.; Wang, Y.; Wang, Y. Deep Isolation Forest for Anomaly Detection. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 12591–12604. [CrossRef]
98. Kaitai Struct. Available online: https://kaitai.io/ (accessed on 31 January 2026).