Towards Verifiable Network Telemetry without Special Purpose Hardware

Jaechan An, Zeying Zhu, Ian Miers, Zaoxing Liu
University of Maryland
College Park, MD, USA

Abstract

Verifiable network telemetry is crucial for ensuring transparency and trust in network measurements. However, telemetry logs (e.g., NetFlow records) often contain sensitive data, making public verification challenging. Recent work has attempted to address this problem using Trusted Execution Environments (TEEs), such as Intel SGX, to provide confidentiality and integrity guarantees. However, TEEs are known to suffer from complex deployment requirements and limited scalability. In this paper, we introduce a software-based approach utilizing the latest advances in Zero-knowledge Proofs (ZKPs) to enable verifiable network telemetry without revealing the underlying sensitive logs or relying on special-purpose hardware. Our system employs a general-purpose ZKP virtual machine (RISC Zero) to generate cryptographic proofs over NetFlow data, enabling operators to securely attest to network flow metrics. Our preliminary results indicate that our ZKP-based design offers a viable path toward overcoming deployment and scalability limitations inherent in the solutions that require special-purpose hardware.

CCS Concepts

• **Networks** → **Network measurement**; • **Theory of computation** → **Cryptographic protocols**.

Keywords

Network Telemetry, Verifiable Measurement, Zero Knowledge Proof

ACM Reference Format:
Jaechan An, Zeying Zhu, Ian Miers, Zaoxing Liu. 2025. Towards Verifiable Network Telemetry without Special Purpose Hardware. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3772356.3772392

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772392

1 Introduction

The End-to-End Principle (E2EP) anticipates that the Internet should function not only as an interconnecting platform but also as an economically neutral one. While this well-known principle suggests that the Internet Protocol layer should refrain from unfairly discriminating among packets based on their higher-layer protocols, there is nothing preventing the Internet Service Providers (ISPs), or the private networks of the hyperscalers, from discriminating against others and specific customers [7].

In this paper, we revisit the problem of ***how to make network performance more verifiable to users and regulators*** [4, 19, 22]. This problem requires a solution to conduct accurate and trustworthy network telemetry for operators, regulators, and end-users to enforce service-level agreements, optimize infrastructure, and conduct reliable security and compliance audits. As networks grow more complex and dynamic, *verifiable network telemetry* has become essential, not just for internal visibility, but for ensuring external accountability. For instance, verifiable network telemetry can be used as a credible verification of performance metrics reported by operators.

Verifiable network telemetry faces two interrelated challenges: *(C1) how to ensure the integrity and authenticity of collected telemetry logs, and (C2) how to preserve the confidentiality of sensitive telemetry data.* Unlike traditional data integrity problems where digital signatures and fingerprints can be useful, in the verifiable telemetry setting, operators do not want to reveal their full telemetry logs (e.g., NetFlow [9] and sFlow [20] records) in the first place for privacy and business considerations: logs often contain proprietary information such as internal policies or user activity patterns. At the same time, any meaningful verification requires the assurance that the reported metrics are based on authentic and untampered logs.

To date, recent work [8] has made attempts to achieve verifiable network telemetry with sketching algorithms [3, 13, 15, 16, 23, 25] on end hosts by leveraging hardware roots of trust, such as Trusted Execution Environments (TEEs). In these approaches, TEEs serve as "secure enclaves" that execute telemetry algorithms with runtime guarantees on the integrity of both code and data. They can also protect the confidentiality of the telemetry logs at the point of data

collection. However, adopting TEEs in practice is limited by complex deployment requirements and restricted scalability. In particular, TEE-based telemetry requires deploying TEEs on every vantage point to collect telemetry data, which may be infeasible in large or heterogeneous environments. Ideally, we want a solution that addresses both C1 and C2 in verifiable network telemetry without having to deploy special-purpose hardware in the network.

In this paper, we propose a software-based solution that ensures both confidentiality and verifiable integrity for network telemetry. Our system ensures integrity by requiring network operators to periodically commit to collected telemetry data through lightweight cryptographic hash functions. It then uses zero-knowledge proofs (ZKPs) [6, 10–12] to demonstrate that reported metrics, such as packet loss rate or flow counts, were computed correctly from the committed data, without revealing the underlying logs. Because our system is built on a general-purpose ZKP virtual machine, it supports arbitrary queries over the committed telemetry data and can use any logging or sketching algorithm. This design enables third parties to independently verify reported network behavior while preserving the confidentiality of internal telemetry records.

Our approach is applicable to various use cases which require trustworthy reporting of network behavior, including service-level agreement (SLA) verification, regulatory compliance audits, and policy enforcement cases such as network neutrality. While our prototype implementation is not yet ready for large-scale deployment, preliminary results show that it successfully enforces correctness and preserves confidentiality in telemetry computations. This demonstrates the feasibility of a purely software-based, privacy-preserving telemetry verification, representing an important step toward scalable and verifiable network telemetry without reliance on specialized hardware.

In summary, we make the following contributions:

- We present a preliminary design of a software-based verifiable network telemetry system using zero-knowledge proofs, eliminating the deployment complexity and scalability limitations of TEE-based approaches.
- We design a commit-based data integrity mechanism that allows the detection of any modification to telemetry data after collection, using cryptographic hashes and authenticated data structures.
- We support arbitrary queries over aggregated telemetry data, and generate cryptographic proofs that certify correctness without revealing raw logs.
- We decouple the aggregation phase from both logging and query processing, allowing it to be performed off-path.
- Our preliminary experiments show that while proof generation remains a performance bottleneck, it can be moved offline and off-device with small proof sizes, highlighting the potential for future deployment and optimizations.

2 Background and Motivation

In this section, we begin by discussing motivating scenarios of verifiable network telemetry. We then provide a concise overview of ZKP and explore how ZKP has been applied across various domains beyond networking. Lastly, we introduce the architecture of RISC Zero, the general-purpose ZKP framework that underpins our system.

2.1 Motivating Scenarios

Network neutrality and compliance audits. Network neutrality regulations require that ISPs and CDN operators treat traffic from all applications and content providers equitably. In practice, however, several studies have revealed that network operators sometimes throttle, prioritize, or otherwise differentiate traffic classes such as video streaming, gaming, or peer-to-peer flows, to optimize performance or favor business partners [1, 17]. Existing tools often attempt to detect unfair treatments from end-to-end measurements, but cannot localize its origin within complex Internet paths or attribute it to specific network domains. Verifiable telemetry introduces a new paradigm: regulators or public auditors could request cryptographic proofs that are constructed in zero-knowledge, demonstrating neutrality compliance. An edge operator could, for instance, prove that flows from distinct content providers exhibit statistically equivalent latency, throughput, and jitter distributions, without disclosing individual user data or proprietary network configurations. This enables transparent, privacy-preserving audits of neutrality policies that were previously infeasible.

Service-level agreements (SLAs). ISPs and CDN providers frequently establish SLAs with content providers or peering networks that specify quantitative performance guarantees, such as minimum bandwidth, latency, or packet delivery rates. Today, compliance with these agreements relies largely on private monitoring and contractual trust, leaving limited recourse in the event of disputes. When performance degradation occurs, neither party is willing to reveal raw telemetry due to business confidentiality or user privacy concerns. Verifiable telemetry resolves this tension by enabling cryptographic proofs of performance compliance. An operator can prove, for example, that at least 90% of flows achieve RTT < X ms, throughput > Y Gbps, and jitter < Z ms, satisfying the SLA requirements without exposing any underlying measurement data or revealing the structure of the network. This shifts SLA enforcement from trust-based

to proof-based, reducing disputes and fostering greater accountability in inter-network operations.

Summary. The motivating scenarios above highlight the need for verifiable network telemetry. Our goal is not to formalize what constitutes a violation of network neutrality or an SLA breach; such definitions depend on policy and contractual contexts beyond the scope of this work. Instead, we focus on enhancing the transparency and verifiability of network measurements themselves. Within this design space, zero-knowledge proofs emerge as a promising solution: they allow network operators to prove performance properties (e.g., flow statistics, latency bounds, or throughput guarantees) while keeping underlying telemetry data private. This approach bridges the gap between accountability and confidentiality, two goals that have been at odds in network operations without deploying special secure hardware inside the network.

2.2 Zero-knowledge Proofs

Zero-knowledge proofs [11] are cryptographic protocols that allow a prover to convince a verifier that a computation is correct without revealing the underlying data. Succinct Non-interactive Arguments of Knowledge (SNARK), [6], a type of zero-knowledge proof, support efficient proofs for Turing-complete computations, making them well-suited for privacy-preserving applications where the correctness of a result must be verified without exposing sensitive inputs. They have been commercially deployed in, for example, privacy-preserving payments [5]. Here, we use SNARKs for verifiable network telemetry.

As an example of a simple zk-proof, consider password authentication. A server stores a user's hashed password (Y).To authenticate, the user must show they know the password X such that $Y = hash(X)$. Instead of revealing the password, a zero-knowledge proof allows the user to provide a succinct cryptographic proof that they correctly performed the computation, thus verifying their identity while keeping the password private[1].

This principle extends directly to verifiable network telemetry, where raw network logs often contain sensitive information that cannot be revealed. A service provider can commit to these logs with a cryptographic hash, and then use a ZKP to prove that metrics such as packet loss or latency were computed correctly from the original data. This approach allows external parties to verify reported network behavior without the provider needing to reveal the sensitive logs themselves, ensuring both data privacy and accountability.

ZKPs for verifiable databases. Recent ZKP systems have enabled their use in a wide range of verifiable computation scenarios. A notable example is verifiable databases [14, 24], where ZKPs are used to prove that operations such as aggregations (e.g., SUM, AVG, $COUNT$) are executed correctly over private data, without revealing the underlying contents. Recent advances even support proofs of complete state transitions [18], allowing clients to verify that the current database state is the result of a valid sequence of operations.

However, the networking context introduces unique challenges that make direct adoption of database-style ZKP techniques insufficient. First, network telemetry systems operate at extremely high data generation rates, often collecting millions of flow records per second across distributed routers and switches. Unlike databases, where it is feasible to generate proofs per transaction or per-query, network telemetry requires a lightweight, periodic commitment model that can scale with real-time log generation. Second, network logs are typically ephemeral. Due to storage and privacy constraints, raw logs are often discarded after a period of time. This demands ZKP designs that allow for continuous integration based verification over committed summaries rather than on raw, persistent data. Third, routers and middleboxes generating telemetry are resource-constrained and not suited to perform expensive cryptographic operations. This necessitates proof generation to be offloaded to an external verifier or collector, which is uncommon in database-style settings.

While our work does not address every design challenge introduced above, it focuses on two key aspects: data integrity and scalable proof generation. To this end, our system employs lightweight per-router hash commitments to ensure the integrity of locally collected telemetry data and uses a centralized aggregation mechanism built upon Merkle trees to verify computations across routers. Proof generation is performed on an off-path compute environment, decoupled from the data collection process, enabling scalability without burdening network infrastructure. The complete system architecture is described in Section 4.

To the best of our knowledge, this is the first ZKP-based approach to verifiable network telemetry. This ZKP-driven design offers two key advantages: (1) the entire telemetry system operates purely in software, eliminating the need to access trusted hardware and thereby simplifying deployment in practice; and (2) third party verifiers can validate telemetry results without accessing the underlying sensitive logs as only the query results are exposed. This combination of software-based privacy and cryptographic verifiability enables both public accountability and scalable deployment.

2.3 RISC Zero and the zkVM Model

RISC Zero [21] is a ZKP framework designed to enable verifiable execution of arbitrary programs in a general-purpose programmable environment. It allows code written in a high-level language (i.e., Rust) to be compiled into a RISC-V binary,

[1]Assuming the password is strong enough to not be brute forced.

which is then executed inside a zkVM (zero-knowledge virtual machine). The zkVM produces a succinct, cryptographic proof that certifies the correct execution of the program on specific inputs without revealing the inputs themselves.

The architecture consists of two main components, both implemented as conventional Rust programs:

- The **host**, which prepares the inputs, coordinates proof generation, and processes the output.
- The **guest**, which runs inside the zkVM and contains the computation logic to be proven.

To illustrate how RISC Zero works, consider the hash example from Section 2.2. The host passes a batch of telemetry logs into the zkVM, where the guest program computes the hash as it would in conventional Rust. The zkVM then outputs both the result (Y) and a zero-knowledge proof attesting that $Y = hash(X)$ was computed correctly, without revealing X. This structure extends naturally to more complex operations like Merkle verification and aggregate computations.

This design makes RISC Zero particularly appealing for prototyping verifiable systems, as developers do not need to manually design ZKP circuits. In our system, the guest code is responsible for parsing committed NetFlow records, checking their inclusion using Merkle proofs, and computing performance metrics such as packet loss or throughput. The zkVM then produces a ZKP that the computation was performed correctly over authentic, unmodified data. This enables ISPs to provide cryptographic attestations over network telemetry metrics without exposing raw NetFlow records.

3 Threat Model

We assume that the embedded NetFlow program in each router hardware are generating telemetry logs faithfully at the time of packet observation. This assumption is consistent with many network monitoring use cases, where raw NetFlow records are used for operational and diagnostic purposes. However, we do not assume that the records remain trustworthy after their initial generation. For example, a malicious service provider may attempt to retroactively modify logs in order to misrepresent network performance metrics or conceal policy violations.

Previous efforts have attempted to address this challenge by running the telemetry algorithms inside the TEEs, which guarantee the integrity of telemetry computation at the point of capture. While such solutions offer strong guarantees, they require specialized hardware and introduce substantial complexity in deployment, requiring a third party to directly access network operator's infrastructure.

In contrast, our system adopts a software-based alternative. We require service providers to periodically commit to their raw logs by computing a cryptographic hash over the data in each router. These hash commitments are published periodically and serve as tamper-evident attestations, where any modification to the logs after commitment will result in a hash mismatch. During aggregation, the system recomputes the hashes over the raw logs and checks them against the published commitments to ensure that only unaltered data is used in the computation. This lightweight mechanism ensures data integrity of the raw logs without heavy computational overhead.

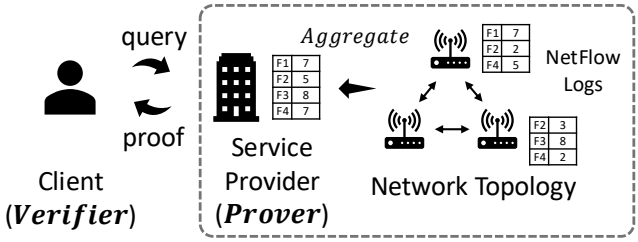

Figure 1: Overall Architecture

4 Preliminary Architecture

This section describes the overall architecture of our system for verifiable network telemetry using ZKP. We consider a typical network topology that involves multiple routers deployed across its system. Each router locally generates conventional telemetry records (e.g., NetFlow entries), capturing statistics about observed traffic such as 5-tuples, in and out bytes, and timestamps. For convenience, we refer to the raw telemetry records as *RLogs* (short for raw logs), and to the aggregated results produced by our system as *CLogs* (short for combined logs).

As illustrated in Figure 1, our system consists of a Prover and a Verifier. Routers periodically commit to their NetFlow records (RLogs) and publish the hashes. The service provider (*Prover*) collects RLogs from routers within its network topology and aggregates them into a unified dataset (CLogs) based on a predefined aggregation policy. For instance, packet loss counts from each router for the same flows can be summed to produce a total loss count per flow. Each time it aggregates the new RLogs into its CLog, the service provider creates a zero-knowledge proof that the updated aggregation was correct. Clients (*Verifier*) may then issue queries over the CLogs, and the provider responds with a cryptographic proof that the result was computed correctly over authentic, committed data. By verifying both the query proof and the aggregate proof, clients are convinced the query was correctly run on a genuine aggregate log that accurately reflects what the routers logged and announced hashes of. Importantly, the proof does not reveal any RLogs nor CLogs, preserving the confidentiality of internal telemetry.

The aggregation phase is decoupled from query processing and runs independently in the background. This allows it to be scaled according to the available resources of the provider.

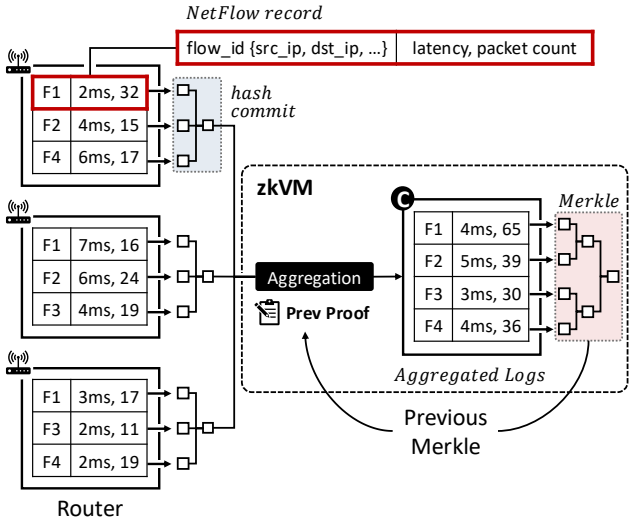

Figure 2: Aggregate log structure

Queries operate solely on the most recently committed aggregation results (CLogs), and the system can support arbitrary computation over these results within the ZKP framework.

4.1 Aggregation

We leverage Merkle tree construction as a key component in the aggregation phase. A Merkle tree (depicted as *Merkle* in Figure 2) is a cryptographic data structure used to ensure data integrity. It is organized as a binary tree in which each leaf node stores the hash of an individual data block (in our system, a CLog entry), and each internal node contains the hash of the concatenation of its two child nodes. The root of the tree, known as the Merkle root, acts as a compact cryptographic commitment to the entire dataset. To verify that a particular data entry is part of the committed dataset, a Merkle proof is constructed using the hashes along the path from the leaf node to the root. A verifier can use this proof to recompute the root and confirm it matches the original, thereby detecting any tampering. Merkle tree structure enables efficient tamper-evident verification of data integrity, playing an essential role in our aggregation logic.

Figure 2 describes the aggregation process and Algorithm 1 illustrates the actual procedure. Aggregation is performed periodically and serves as a core component to maintain an up-to-date and verifiable global dataset (CLogs) over NetFlow entries (RLogs). Each aggregation round involves three critical checks to ensure both data integrity and correctness of computation. First, the system checks the validity of the previous aggregation step by verifying its associated proof (Figure 2 – *Prev Proof*), thereby maintaining a consistent and trusted chain of computations (lines 1-4). Second, to guarantee that only authentic data is aggregated, the system verifies the integrity of the RLogs using the previously published hash commitments from each router (lines 5-11).

Algorithm 1 Risc0 Guest: Aggregation Procedure

Require: Raw logs $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$ from n routers
Require: Published hash commitments $H_1, H_2, \ldots, H_n$
Require: Previous Merkle tree T_{prev} and proof π_{prev}
Ensure: Aggregated dataset C and Merkle tree T_{new} with new proof π_{new}

1: // **Step 1: Verify Previous Aggregation**
2: **if** $\neg$`VerifyProof`(π_{prev}) **then**
3: **abort** // invalid previous proof
4: **end if**
5: // **Step 2: Verify Authenticity of Raw Logs**
6: **for** $i = 1$ to n **do**
7: Compute $H_i' \leftarrow$ `Hash`$(\mathcal{R}_i)$
8: **if** $H_i' \neq H_i$ **then**
9: **abort** // integrity check for $\mathcal{R}_i$ failed
10: **end if**
11: **end for**
12: // **Step 3: Verify, Aggregate, and Update Merkle Tree**
13: **for all** new entry r_{new} in $\mathcal{R}_i$ **do**
14: $f \leftarrow$ `FlowID`(r_{new})
15: **if** $f \in C_{prev}$ **then**
16: **if** $\neg$`VerifyMerkle`(T_{prev}, f) **then**
17: **abort** // integrity check for C_{prev} failed
18: **end if**
19: $C[f] \leftarrow C[f] + r_{new}$ ▷ Aggregate (e.g., sum)
20: **else**
21: $C[f] \leftarrow r_{new}$
22: **end if**
23: **end for**
24: $\pi_{new} \leftarrow$ `ProveAggregation`$(\mathcal{R}_1, \ldots, \mathcal{R}_n, C, T_{new})$
25: $T_{new} \leftarrow$ `UpdateMerkleTree`(C)
26: **return** (C, T_{new}, π_{new})

This ensures that no tampering has occurred since the logs were generated. Third, the system updates the CLogs and the Merkle tree over the modified and newly inserted CLog entries (line 25). Before updating, it uses the prior Merkle tree to validate the integrity of existing entries, ensuring that only verified CLogs are modified (lines 16-18).

The resulting new Merkle tree serves two purposes: it acts as the basis for integrity checking during (1) query proof generation, and (2) next round of aggregation proof generation. This periodic design ensures that both historical and future aggregation operations remain verifiable and tamper-evident.

4.2 Query

Once aggregation is completed and the new dataset has been committed as a Merkle tree, clients may issue queries to the service provider to verify specific performance metrics or

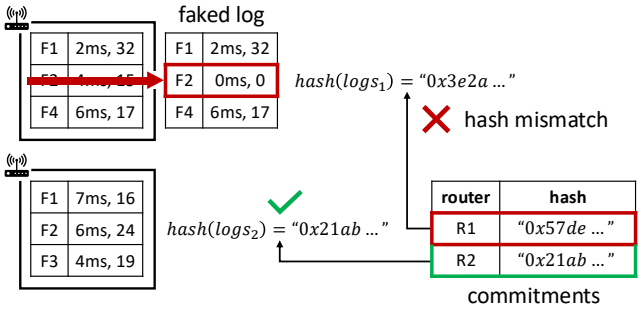

Figure 3: Integrity Check

policies over the aggregated data. These queries can range from simple statistics such as total packet loss for a given flow to more complex conditions involving filtering and computation across multiple flows. In principle, the system supports arbitrary queries over the aggregated dataset.

To respond to a query, the prover first verifies that the query is being executed against a valid aggregated state by referencing the aggregation proof. Then it checks the integrity of the relevant CLog entries using Merkle inclusion proofs. Finally, it executes the query logic over the verified data and generates a proof of correct computation.

This proof is generated inside the zkVM and guarantees two properties:

- The computation required for the query output was executed correctly.
- The data used in the computation matches the committed entries in the aggregated Merkle tree.

The client receives both the query result and the proof, and can verify them locally without needing access to the raw NetFlow records. Because the proof enforces both computational correctness and data integrity, clients gain strong guarantees that the result faithfully reflects the committed telemetry state at the time of aggregation.

This query mechanism enables a wide range of auditing and monitoring scenarios, including those required for policy enforcement, SLA compliance, and anomaly detection, without exposing sensitive network data.

5 Security Analysis

We now revisit our threat model and analyze how our system defends against malicious behavior.

First, note that query responses cannot be forged directly, as they are accompanied by zero-knowledge proofs that attest to the correctness of the underlying computation. These proofs are cryptographically sound, meaning that a forged proof for an incorrect computation cannot be constructed without breaking underlying hardness assumptions.

The more subtle threat lies in the potential manipulation of log entries to influence query results. Our system mitigates this risk through a combination of timely hash commitments and authenticated data structures:

- **Per-router hash commitments.** Each router periodically publishes a cryptographic hash over its local NetFlow logs. Any modification to the logs after publication will result in a hash mismatch, as shown in Figure 3, which is detectable during proof generation.
- **Aggregated logs with ZKP.** We combine the logs into an aggregate log, and a ZKP is generated to certify that the aggregation was performed correctly.
- **Authenticated queries over Merkle trees.** We maintain a Merkle tree of the aggregated logs, which is used as an authentication data structure that ensures data integrity for all queries (and indeed, the aggregation operations).

In summary, even a single post-commitment modification to a log entry causes a mismatch in the hash commitments or break Merkle inclusion consistency—both of which invalidate the generated proofs and expose adversarial behavior.

6 Preliminary Evaluation

We evaluate our system in a controlled environment using a custom-built NetFlow simulator that emulates a simplified network topology setting on a single machine. The simulated setting comprises 4 routers, each generating NetFlow telemetry logs in parallel via dedicated threads. These logs are written to a shared PostgreSQL backend, and each router periodically commits a cryptographic hash of its log data every 5 seconds to model a realistic integrity window. This commitment is later used to validate the authenticity of the raw logs during aggregation and query operations.

All experiments are conducted on a server equipped with an AMD Ryzen Threadripper PRO 5955WX (16 cores), 64 GiB of DDR4 memory (Samsung M393A8K40B22-CAE @3200 MHz), 2 TiB NVMe SSD, and PostgreSQL version 12.22. We use RISC Zero version 3.0 as the ZKP backend for all proof generation and verification.

End-to-end query verification. We begin by measuring the overhead of generating a query proof over aggregated telemetry data. Queries operate over the CLogs and apply filtering and compute logic similar to the following SQL:

```
SELECT SUM(hop_count) FROM clogs
WHERE src_ip = "1.1.1.1" AND dst_ip = "9.9.9.9";
```

Figure 4 illustrates the latency of aggregation and query proof generation across different query sizes. The increase in latency correlates with input size, primarily due to the computational cost of Merkle tree construction within the zkVM. However, verification remains lightweight, completing in 3 ms regardless of the number of entries. We also simulated a data tampering scenario as described in Section 5, and confirmed that any attempt to modify committed data

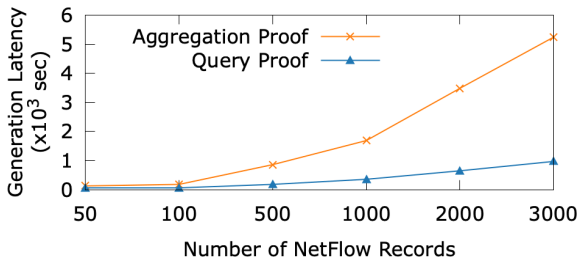

Figure 4: Proof Generation Latency

results in failed proof generation due to hash mismatches or Merkle inconsistencies.

Next, we evaluate the performance of our aggregation step, which combines raw NetFlow logs into verifiable CLogs and updates the authenticated Merkle tree. As shown in Figure 4, the generation time for the aggregation proof increases with the number of raw NetFlow entries, reaching approximately 87 min for 3,000 entries. Query proof generation shows a similar trend, taking about 16 minutes at the same scale. Profiling with RISC Zero indicates that the majority of this overhead stems from Merkle tree updates performed within the zkVM. We discuss potential improvements and optimizations in Section 7.

Proof and verification. Table 1 summarizes the size of proofs, journals (i.e., public output), and receipts for aggregation on different dataset sizes. Proof sizes remain constant (256 bytes), as expected from zk-SNARKs, while the journal and receipt sizes grow with the number of entries. Verification time remains consistently low (3 ms), validating the practicality of lightweight client-side verification. The query proof and verification show similar behavior.

7 Discussions

Our prototype demonstrates the feasibility of verifiable network telemetry based on ZKPs. While the current system is primarily a proof of concept, several directions can further improve its scalability, performance, and practical applicability in real-world deployments.

Query complexity. While our ZKP framework is general-purpose and in principle supports arbitrary queries, the cost of proof generation increases with query complexity. Queries involving heavy iteration, sorting, or non-linear operations may incur significant overhead. Exploring efficient support for more complex query types remains an important direction for future work.

Proof parallelization. ZKP generation in our system can be parallelized by dividing the workload into smaller, independent segments. For example, NetFlow entries can be partitioned by flow ID or router ID, with separate proofs generated in parallel. These partial proofs can then be merged into a single final proof, reducing end-to-end latency and leveraging multicore architectures more effectively.

# of records	Proof (bytes)	Journal (KB)	Receipt (KB)
50	256	3.6	7.6
100	256	5.6	12
500	256	29.3	58
1000	256	58.9	116
2000	256	118.1	231
3000	256	176.7	346

Table 1: Proof Size of Aggregation

GPU acceleration. Since proof generation and aggregation dominate runtime, hardware acceleration presents a natural optimization opportunity. The RISC Zero framework [21] supports GPU acceleration, and preliminary benchmarks suggest that GPU-assisted hashing and modular arithmetic can yield order-of-magnitude improvements.

Specialization proof systems. We implemented both our query logic and aggregation logic in a zkVM. Aggregation proofs can be significantly sped up, by removing the machinery for arbitrary code execution and memory in a zkVM and by switching to more specialized proof systems. For instance, the work of [2] offers 600,000 hashes per second on an M3 MacBook Pro. Since aggregating 3,000 NetFlow records in a Merkle tree of depth 11 requires $\approx$ 35,000 hashes, this would offer a substantial improvement over our current running time of 87 minutes for aggregating 3,000 entries.

Off-path computation. The off-path property of aggregation opens the door for deploying powerful external compute nodes to handle heavy cryptographic workloads without impacting the network environment.

8 Conclusions

In this paper, we presented a ZKP-based approach for verifiable network telemetry that addresses both data integrity and confidentiality without relying on specialized hardware. By combining cryptographic commitments with zero-knowledge proofs, our system allows third parties to verify reported performance metrics without exposing sensitive logs. Built atop a general-purpose ZKP framework, our design supports arbitrary queries and demonstrates practical feasibility through a working prototype. While current performance limitations prevent immediate real-world deployment, our results highlight a promising direction toward scalable and privacy-preserving network accountability.

9 Acknowledgments

We thank the anonymous reviewers and our shepherd for their constructive comments and feedback. This work was supported in part by the U.S. NSF grants CNS-2431093 and SaTC-2415754.

References

[1] 2014. T-Mobile forced to stop hiding slow speeds from throttled customers. https://arstechnica.com/information-technology/2014/11/t-mobile-forced-to-stop-hiding-slow-speeds-from-throttled-customers/.

[2] 2025. StarkWare sets new proving record. https://starkware.co/blog/starkware-new-proving-record/

[3] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. {HeteroSketch}: Coordinating network-wide monitoring in heterogeneous and dynamic networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 719–741.

[4] Katerina Argyraki, Petros Maniatis, and Ankit Singla. 2010. Verifiable network-performance measurements. In *Proceedings of the 6th International COnference*. 1–12.

[5] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, May 18-21, 2014*. 459–474.

[6] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. In *34th Annual Cryptology Conference (Crypto) (Lecture Notes in Computer Science, Vol. 8617)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, 276–294.

[7] Lloyd Brown, Emily Marx, Dev Bali, Emmanuel Amaro, Debnil Sur, Ezra Kissel, Inder Monga, Ethan Katz-Bassett, Arvind Krishnamurthy, James McCauley, et al. 2024. An Architecture For Edge Networking Services. In *Proceedings of the ACM SIGCOMM Conference*. 645–660.

[8] Zhuo Cheng, Maria Apostolaki, Zaoxing Liu, and Vyas Sekar. 2024. Trustsketch: Trustworthy sketch-based telemetry on cloud hosts. In *The Network and Distributed System Security Symposium (NDSS)*.

[9] Benoit Claise. 2004. *Cisco systems netflow services export version 9*. Technical Report.

[10] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *2nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt) (Lecture Notes in Computer Science, Vol. 7881)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer, 626–645.

[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, Robert Sedgewick (Ed.). ACM, 291–304. doi:10.1145/22145.22178

[12] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt) (Lecture Notes in Computer Science, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, 305–326.

[13] Nikita Ivkin, Ran Ben Basat, Zaoxing Liu, Gil Einziger, Roy Friedman, and Vladimir Braverman. 2019. I know what you did last summer: Network monitoring using interval queries. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2019), 1–28.

[14] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. ZKSQL: Verifiable and Efficient Query Evaluation with Zero-Knowledge Proofs. *Proc. VLDB Endow.* 16, 8 (April 2023), 1804–1816. doi:10.14778/3594512.3594513

[15] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM SIGCOMM Conference*. 334–350.

[16] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the ACM SIGCOMM Conference*. 101–114.

[17] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. 2015. Identifying traffic differentiation in mobile networks. In *Proceedings of the 2015 Internet Measurement Conference*. 239–251.

[18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. 2018. Proving the correct execution of concurrent services in zero-knowledge. Cryptology ePrint Archive, Paper 2018/907. https://eprint.iacr.org/2018/907

[19] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. 2009. Detecting network neutrality violations with causal inference. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. 289–300.

[20] Mea Wang, Baochun Li, and Zongpeng Li. 2004. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 628–635.

[21] RISC Zero. [n. d.]. RISC Zero. https://risczero.com/ year =2023.

[22] Xin Zhang, Zongwei Zhou, Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Adrian Perrig, and Patrick Tague. 2012. Shortmac: efficient data-plane fault localization.. In *The Network and Distributed System Security Symposium (NDSS)*.

[23] Yinda Zhang, Peiqing Chen, and Zaoxing Liu. 2024. OctoSketch: Enabling Real-Time, Continuous Network Monitoring over Multiple Cores.. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[24] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology ePrint Archive, Paper 2017/1146. https://eprint.iacr.org/2017/1146

[25] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the ACM SIGCOMM Conference*. 207–222.