A Framework for Monitoring and Auditing the Activity of Cross-Chain Mechanisms

Jorge Santos
INESC-ID, Instituto Superior Técnico,
University of Lisbon – Lisbon, Portugal
jorge.m.santos@tecnico.ulisboa.pt

André Augusto
INESC-ID, Instituto Superior Técnico,
University of Lisbon – Lisbon, Portugal
andre.augusto@tecnico.ulisboa.pt

André Vasconcelos
INESC-ID, Instituto Superior Técnico,
University of Lisbon – Lisbon, Portugal
andre.vasconcelos@tecnico.ulisboa.pt

Miguel Correia
INESC-ID, Instituto Superior Técnico,
University of Lisbon – Lisbon, Portugal
miguel.p.correia@tecnico.ulisboa.pt

Abstract

Blockchain interoperability presents complex challenges regarding transparency and accountability. This paper proposes a modular framework for monitoring and auditing cross-chain systems by capturing, correlating, and visualizing telemetry data, namely logs, metrics, and traces produced during execution. The proposed solution provides real-time observability and post-incident auditing through customizable dashboards and automated data aggregation. The framework maintains high throughput, low latency, and scalable performance while introducing negligible operational overhead. By allowing stakeholders to perform root-cause analysis and track performance indicators, the system improves cross-chain transparency, operational resilience, and trustworthiness.

CCS Concepts

• **Applied computing** → **Enterprise interoperability**; • **Software and its engineering** → **Software design engineering**.

Keywords

Blockchain, Monitoring, Auditing, Interoperability, Evaluation

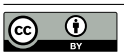
ACM Reference Format:
Jorge Santos, André Augusto, André Vasconcelos, and Miguel Correia. 2026. A Framework for Monitoring and Auditing the Activity of Cross-Chain Mechanisms. In *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26), March 23–27, 2026, Thessaloniki, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3748522.3779917

1 Introduction

The ability to monitor and conduct audits represents a crucial component for inferring conclusions about the execution of a system. Software systems are statistically bound to fail at some point [14]; however, damage can be minimized by continuously improving these systems and addressing detected flaws. To effectively mitigate the impact of failures, flaws should be identified as quickly as possible through systematic monitoring of system behavior and subsequently analyzed during audit processes to identify root causes and prevent recurrence [15].

Blockchain interoperability is a relatively new branch of study within the field of blockchain systems [12, 10]. Its importance arises from the need to address the increasing fragmentation of blockchain ecosystems, where independent networks often operate in isolation due to differences in architecture, consensus mechanisms, and data models [9, 10]. Interoperability enables seamless communication and asset exchange between heterogeneous blockchains, thus improving scalability, fostering innovation, and supporting the wide adoption of decentralized technologies [9, 12]. Thus, from the late 2010s to the early 2020s, research began to shift focus: as individual blockchain systems matured, the central challenge evolved from improving isolated platforms to enabling interaction and integration across heterogeneous blockchains [12]. An early and influential reference to the concept of *blockchain interoperability* in the *Semantic Scholar*[1] database of academic works dates back to 2014, in a work by *Back et al.* [7], where the authors propose a sidechain capable of bridging different blockchains. Their proposal garnered significant attention from the blockchain research community, inspiring subsequent research on cross-chain communication and ways of securely transferring assets between distinct blockchains.

The growing pursuit of interoperability has sparked the development of numerous cross-chain systems and applications. According to the DefiLlama database[2], there are currently at least 134 operational cross-chain protocols [16], many of which manage total locked values in the millions or even billions of dollars. However, these interoperable technologies are not flawless, especially in the early stages of production, resulting in bugs and vulnerabilities that, from 2021 to 2024, have caused reported losses of nearly 4.3 billion dollars [36]. These incidents reveal deeper challenges in the operational reliability and observability of interoperable systems. Specifically, existing approaches often lack systematic monitoring practices to promptly detect abnormal behavior [5, 13], provide limited support for rapid root cause analysis following incidents [26, 32, 33], and offer few visualization tools capable of presenting cross-chain data in a meaningful and accessible manner [9, 35]. Addressing these issues is critical to improving the transparency,



SAC '26, Thessaloniki, Greece

ACM ISBN 979-8-4007-2294-3/2026/03
https://doi.org/10.1145/3748522.3779917

[1] Available at https://www.semanticscholar.org/.
[2] Available at https://defillama.com.

resilience, and trustworthiness of cross-chain protocols, underscoring the need for enhanced monitoring and auditing capabilities in blockchain interoperability research.

In this work, we propose and implement a monitoring and auditing framework capable of capturing, storing, correlating, and displaying telemetry data–that is, execution information such as metrics, logs, and traces that describe the runtime behavior of a system–to enable users to analyze a cross-chain system's activity. The proposed solution is modular and adaptable, composed of several components, including data collectors, aggregators, and interface managers to facilitate component substitution or addition. The system receives telemetry data, including metrics, logs and traces, from a cross-chain mechanism using a telemetry collector, processes it, and redirects it to the respective data collector. The information is then processed according to the type of telemetry and sent to a telemetry aggregator that correlates different types of telemetry data to find common links to aggregate information. Finally, the data is ready to be displayed, with the dashboard manager responsible for providing an interface to the end user to create dashboards to analyze the processed data.

2 Background

This section presents background information on monitoring, auditing, observability, and the SATP-Hermes protocol.

2.1 Monitoring and Auditing

Monitoring and auditing are complementary processes that together ensure the security, reliability, and accountability of complex systems [29]. Monitoring is primarily concerned with the real time collection, aggregation, and analysis of telemetry data to identify abnormal events, performance issues, or security incidents as they occur [28, 2]. Its role is inherently diagnostic: monitoring captures signals such as metrics, logs, and traces that indicate when something has gone wrong or is trending toward failure [31]. In other words, monitoring identifies what is happening in the system.

Auditing, on the contrary, operates on a different time scale and with a different purpose. Rather than focusing on immediate detection, auditing provides a retrospective and explanatory account of system behavior [8, 23]. Through a systematic examination of past records of events, auditing seeks to explain why an event occurred, whether it aligned with established policies or invariants, and who or what was responsible for the underlying actions. Thus, auditing complements monitoring by transforming raw signals into accountability, assurance, and verifiable evidence of correctness.

When applied to blockchain and cross-chain environments, this distinction becomes particularly important. Monitoring enables the timely detection of anomalies, such as unexpected transaction delays, abnormal transaction fee values, or suspicious cross-chain message flows. Auditing, in turn, enables detailed investigations of system activity to verify transaction integrity, ensure that the cross-chain system operates without faults, and confirm that no unauthorized or malicious activities occurred [17]. Together, these two processes provide a comprehensive framework: monitoring identifies potential risks as they emerge, while auditing explains their root causes and verifies compliance with security and operational guarantees [29].

2.2 Three Pillars of Observability

To achieve observability in practice, modern monitoring and auditing systems rely on three types of telemetry data, also known as the "three pillars of observability": logs, metrics and traces [20, 30]. Logs capture discrete events and contextual information, metrics provide aggregated numerical indicators of performance over time, and traces record end-to-end execution paths across different components [27]. Each pillar offers a distinct perspective on system behavior; however, when combined, they provide a strong foundation for monitoring and auditing a system's behavior and activity [20]. Table 1 systematizes and compares the information captured, use cases and limitations of each of the telemetry data types:

Logs are collections of semi-structured or unstructured strings. They provide fine-grained detailed information along with a rich system context, serving as audit trails to track user actions and system events [1]. These trails provide information on what happened before, during and after an event, making them invaluable for understanding why unexpected behavior occurred in the system [24].

Metrics are structured numerical representations of data collected over intervals of time [20, 27]. They provide a simplified and systematic view of the performance of the monitored system and resource utilization, allowing the detection of anomalies and the observation of long-term trends [24]. Monitoring systems collect, aggregate, and analyze metrics to sift through known patterns that indicate trends that stakeholders find relevant [27].

Traces capture the end-to-end journey of a request as it propagates through a system, providing detailed visibility into its execution across components [34, 37]. A trace is composed of a sequence of spans, where each span represents a unit of work or an interaction between the system components [37]. Tracing allows operators and developers to identify performance bottlenecks, uncover dependency issues, and detect failures that may not be apparent through logs or metrics alone.

2.3 SATP-Hermes

The SATP-Hermes project[3] is a plugin implementation of the Secure Asset Transfer Protocol [22] and the Hermes [11] fault-tolerant middleware on the Hyperledger Cacti[4] interoperability framework. The Secure Asset Transfer Protocol [22] (SATP) is an asset transfer protocol between two networks, being developed by the IETF, based on the concept of trusted gateways, a type of hybrid connector [12] that runs an interoperability protocol capable of connecting heterogeneous systems such as private and public blockchains. These connectors bridge differences in architecture, consensus mechanisms, and transaction models, enabling cross-chain interactions while preserving each network's operational independence [4]. The primary objective of the SATP is to ensure the consistency of the asset state across both the origin and destination networks, guaranteeing that the asset is located in only one system or network at any given time, and that asset movements into (out of) networks via gateways can be accounted for [22]. To complement the asset transfer protocol and ensure a secure execution of the protocol, the

[3] Available at https://github.com/hyperledger-cacti/cacti/tree/main/packages/cactus-plugin-satp-hermes.
[4] Available at https://github.com/hyperledger-cacti/cacti/tree/main/.

system incorporates a crash recovery mechanism [11] that provides both recovery and rollback capabilities in the event of system or gateway failures [4]. This mechanism defines the procedures required for a crashed gateway to resume protocol execution from a consistent state, minimizing possible disruption to the overall transfer process. In extreme cases where recovery is simply not possible, the mechanism enables a controlled rollback of the protocol. Since a distributed ledger is an append-only data structure, such a rollback does not involve deleting previous transactions, instead, issuing compensatory transactions that reverse the effects of the operations already committed [11, 4]. This approach preserves the integrity and immutability of the ledgers while ensuring that the protocol's consistency guarantees remain intact.

3 Motivation

Although blockchain interoperability has gained significant traction in recent years, its practical deployment continues to face critical challenges [12]. Despite the high stakes, many existing approaches to cross-chain interoperability remain immature in terms of observability, leading to substantial financial losses [36] and low user trust [25]. These shortcomings give rise to several pressing problems, which we outline in the following subsections as distinct challenges requiring attention.

Motivation 1: Lack of Monitoring Practices. As the transition to complex applications that interconnect multiple blockchain systems accelerates and as the value locked in these systems grows, the demand for reliable cross-chain platforms with minimal error rates correspondingly increases. Existing studies have shown that protocols often take too long to react to an attack [5, 13, 19], which may be attributed to inadequate monitoring and suboptimal SecOps practices [3, 5, 13]. The lack of monitoring practices makes it difficult to ensure the correct execution of cross-chain systems, contributing to a less safe and more attack-prone operational environment [13, 5].

Motivation 2: Need for Quick Root-Cause Analysis. The necessity for rapid post-mortem identification of root causes is critical to ensure timely issue resolution in cross-chain environments. As cross-chain transactions often involve heterogeneous systems, each with distinct consensus mechanisms, communication protocols, and security assumptions, the task of tracing failures back to their origin is sometimes complicated [26]. For instance, the Ronin Network hack in August 2024 resulted in the theft of over $12 million, yet the bridge was paused approximately 40 minutes to assess the vulnerability [33]. Similarly, the Wormhole bridge exploit in February 2022 led to a loss of $320 million, the attack's root cause traced back to a vulnerability in the signature verification process [32], highlighting the critical need for quick and robust root cause analysis capabilities. Without quick and robust root cause analysis capabilities, the operational resilience and trustworthiness of cross-chain systems remains severely constrained.

Motivation 3: Lack of Visualization Tools. A survey concluded that there is no publicly available mechanism for gathering and visualizing cross-chain operations metrics [9]. This limitation poses a significant challenge, as effective monitoring and auditing require not only the collection of telemetry data, but also its clear presentation in a form that stakeholders can quickly interpret. The lack of user-centric visualization tools hampers the ability to perform timely post-mortem analyzes, identify root causes, and mitigate actions. Consequently, there is a pressing need for visualization frameworks that can integrate cross-chain data into intuitive dashboards, providing actionable insights for developers, auditors, and end-users [9, 35].

4 Design and Implementation

To ensure correct execution and facilitate the process of examining cross-chain mechanisms, we propose a system composed of several components, including data exporters, telemetry aggregators, and dashboard management infrastructures. This system is designed to address the interoperability challenges identified in Section 3, offering a solution capable of providing real-time monitoring, quick analysis of events, and information based on the needs of stakeholders. This section starts by introducing the system requirements. Secondly, we present the implementation and design decisions of the proposed solution.

4.1 Requirements

The monitoring framework is expected to capture telemetry data to monitor and audit the execution of a system, allowing users to gain insights into each process step. The non-functional requirements of the system are:

(1) *Adaptability*: Since new cross-chain mechanisms are emerging every day, it is necessary for the framework to be adaptable enough to be implemented in other use cases.
(2) *High Processing Capacity*: Since telemetry is generated and analyzed in real-time, the system needs to have the ability to process a large amount of telemetry information per second(high throughput) [24].
(3) *Availability*: A monitoring system should function with minimal down-time to ensure that there is no unrecorded information.
(4) *Centralization*: With several different types of telemetry information being captured by different monitoring components, the system should be capable of aggregating the information for the user to explore [18].
(5) *Seamless Integration*: The implementation should be non-disruptive and ensure compatibility with existing systems and components. Transaction latency, for example, should not be hindered by the integration of the monitoring and auditing system.
(6) *Scalability*: The solution should allow for the extension and integration of future project components into the existing monitoring system without hindering performance.
(7) *Customization*: With a large amount of data being captured, a degree of filtering and customization is required to provide different users with their relevant information.

Concerning the functional requirements, Fig. 1 presents a use case diagram of the monitoring framework. The identified functionalities can be summarized as follows: (1) Capture execution metrics. (2) Capture execution logs. (3) Capture execution traces. (4) Correlate logs and traces. (5) Define new metrics to be captured. (6) Create dashboards. (7) Visualize telemetry data.

Table 1: Summary view of telemetry data types' information captured, use cases and limitations.

Data Type	Information Captured	Use Cases	Limitations
Logs	- Discrete events, system state, or user actions. - Context about past, present and future events.	- Debugging; - Auditing and security monitoring; - Compliance; - Incident response;	- Storage and processing costs; - Excessive logging overhead; - Requires careful structuring; - Privacy/security concerns.
Metrics	- Numerical and aggregated data collected over time; - Quantitative overview of system health.	- Monitoring system health; - Tracking service-level objectives; - Alerting on threshold breaches; - Performance trend analysis;	- Lacks detailed context; - Simplification of complex behaviors; - Requires well-designed metrics.
Traces	- End-to-end paths of requests across services; - Temporal and causal relationships between components.	- Identification of latency sources; - Dependency mapping; - Failure detection; - Correlation with other data types;	- Storage and processing costs; - Requires instrumentation across services; - Dependent on visualization tools; - Implementation complexity;

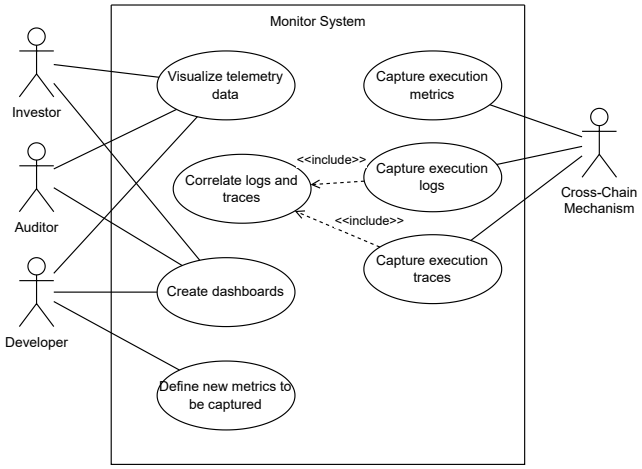


Figure 1: Use case diagram for the monitoring and auditing system's functional requirements.

The proposed system should capture and correlate execution metrics, logs, and traces. This information should be available for visualization by users, allowing them to create dashboards. Finally, developers should also be able to define new metrics to be captured.

4.2 System Architecture

This section presents the system architecture. Our solution consists of several components that enable the processing and display of telemetry data: the Telemetry Collector, Data Exporters, including a collector for each telemetry data type, a Telemetry Aggregator, and a Dashboard Manager. Fig. 2 depicts the proposed modular system architecture. The proposed solution is designed to be adaptable, allowing for implementation in different cross-chain use cases. The framework is also easily scalable and integrable, maintaining a modular architecture that simplifies changing modules and the integration of new system components without hampering performance. Furthermore, the system centralizes all telemetry information to allow for simplified user access.

Cross-chain systems function by interacting with isolated blockchains. When these interactions occur the cross-chain system executes a certain protocol (for example, a cross-chain transaction, or cross-chain state proof) which in turn produces telemetry data, such as logs of the execution, metrics increments for the number of transactions processed and traces of the execution of the transaction from end-to-end. This telemetry is essential for providing stakeholders with accurate information on the system's execution and current state, enabling effective monitoring and auditing. Additionally, developers are empowered to define new metrics tailored to emerging use cases or specific performance indicators, ensuring that the monitoring framework remains adaptable and capable of capturing all relevant operational aspects of the cross-chain system.

4.2.1 Telemetry Collector. When telemetry data is produced, it should be directed to a component capable of collecting, interpreting, and forwarding it for further processing. The Telemetry Collector fulfills this role by acting as an abstraction and unification layer for all telemetry data types. Although this component could be bypassed in the overall pipeline of telemetry processing by simply connecting the cross-chain system to each of the data exporters, it was included to simplify the management of all data types, as a single collector agent reduces the operational complexity and allows the data exporters to be easily substituted without requiring new instrumentation within the cross-chain system's codebase.

In addition to basic collection, the Telemetry Collector also provides a set of intermediate processing capabilities. These include data enrichment with contextual metadata (e.g., operational contexts, function details, or transaction related information), data normalization, and preliminary aggregation. Such functionality ensures that only relevant and structured telemetry data is forwarded to subsequent components, thereby improving the system's performance and scalability.

Furthermore, the modular design of the Telemetry Collector promotes extensibility. New telemetry formats or protocols can be integrated through custom receivers or data exporters without altering the collector's core logic, instead relying on simple reconfigurations of the collector.

4.2.2 Data Exporters. After the telemetry data is collected and preprocessed by the Telemetry Collector, it is forwarded to the respective data exporter. The processing step greatly differs across different data type collectors. As seen in Section 2.2, each telemetry type captures different dimensions of the system's execution, requiring specific handling, transformation, and storage mechanisms. The Data Exporters are responsible for ensuring that the collected

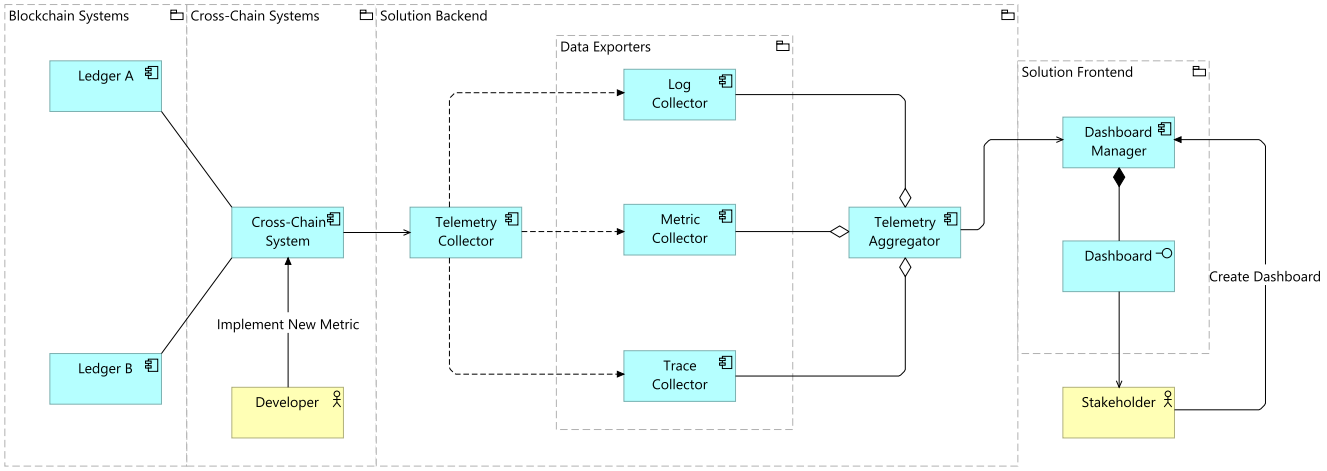


Figure 2: System architecture in Archimate language.

telemetry is correctly formatted, transmitted, and stored in the appropriate backend systems.

Each exporter is designed to handle a single telemetry signal type (logs, metrics, or traces) in order to optimize the performance and reliability of the data flow. For instance, log exporters handle unstructured or semi-structured textual data that captures discrete events and error messages. Metric exporters, on the other hand, manage numerical data representing quantitative measurements (e.g., the number of transactions, latencies, and operational gas used). Finally, trace exporters are responsible for conveying distributed tracing data that describes end-to-end execution paths across the cross-chain system.

The use of dedicated exporters for each telemetry type allows for fine-grained control over the data transmission process, including the definition of export intervals, batching policies, and retry strategies in the event of network failures. Moreover, exporters can be configured to support multiple output destinations simultaneously, enabling data replication for redundancy or multi-platform observability. This capability ensures both fault tolerance and flexibility in adapting to different deployment environments and analytical requirements.

From a design perspective, the exporters maintain a modular and loosely coupled structure. This modularity allows developers to easily substitute one exporter implementation for another without impacting the rest of the telemetry pipeline. For example, a Prometheus metric exporter[5] can be replaced with a Mimir metric exporter[6] without modifying the data generation or collection logic. Such flexibility is essential for ensuring the long-term maintainability and interoperability of the proposed framework.

4.2.3 Telemetry Aggregator. After receiving specific treatment, the telemetry data flows to the Telemetry Aggregator. This component serves as the central integration layer of the monitoring and auditing framework, responsible for unifying and correlating telemetry signals originating from different data sources. The aggregation of information enables the correlation of metrics, logs and traces, thus providing deeper insights into the system's operational behavior and facilitating the detection of cross-component dependencies and anomalies.

The Telemetry Aggregator's primary function is to consolidate the various telemetry streams into a coherent and queryable dataset. By correlating metrics (quantitative performance indicators), logs (event-based textual information), and traces (end-to-end execution paths), the aggregator constructs a holistic view of the system's execution. This correlation allows users to traverse from high-level performance trends down to specific root causes of failures or inefficiencies. For instance, an abnormal increase in transaction latency (metric) can be correlated with a specific error message (log) and traced back to the responsible cross-chain operation (trace).

To achieve this, the Telemetry Aggregator employs synchronization and indexing mechanisms that align telemetry signals based on timestamps and trace IDs. This process ensures that events captured across multiple systems or services are accurately correlated in time and context. Additionally, the aggregator supports temporal and contextual queries, enabling users to perform analyzes over specific execution windows or components within the cross-chain architecture.

4.2.4 Dashboard Manager. Once the telemetry data has been aggregated and correlated, it is made accessible through the Dashboard Manager, the component responsible for managing the visualization and interaction layer of the monitoring and auditing framework. This module provides end users, such as developers, system operators, and auditors, with intuitive and customizable dashboards for analyzing the behavior of the cross-chain mechanisms in real time.

The Dashboard Manager acts as the primary interface between users and the underlying telemetry data. It retrieves information from the Telemetry Aggregator and allows for data exploration, through a combination of graphical representations like, for example, charts, tables and time series, enabling users to observe key performance indicators, identify trends, and audit system events across different telemetry dimensions.

[5]Available at https://prometheus.io/docs/introduction/overview/.
[6]Available at https://grafana.com/docs/mimir/latest/.

A key design goal of the Dashboard Manager is customization. Different users may have distinct monitoring needs depending on their role or focus area. For instance, a developer may prioritize visualizing execution traces and error logs to debug system behavior, whereas an investor may focus on number of successful transactions, gas fees or even total locked value of a certain token. To accommodate this, the Dashboard Manager allows users to create, modify, and save personalized dashboard configurations, defining specific queries, filters, and visualization types suited to their objectives.

Moreover, the Dashboard Manager supports real-time and historical data visualization, enabling users not only to monitor ongoing system activity but also to analyze past executions for auditing and post-incident reviews. This dual capability is essential for verifying system integrity and evaluating performance over time.

4.3 System Implementation

The monitoring and auditing system proposed in this work was fully implemented as part of the *SATP-Hermes* package of the Hyperledger Cacti project (introduced in Section 2.3), that connects Fabric, Besu and Ethereum networks. The implementation integrates the custom components defined in Section 4.2–namely the *Telemetry Collector*, *Data Exporters*, *Telemetry Aggregator*, and *Dashboard Manager*–to form a cohesive and extensible monitoring and auditing framework for cross-chain systems. To support these components, a telemetry backend was developed around the open-source *Grafana/OTel-LGTM* stack[7], which provides a preconfigured environment including the OpenTelemetry Collector, Prometheus, Loki, Tempo, and Grafana [21]. This stack serves as the foundation upon which the system's custom integration and observability logic were built. The proposed framework extends their capabilities through the implementation of tailored data export mechanisms, correlation logic, and analytical dashboards that fulfill the requirements of the SATP-Hermes cross-chain mechanism. A stable version of the implementation source code is available *here*.

4.3.1 Grafana/Otel-LGTM Components. The Grafana/OTel-LGTM docker image is an open-source framework composed of an OpenTelemetry Collector, three telemetry data sources, Prometheus, Loki and Tempo, and Grafana for data visualization.

The OpenTelemetry Collector works as the proposed architecture's Telemetry Collector, receiving execution data. This data is then relayed to the correspondent collector, in this case, Prometheus for metrics, Tempo for traces and Loki for logs. Inside these collectors, each type of telemetry is processed accordingly to make the information usable for querying. Finally, it is sent to Grafana, which serves as both the Telemetry Aggregator and the Dashboard Manager (from section 4.2), correlating telemetry data to display more concise insights and allow for a better and faster understanding of the execution of the cross-chain mechanism. As mentioned, it also serves as the Dashboard Manager, storing user interfaces that allow the end user to create and get personalized dashboards with the information deemed relevant.

4.3.2 Captured Telemetry. Thirteen metrics were implemented, which can be seen in Table 2, capturing each different performance aspects of the execution. To support other implementations and needs, new metrics can/should be defined. For logs, the ones present in the project already were kept, as a comprehensive overview of the system was in place. The implementation of tracing is deliberately limited in scope, as not all project components are instrumented with tracing logic. As mentioned in Section 2.2, if functions which contain simple programming logic are incorporated with tracing logic, the processing overhead may affect the performance of the overall system. Thus, tracing is applied selectively to components whose behavior is considered more complex and critical for observability and diagnostics purposes.

4.3.3 Grafana Dashboards. To allow users to interact with the monitoring framework and extract meaningful insights, a set of Grafana dashboards was developed. These dashboards act as the main visualization and exploration layer, providing both high-level overviews and detailed debugging tools.

The dashboards combine metrics, logs, and traces, leveraging Grafana's correlation capabilities. Specifically:

- **System Overview Dashboard**: Displays the current number of connected DLTs, gateways, and supported assets. It includes real-time gauges, counters for total sessions and transactions, and a breakdown of transaction outcomes (successful vs. failed).
- **Financial Metrics Dashboard**: Focused on value flows and economic performance, it reports cumulative exchanged value and average resource consumption per transaction (e.g., gas usage). These visualizations target decision makers concerned with efficiency and sustainability.
- **Debug and Performance Dashboard**: Presents transaction duration histograms, operation latencies, and failure rates. Developers can use this dashboard to detect bottlenecks and anomalous behaviors.

All dashboards were created using Grafana's built-in editor, with queries directly targeting the Prometheus, Loki, and Tempo data sources. As new metrics are added to the monitoring framework, dashboards can be updated by adding new panels or changing queries. This design allows for constant adjustments to new cross-chain mechanisms and evolving project needs.

5 Evaluation

This section evaluates the proposed framework for monitoring and auditing. We describe evaluation methodology and each of evaluation metrics. Later, the information provided by the evaluation is collected, analyzed and discussed.

5.1 Evaluation Methodology

The evaluation focuses on the monitoring and auditing system as a whole, meaning single operations such as the individual creation of metrics or the individual creation of logs are not considered. The assessment suite comprises both non-functional and functional evaluation, thereby ensuring a comprehensive analysis of the system's operational effectiveness and performance characteristics.

We answer four questions with the experiments: i) Are the implemented metrics relevant in the context of cross-chain operations? ii) what is the maximum throughput the framework can achieve,

[7] Available at https://hub.docker.com/r/grafana/otel-lgtm.

i.e., how much telemetry data can be processed? iii) what is the latency overhead of the monitoring system in the latency of a transaction?, and iv) what is the cost, in terms of storage, of storing the telemetry data produced in the proposed system, i.e., how do the storage requirements scale with system growth? Answers to such questions allow us to conclude the suitability of the implemented solution in regards to the proposed use case.

5.2 Implementation Evaluation

In this section, we assess the system under controlled experimental conditions that emulate a realistic cross-chain environment, including a gateway connecting heterogeneous ledgers (Ethereum and Besu) and the Otel-LGTM monitoring and auditing infrastructure. The evaluation focuses on key aspects such as system throughput, transaction latency, storage requirements, and the coverage of implemented metrics. By analyzing these dimensions, we aim to quantify the framework's ability to capture telemetry data at scale, its impact on transaction execution, and its suitability for continuous, high-frequency monitoring in cross-chain scenarios. This implementation-focused assessment provides empirical evidence supporting the system's robustness, scalability, and practical applicability.

5.2.1 Coverage. The goal of the coverage assessment is to verify that the implemented telemetry metrics adequately cover the main non-functional concerns identified in prior research, ensuring that the system effectively captures relevant aspects of cross-chain performance and interoperability.

Belchior et al. [9] identified performance metrics such as end-to-end latency, throughput, and cost (transaction fees), as the primary concerns for cross-chain analysis. Besides these, energy consumption, carbon footprint, parties endorsing transactions (for Hyperledger Fabric-based blockchains exclusively) and cross-chain logic were other concerns of relevant parties in the context of cross-chain operations. Of these seven identified metric concerns, we implemented the metrics as shown in Table 2.

Overall, five of the seven metrics were implemented. Table 2 is divided in sections. First, the reddish section refers to the aforementioned performance metrics, all implemented. Then, the yellowish and blueish section corresponding to the other implement metrics. The metric for tracking cross-chain logic was decomposed in several metrics, that together give the user an overview of the cross-chain system activity and logic. For parties endorsing transactions, is not exactly an implemented metric, but an attribute (an additional information), something that can still be tracked using the transaction metrics. Finally, energy consumption and carbon footprint are currently not implemented (grey color in the table), as these two indicators are not yet implemented within the scope of the SATP-Hermes project. However, when these features are added in the future, tracking their values will not be hard, as the only requirement to track them is to define the new metric and record the values, as done in other metrics, such as operation_duration for example.

5.2.2 System Throughput. The throughput evaluation measures how many telemetry data points the monitoring framework can handle per second under varying load conditions. To assess this, we executed a series of benchmarks using different iteration counts, ranging from 1,000 to 1,000,000 operations, focusing on counters, histograms, logs, and spans. Gauges were excluded from this analysis due to their read-only nature and negligible overhead compared to the other telemetry types. Table 3 summarizes the results of these benchmarks.

A transaction using the SATP-Hermes protocol produces 16 counter operations (including both creation and increment), 21 histogram operations (including both creation and recording), 1572 logs, and 537 spans. The values presented in Table 4 were obtained using the best-performing throughput measurements (i.e., the highest operations-per-second values, corresponding, in general, to the highest values of iterations) from Table 3. For each telemetry type, the maximum number of transactions per second was computed by dividing the measured throughput in operations per

Table 2: Survey vs. implemented metrics: performance metrics in red, metrics related to interoperability and operational states in yellow, metrics implemented differently in blue, currently not implemented metrics in grey.

Survey Metric [9]	Implemented Metric	Description
end-to-end latency	transaction_duration	Transaction duration in milliseconds
end-to-end throughput	successful_transactions	Total number of successful transactions
end-to-end cost (fees)	transaction_gas_used	Gas used during transaction
cross-chain logic	initiated_transactions	Total number of initiated transactions
	failed_transactions	Total number of failed transactions
	ongoing_transactions	Total number of ongoing transactions
	total_value_exchanged	Total token value exchanged
	operation_duration	Operation duration in milliseconds
	operation_gas_used	Gas used during operation
	created_sessions	Total number of sessions created
	number_of_supported_assets	Current number of supported assets
	connected_DLTs	Current number of connected DLTs
	gateways	Current number of gateways connected
parties endorsing transactions	*	-
carbon footprint	-	-
energy consumption	-	-

Table 3: Throughput results for counters, histograms, logs, and spans at various iteration counts.

Iterations	Counter (ops/sec)	Histogram (ops/sec)	Logs (ops/sec)	Spans (ops/sec)
1,000	41,667	26,316	125,000	16,393
5,000	27,473	51,020	151,515	33,557
10,000	65,359	80,645	129,870	43,103
50,000	115,207	104,384	217,391	68,399
100,000	128,205	111,483	246,305	72,202
500,000	167,785	122,279	269,978	83,794
1,000,000	154,107	127,861	257,798	85,521

Table 4: Analysis of Throughput Results.

	Operations per Transaction	Operations per Second	Maximum Transactions per Second	Maximum Transactions per Year
Counter	16	167,785	10,486.56	$3.31 * 10^{11}$
Histogram	21	127,861	4,735.59	$1.50 * 10^{11}$
Logs	1,572	269,978	171.74	$5.42 * 10^{9}$
Spans	537	85,521	159.26	$5.02 * 10^{9}$

second (O_{sec}) by the number of operations generated per transaction (O_{cctx}), following ($T_{sec} = O_{sec}/O_{cctx}$). The corresponding maximum number of transactions per year was then estimated as ($T_{year} = T_{sec} * 31,536,000$), where 31,536,000 represents the total number of seconds in one year. This calculation provides an upper-bound estimation of the system's transaction-handling capacity under continuous operation at peak throughput.

Using the results of the data generated for five bridges (*CCTP*, *CCIP*, *Stargate (Taxi)*, *Stargate (Bus)* and *Across*) during the last 6 months of 2024 (from Jun 1, 2024 to Dec 31, 2024) [6], we assume there is an upper bound of $3,864,421 * 2 = 7,728,842$ transactions per year. Considering the best results for each type of telemetry data points, we can conclude that the implemented monitoring framework exhibits sufficient capacity to handle the expected operational load. Even under conservative assumptions, where the system must accommodate up to $7.7 * 10^6$ transactions per year, the measured throughput across all telemetry categories (counters, histograms, logs, and spans) exceeds the required processing rate by several orders of magnitude. This indicates that the monitoring infrastructure is not only capable of sustaining current workloads but also possesses substantial performance capacity for future scaling.

5.2.3 Transaction Latency. Understanding how the monitor framework impacts the execution time (latency) of a transaction is crucial to justify the suitability of the proposed solution as a viable component of a project. In case the solution affects a transaction's end-to-end latency too much, the monitoring system might be deemed unfit as it stands. To evaluate the effects of the monitoring code, we run 50 transactions with and without the monitoring framework functioning.

We defined a transaction's end-to-end latency in the scope of the project SATP-Hermes as:

transaction end-to-end latency = transferCompleteMessageTimestamp - transferCommenceMessageTimestamp

Table 5 showcases the latency results and the difference between them in percentage. The minimum, maximum, median, and average latencies exhibit differences close or below 1%, which are within the expected variance of results. This suggests that the monitoring framework imposes little overhead under normal operating conditions. However, the analysis of higher-order statistics reveals a more nuanced picture. In particular, the p95 latency increases by 25.18% when monitoring is enabled. This considerable rise shows that the monitoring code introduces additional variability in latency, disproportionately affecting a subset of transactions. Such a shift in the tail distribution implies that while most operations are not affected, a non-negligible fraction of executions experiences a decrease in performance. The standard deviation, which rises by 16.04%, corroborates this conclusion, highlighting an overall increase in dispersion. On the other hand, the p99 latency increases by only 1.61%, which suggests that the most extreme outliers are not significantly hindered by the monitoring logic.

Table 5: Latency of running 50 transactions with and without monitoring.

	W/out Monitoring	W/ Monitoring	Diff
Min	13998	14036	+0.27%
Max	20231	20557	+1.61%
Average	15124.08	15223.86	+0.66%
Median	14865	14894	+0.20%
p95	15890	19892	+25.18%
p99	20231	20557	+1.61%
Std. Dev.	1130.46	1311.78	+16.04%

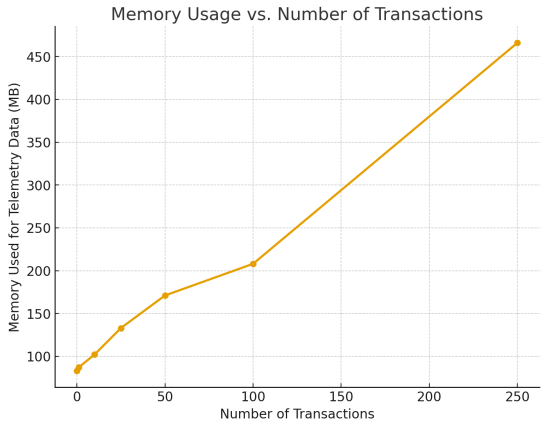


Figure 3: Memory Usage vs Number of Transactions.

5.2.4 Storage. The goal of storage evaluation is to forecast the monitoring system's long-term storage needs. We address this through an experimental analysis, where telemetry data is produced and stored in the docker image, comparing the size of the data folder as we vary the number of executed transactions.

The transactions were run sequentially and sent 100 tokens each from the *Besu network*[8] to the *Ethereum network*[9], producing runtime metrics, traces and logs. Telemetry data was exported from the data sources to Grafana with a 1 second interval, offering better temporal granularity compared to larger intervals, while reducing the volume of captured data relative to smaller intervals.

Fig. 3 showcases the relation between the number of transactions and the memory used to store telemetry data, for 1 second export intervals. By producing a linear regression of the storage costs, we can predict how much storage is necessary for one year of transactions. The linear regression for this values is:

$$S = 1.49x + 85.69 \quad (1)$$

Using the results of the data generated for five bridges (*CCTP*, *CCIP*, *Stargate (Taxi)*, *Stargate (Bus)* and *Across*) during the last 6

[8]Documentation at https://besu.hyperledger.org/.
[9]Documentation at https://ethereum.org/developers/docs/.

months of 2024 (from Jun 1, 2024 to Dec 31, 2024) [6], we assume there is a lower bound of $11,430 * 2 = 22,860$ transactions per year (tpy), an upper bound of $3,864,421 * 2 = 7,728,842$ transactions per year and an average of $11,274,334,43 * 2/5 \simeq 4,509,734$ transactions per year. Neglecting the constants from the linear regression (as these are irrelevant when studying the overall system behavior on a large number of transactions), the total storage required for a year of execution of the monitoring system is given by:

$$S_{Total} = 1.49 * tpy \quad (2)$$

With a tpy = 22,860 as lower bound, a tpy = 7,728,842 as upper bound, and an average tpy = 4,509,734 and from Equations 1 and 2, the monitoring system requires between 34.06GB and 11.52TB, with an average of 6.72TB per year.

5.3 Discussion

In Section 3, we outlined the three main problems; now we explain how we addressed them in our solution proposal, one by one.

First, in Section 3 we discussed how current cross-chain protocols may lack monitoring practices, thus contributing to a less safe and more attack-prone operational environment. To address this problem, our solution enables users to implement a customizable monitoring framework, based on open-source technologies such as the Otel-LGTM docker image. By leveraging user defined metrics, logs and traces, tailored for the user's specific business use case, the framework allows stakeholders to observe relevant aspects of the system's execution. Through the definition of custom metrics, users can quantify and monitor domain-specific events, such as the number of cross-chain transactions or the total value exchanged across networks. Similarly, logs and traces provide qualitative and contextual information that help identify abnormal behavior, failures, or performance bottlenecks in real time. This holistic observability capability ensures that system operators are continuously informed about the operational state of the cross-chain protocol, enabling proactive incident detection and response. Ultimately, by introducing a structured and extensible monitoring practice into cross-chain environments, the solution directly mitigates the lack of visibility that often leads to undetected errors, vulnerabilities, or inefficiencies, therefore contributing to a safer and more auditable operational landscape.

Second, in Section 3 we discussed the importance of quick audit capabilities in a monitored cross-chain system. Given the distributed and multi-ledger nature of these environments, pinpointing where and why a failure occurred can be challenging, often involving multiple gateways, ledgers, and system components. To address this issue, our framework integrates logs, metrics, and traces under a unified observability layer, allowing for temporal and contextual correlation across telemetry sources. This means that when an anomaly occurs (such as a failed transaction or an unexpected delay) operators can trace its propagation path through spans, inspect associated logs for contextual details, and examine correlated metrics to identify deviations from normal behavior. Furthermore, by leveraging custom dashboards in Grafana, users can visualize dependencies and quickly navigate from high-level indicators (e.g., increased transaction latency) to specific root causes (e.g., failure in a gateway operation). This integrated audit and analysis capability significantly shortens the mean time to recover from incidents, enhancing the overall reliability and maintainability of cross-chain operations.

Lastly, in Section 3 we discussed the lack of visualization tools capable of gathering and visualizing cross-chain operation metrics. Cross-chain systems handle a considerable amount of operations per day, moving high quantities of money, which in turn generates interest from several stakeholders of the ecosystem. Analytics are, thus, important in this context to keep up with an overview of the system performance. In order to address this need, our proposed solution integrates captured metrics, logs, and traces into a Dashboard Manager, enabling real-time system evaluation. Through user-defined panels and queries, different stakeholders can visualize relevant data, such as transaction throughput and operational costs (e.g. for investors), as well as cross-chain activity indicators like the number of active sessions, gateways, latencies and supported assets (e.g. for developers). This multi-layered visualization approach not only enhances system transparency for stakeholders, but also facilitates proactive performance monitoring. By transforming raw telemetry into actionable insights, the framework empowers users to gain a holistic understanding of cross-chain operations, bridging the visibility gap that currently hinders effective management and optimization of interoperable blockchain systems.

Overall, the evaluation demonstrates that the proposed monitoring and auditing framework effectively addresses the challenges identified in Section 3. By combining customizable metrics, logs, and traces with a flexible visualization layer, the system provides comprehensive monitoring, enables rapid root-cause analysis, and ensures transparency for multiple stakeholders. The functional assessments confirm that the framework can handle high operational loads with minimal impact on transaction performance, while maintaining the ability to scale and not hindering long-term resource costs. Moreover, the modular and extensible design ensures that the system can evolve to incorporate additional metrics, such as energy consumption or carbon footprint, without requiring significant redevelopment. Taken together, these results validate the practical applicability, scalability, and robustness of the framework, positioning it as a reliable solution for enhancing security, auditability, and operational insight in interoperable blockchain networks.

6 Conclusion

This work introduced a modular monitoring and auditing framework that enhances the observability and accountability of cross-chain systems. By unifying logs, metrics, and traces into a coherent visualization and analysis system, the solution enables real-time monitoring and efficient post-mortem analysis of cross-chain operations. Experimental results confirm that the framework supports high performance with minimal impact on latency and substantial scalability, addressing the core challenges of monitoring, rapid diagnosis, and stakeholder transparency. By improving reliability, transparency, and trust across heterogeneous blockchain networks, the proposed system strengthens the overall interoperability ecosystem, leading to safer and more efficient cross-chain collaboration.

Acknowledgments

This work was financially supported by Project Blockchain.PT – Decentralize Portugal with Blockchain Agenda, (Project no 51), WP

7: Interoperability, Call no 02/C05-i01.01/2022, funded by the Portuguese Recovery and Resilience Program (PRR), The Portuguese Republic and The European Union (EU) under the framework of Next Generation EU Program. This work was also supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/50021/2025 e UID/PRR/50021/2025 (INESC-ID).

References

[1] Ashar Ahmad, Muhammad Saad, and Aziz Mohaisen. 2019. Secure and transparent audit logs with blockaudit. (2019). https://arxiv.org/abs/1907.10484 arXiv: 1907.10484 [cs.DC].

[2] Riyaz Ahamed Ariyaluran Habeeb, Fariza Nasaruddin, Abdullah Gani, Ibrahim Abaker Targio Hashem, Ejaz Ahmed, and Muhammad Imran. 2019. Real-time big data processing for anomaly detection: a survey. *International Journal of Information Management*, 45, 289–307. doi:https://doi.org/10.1016/j.ijinfomgt.2018.08.006.

[3] André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. 2024. Sok: security and privacy of blockchain interoperability. In *2024 IEEE Symposium on Security and Privacy (SP)*, 3840–3865. doi:10.1109/SP54263.2024.00255.

[4] André Augusto, Rafael Belchior, Imre Kocsis, László Gönczy, André Vasconcelos, and Miguel Correia. 2023. CBDC bridging between Hyperledger Fabric and permissioned EVM-based blockchains. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 1–9. doi:10.1109/ICBC56567.2023.10174953.

[5] André Augusto, Rafael Belchior, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2025. Xchainwatcher: identifying anomalies in cross-chain bridges. In *Proceedings of the 26th International Middleware Conference*, 413–426.

[6] André Augusto, André Vasconcelos, Miguel Correia, and Luyao Zhang. 2025. Xchaindatagen: a cross-chain dataset generation framework. (2025). https://arxiv.org/abs/2503.13637 arXiv: 2503.13637.

[7] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew K. Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. 2014. Enabling Blockchain Innovations with Pegged Sidechains. Whitepaper. Blockstream. https://api.semanticscholar.org/CorpusID:61340590.

[8] Ayoosh Bansal, Anant Kandikuppa, Chien-Ying Chen, Monowar Hasan, Adam Bates, and Sibin Mohan. 2022. Ellipsis: towards efficient system auditing for real-time systems. (2022). https://arxiv.org/abs/2208.02699 arXiv: 2208.02699.

[9] Rafael Belchior, Sabrina Scuri, Nuno Nunes, Thomas Hardjono, and André Vasconcelos. 2024. Towards a standard framework for blockchain interoperability: a position paper. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 1–5. doi:10.1109/ICBC59979.2024.10634443.

[10] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2024. A brief history of blockchain interoperability. *Commun. ACM*, 67, 10, (Sept. 2024), 62–69. doi:10.1145/3648607.

[11] Rafael Belchior, André Vasconcelos, Miguel Correia, and Thomas Hardjono. 2022. Hermes: fault-tolerant middleware for blockchain interoperability. *Future Generation Computer Systems*, 129, 236–251. doi:https://doi.org/10.1016/j.future.2021.11.004.

[12] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: past, present, and future trends. *ACM Computing Surveys*, 54, 8, (Oct. 2021). doi:10.1145/3471140.

[13] Nikita Belenkov, Valerian Callens, Alexandr Murashkin, Kacper Bak, Martin Derka, Jan Gorzny, and Sung-Shine Lee. 2025. Sok: a review of cross-chain bridge hacks in 2023. (2025). https://arxiv.org/abs/2501.03423 arXiv: 2501.03423.

[14] Ricky W. Butler and George B. Finelli. 1993. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19, 1, 3–12. doi:10.1109/32.210303.

[15] Sandeep Dalal and Rajender Singh Chhillar. 2013. Empirical study of root cause analysis of software failure. *SIGSOFT Softw. Eng. Notes*, 38, 4, (July 2013), 1–7. doi:10.1145/2492248.2492263.

[16] DefiLlama. [n. d.] https://defillama.com/protocols/bridge.

[17] Haotian Deng, Zihao Wang, Yajie Wang, Licheng Wang, Liehuang Zhu, and Chuan Zhang. 2025. A secure cross-account audit scheme for cross-chain transactions. In *Algorithms and Architectures for Parallel Processing*. Tianqing Zhu, Jin Li, and Aniello Castiglione, (Eds.) Springer Nature Singapore, Singapore, 334–350.

[18] Jennie Duggan et al. 2015. The bigdawg polystore system. *SIGMOD Rec.*, 44, 2, (Aug. 2015), 11–16. doi:10.1145/2814710.2814713.

[19] Elliptic. [n. d.] North Korea's Lazarus group identified as exploiters behind $540 million Ronin bridge heist. (). https://www.elliptic.co/blog/540-million-stolen-from-the-ronin-defi-bridge.

[20] Radoslav Gatev. 2021. Observability: logs, metrics, and traces. In *Introducing Distributed Application Runtime (Dapr): Simplifying Microservices Applications Development Through Proven and Reusable Patterns and Practices*. Apress, Berkeley, CA, 233–252. ISBN: 978-1-4842-6998-5. doi:10.1007/978-1-4842-6998-5_12.

[21] Grafana Labs. [n. d.] https://grafana.com/docs/opentelemetry/docker-lgtm/.

[22] Martin Hargreaves, Thomas Hardjono, Rafael Belchior, Venkatraman Ramakrishna, and Alexandru Chiriac. 2025. Secure Asset Transfer Protocol (SATP) Core. Internet-Draft draft-ietf-satp-core-11. Work in Progress. Internet Engineering Task Force, (Aug. 2025). 47 pp. https://datatracker.ietf.org/doc/draft-ietf-satp-core/11/.

[23] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. 2009. Towards a theory of accountability and audit. In *Computer Security – ESORICS 2009*. Michael Backes and Peng Ning, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 152–167. ISBN: 978-3-642-04444-1.

[24] Suman Karumuri, Franco Solleza, Stan Zdonik, and Nesime Tatbul. 2021. Towards observability data management at scale. *SIGMOD Rec.*, 49, 4, (Mar. 2021), 18–23. doi:10.1145/3456859.3456863.

[25] Scott Keaney and Pierre Berthon. 2025. The blockchain trust paradox: engineered trust vs. experienced trust in decentralized systems. *Information*, 16, (Sept. 2025), 801. doi:10.3390/info16090801.

[26] Ningran Li, Minfeng Qi, Zhiyu Xu, Xiaogang Zhu, Wei Zhou, Sheng Wen, and Yang Xiang. 2024. Blockchain cross-chain bridge security: challenges, solutions, and future outlook. *Distributed Ledger Technologies: Research and Practice*, 4, (Oct. 2024). doi:10.1145/3696429.

[27] Charity Majors, Liz Fong-Jones, and George Miranda. 2022. *Observability Engineering*. O'Reilly Media.

[28] M. Mansouri-Samani and M. Sloman. 1993. Monitoring distributed systems. *Netwrk. Mag. of Global Internetwkg.*, 7, 6, (Nov. 1993), 20–30. doi:10.1109/65.244791.

[29] A. Mounji, B. Le Charlier, D. Zampunieris, and N. Habra. 1995. Distributed audit trail analysis. In *Proceedings of the Symposium on Network and Distributed System Security*, 102–112. doi:10.1109/NDSS.1995.390641.

[30] Rodolfo Picoreti, Alexandre Pereira do Carmo, Felippe Mendonça de Queiroz, Anilton Salles Garcia, Raquel Frizera Vassallo, and Dimitra Simeonidou. 2018. Multilevel observability in cloud orchestration. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 776–784. doi:10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00134.

[31] Monika Steidl, Benedikt Dornauer, Michael Felderer, Rudolf Ramler, Mircea-Cristian Racasan, and Marko Gattringer. 2024. How industry tackles anomalies during runtime: approaches and key monitoring parameters. In *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, (Aug. 2024), 364–372. doi:10.1109/seaa64295.2024.00062.

[32] Chainalysis Team. 2025. Lessons from the wormhole exploit: smart contract vulnerabilities introduce risk; blockchains' transparency makes it hard for bad actors to cash out. en-US. (June 2025). https://www.chainalysis.com/blog/wormhole-hack-february-2022/.

[33] Ronin Network Team. [n. d.] https://x.com/ronin_network/status/1820804772917588339.

[34] Dominique Toupin. 2011. Using tracing to diagnose or monitor systems. *IEEE Software*, 28, 1, 87–91. doi:10.1109/MS.2011.20.

[35] Natkamon Tovanich, Nicolas Heulot, Jean-Daniel Fekete, and Petra Isenberg. 2021. Visualization of blockchain data: a systematic review. *IEEE Transactions on Visualization and Computer Graphics*, 27, 7, 3135–3152. doi:10.1109/TVCG.2019.2963018.

[36] Jiajing Wu, Kaixin Lin, Dan Lin, Bozhao Zhang, Zhiying Wu, and Jianzhong Su. 2025. Safeguarding blockchain ecosystem: understanding and detecting attack transactions on cross-chain bridges. In *Proceedings of the ACM Web Conference 2025*. ACM, (Apr. 2025), 4902–4912. doi:10.1145/3696410.3714604.

[37] Yulun Wu, Guangba Yu, Zhihan Jiang, Yichen Li, and Michael R Lyu. 2025. Trace sampling 2.0: code knowledge enhanced span-level sampling for distributed tracing. *arXiv preprint arXiv:2509.13852*.