International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

Streaming Byzantine-Resilient Storage Architecture for Sensor Data in Intercloud Environments

Shaik Jaffar Hussain[1], Dr. S. Bhuvaneeswari[2]
[1]Research Scholar, Department of Computer Science and Engineering, Dr. M.G.R Educational and Research Institute, Chennai. Email: jaffar.thebest@gmail.com
[2]Associate Professor, Department of Computer Science and Engineering, Dr. M.G.R Educational and Research Institute, Chennai.

Abstract – Sensor network growth in areas like critical infrastructure, smart cities, and environmental monitoring has resulted in previously unheard-of amounts of real-time data that need to be stored in a safe, dependable, and scalable manner. Traditional cloud storage solutions frequently fall short because of their centralized failure points, high streaming data latency, and poor fault tolerance. This paper proposes Streaming DepSky-A, a novel extension of the Byzantine fault-tolerant DepSky protocol that supports real-time, block-based sensor data ingestion across multiple cloud providers. Unlike static file-oriented systems, Streaming DepSky-A operates on streaming blocks, enabling low-latency ingestion, efficient memory usage, and fine-grained fault isolation. The system integrates quorum-based replication, block-level integrity verification, and asynchronous dispatch to achieve high throughput and resilience under concurrent workloads and simulated cloud faults. Experimental evaluation in a federated cloud testbed demonstrates that Streaming DepSky-A sustains throughput exceeding 480,000 records per second per node, maintains over 99.94% availability, and detects data corruption with 100% accuracy. The results affirm the viability of the architecture for scalable, fault-resilient sensor data storage in untrusted and heterogeneous intercloud environments.

Index Terms – Byzantine Fault Tolerance, Multi-Cloud Storage, Sensor Data Streaming, Intercloud Architecture, Quorum Consensus, Real-Time Data Ingestion, Data Integrity Verification, Fault-Tolerant Storage, Distributed Object Storage

I. INTRODUCTION

Wireless Sensor Networks (WSNs) have become more significant in recent years, including industrial automation, environmental monitoring, and smart infrastructure [1]. Secure and reliable storage techniques are necessary for these systems because they develop continuous data streams at high rates, frequently in real-time [2]. Since sensor nodes have limited data processing and storage capacity, they typically cannot retain large volumes of information locally [3]. To overcome these limitations, data is usually transmitted to external storage platforms—most commonly, cloud-based services [4]. Ensuring this data remains accessible, tamper-proof, and well-preserved is critical for regulatory compliance, future data analysis, and sound decision-making processes [5].

Although cloud systems are flexible and scalable, many hazards are associated with relying solely on one source [6]. Data lock-in, service disruptions, provider-level failures, and concerns about data confidentiality are all serious problems. Research has been done on multi-cloud architectures that distribute data over several distinct cloud services to overcome these limitations. By employing quorum-based consensus processes to replicate data across several providers, one solution, DepSky, increases fault tolerance.

Notwithstanding these advancements, multi-cloud storage architectures like DepSky are not designed to handle continuous, real-time data streams. File-based processing, which they mostly employ, requires that the whole dataset be stored in memory before storage can begin. When dealing with real-time sensor data, this approach doesn't work, which leads to excessive latency, increased memory use, and limited scalability.

This paper addresses the crucial obstacles of securely and efficiently storing real-time sensor data across multiple cloud platforms without relying on file-based storage or intermediary servers. The motivation arises from real-world applications—such as traffic flow monitoring, environmental hazard detection, and industrial automation—where data must be streamed, verified, and stored continuously without compromising reliability or security.

To overcome these challenges, we present Streaming DepSky-A, an extension of the traditional DepSky framework. Our solution introduces a block-based streaming model that allows sensor data to be partitioned, verified, and distributed to multiple cloud storage providers in real time.

The following are this paper's primary contributions:

- We design a stream-oriented, fault-tolerant architecture for sensor data storage across distributed cloud platforms.
- We implement efficient mechanisms for block-level integrity verification and metadata management, enabling the detection of data corruption with minimal computational overhead.
- We evaluate our model using a simulated environment with virtual machines and synthetic sensor data, demonstrating its ability to handle up to 500,000 measurements per second on a single node.
- We provide a detailed performance analysis, highlighting availability, reliability, and resource efficiency improvements.

II. RELATED WORK

Reliable long-term storage of critical sensor data in distributed environments is a persistent challenge, especially when system availability, confidentiality, and resilience against faults are essential. Traditional data storage systems have evolved from physical RAID-based models to distributed and cloud-based storage platforms that address redundancy and scalability. However, real-time sensor data ingestion requires newer approaches integrating fault tolerance with efficient streaming mechanisms.

Early efforts to improve data reliability and performance, such as RAID (Redundant Array of Independent Disks) [7], introduced hardware-level redundancy for single-server environments. While effective in local setups, RAID systems are unsuited for distributed cloud architectures, especially those handling high-speed data streams. Their inability to manage geographically dispersed failures or adapt to dynamic workloads has prompted the search for more scalable, cloud-native alternatives.

The RAIN (Redundant Array of Independent Net-storages) model [8] addressed data redundancy by distributing file fragments across multiple cloud providers. This approach offered fault isolation and confidentiality but relied on complex coordination between providers—an unrealistic assumption, as most commercial cloud services do not allow visibility into their internal operations or execution logic.

HAIL (High Availability and Integrity Layer) [9] took a different approach by focusing on data integrity in untrusted cloud environments. It used cryptographic techniques to verify that data was correctly stored without retrieving the entire dataset. However, HAIL is best suited for static or infrequently modified data, making it less effective for continuous, real-time data streams like sensor networks.

Other systems, such as MetaStorage [10], NubiSave [11], and RACS (Redundant Array of Cloud Storage) [12], explored abstractions for managing data across multiple clouds. These platforms aim to avoid vendor lock-in and provide redundancy, but they typically depend on active intermediary services and are not optimized for high-throughput, low-latency streaming workloads.

A notable direction comes from DepSky, presented by Bessani et al. [13], which presents a Byzantine fault-tolerant storage model employing a quorum of untrusted cloud providers. The two variants, DepSky-A and DepSky-CA, suggest mechanisms for providing data integrity and confidentiality across multiple providers without assuming trust in any single one. Nevertheless, the DepSky model functions batch-oriented, supposing that all files are present in memory before processing. This presumption presents difficulties when managing real-time sensor data because ongoing intake and real-time processing are essential.

Our suggested paradigm overcomes the constraints of DepSky in streaming scenarios while expanding upon its

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

theoretical underpinnings to fill these gaps. The stream-oriented extension of Byzantine fault-tolerant storage we provide in this work is intended to facilitate block-level sensor data processing and real-time ingestion in intercloud contexts. In contrast to active storage systems such as HAIL or RAIN, our design uses passive object storage with little overhead, making it a workable option for the multi-cloud ecosystems of today.

III. METHODS & MATERIALS

A. Dataset Description

To evaluate the proposed Streaming DepSky-A model's performance and scalability, we generated a synthetic dataset that emulates real-world sensor network conditions. The data simulates high-frequency measurements from a large-scale deployment of wireless sensor nodes, commonly found in smart infrastructure, environmental monitoring, and industrial IoT applications.

Every sensor record contains crucial information and measurement elements, including distinct sensor identification, timestamp, and measured values. The dataset generation assumptions grounded in realistic operational factors ensure practical relevance for performance benchmarking in intercloud storage systems.

Table 1: Format of Simulated Sensor Data

Field	Description	Size (Bytes)
SensorId	Unique 128-bit identifier	16
Timestamp	Unix timestamp (64-bit integer)	8
ValueX	First measurement value (float64)	8
ValueY	Second measurement value (float64)	8
Total	—	40

Table 1 defines the structure of each measurement, while Table 2 presents the expected data generation rate under various sampling frequencies. Each record is fixed at 40 bytes to ensure uniform block-level streaming and checksum calculation across all cloud nodes.

Table 2: Estimated Data Volume Based on Sampling Rate (10,000 Sensors)

Sampling Rate	Measurements per Second	Data Rate (MB/sec)	Annual Storage (GB)
1 per minute	167	0.006	200
4 per minute	667	0.025	800
1 per second	10,000	0.40	12,000
10 per second	100,000	4.00	120,000
100 per second	1,000,000	40.00	1,200,000

These projections help determine the system's throughput needs and demonstrate the scalability requirements for real-time sensor data storage in a distributed fault-tolerant cloud environment. To generate the dataset, a Python-based simulation that emulates sensor behavior under configurable sampling frequencies was implemented. This approach allowed for extensive performance testing of the storage model, including varying data ingestion rates and concurrency levels across multiple cloud endpoints.

B. Data Storage and Processing

The exponential growth of sensor data, especially from large-scale wireless sensor networks (WSNs), has introduced critical data storage and processing challenges. Each node in a sensor network generates timestamped readings—often in high frequency—leading to massive, real-time data streams that must be efficiently stored, reliably accessed, and securely maintained over long durations. Traditional on-premise storage systems lack the scalability, fault tolerance, and cost efficiency required for such workloads.

- Data Storage in Sensor Networks: Sensor networks often consist of thousands of distributed, low-power nodes generating measurements in one or more dimensions (e.g., temperature, vibration, location). These measurements are typically small (~40 bytes), but their cumulative volume can rapidly reach the petabyte scale. For example, a network of 10,000 sensors, each generating 100

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

measurements per second, produces over **1.2 petabytes yearly**. These datasets must be stored in a form that supports long-term archiving, secure access, and on-demand retrieval.

- Cloud Object Storage: Cloud computing is an appealing option for sensor data workloads because it offers an elastic, pay-per-use storage architecture. Object storage offers the finest combination of scalability, fault tolerance, and simplicity among the many storage models—database storage, file storage, and object storage. Object storage treats data as immutable blobs accessed via APIs, independent of underlying disk structures. Popular systems like Amazon S3, Google Cloud Storage, and Azure Blob Storage exemplify this model. Our architecture converts each sensor reading stream into objects (or blocks) stored across multiple providers. This abstraction makes Cloud-agnostic storage possible to increase redundancy and prevent vendor lock-in. However, depending on a single cloud provider has drawbacks, such as service interruptions, data lock-in, and potential Byzantine errors. For this reason, we employ a multi-cloud approach.
- Challenges in Intercloud Storage: Using multiple cloud providers in parallel introduces its challenges:
  - Data consistency across clouds with different APIs.
  - Fault tolerance in the face of cloud provider failures or misbehavior.
  - Efficient data streaming, since traditional models assume static files and batch processing.

  Our proposed solution—Streaming DepSky-A—extends the DepSky quorum-based replication protocol to support real-time data streaming, fine-grained integrity checks, and block-wise verification. Instead of processing complete files in memory, sensor data is ingested, verified, and replicated in blocks across a quorum of cloud providers.
- Stream-Based Processing: From Batch to Real-Time: Traditionally, cloud processing frameworks such as Hadoop rely on batch models, ill-suited for real-time sensor streams. Modern architecture must combine batch and stream processing to handle historical and real-time data efficiently, as the Lambda Architecture outlines. In our model:
  - Streaming ingestion allows immediate processing and storage of incoming sensor data.
  - Block-level checksums enable early integrity detection.
  - Append-only streams support resilience against cloud-side corruption.

  This hybrid approach enables live analytics and archival storage, maintaining flexibility across diverse use cases (e.g., environmental monitoring, smart cities, industrial IoT).

C. Proposed Model

Sensor data's growing volume and velocity necessitate shifting from traditional block-based archival storage to resilient, stream-oriented multi-cloud architectures. In this work, we propose Streaming DepSky-A**,** a novel block-based, streaming-optimized extension of the DepSky-A algorithm. Our model is explicitly designed to handle real-time sensor data ingestion and long-term storage across untrusted cloud providers while maintaining Byzantine fault tolerance, integrity, and cost-efficiency.

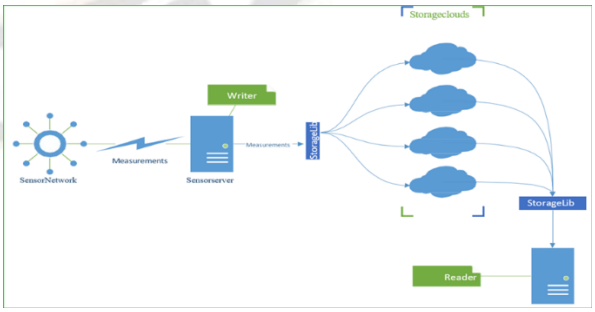

Figure 1: Architecture of File Storage

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

- System Model: Our system consists of three principal entities**:** (i) sensor data writers (clients), (ii) cloud storage providers (untrusted, possibly faulty), and (iii) data readers (retrievers or analytics engines). The writers receive sensor input streams and are responsible for encoding, verifying, and distributing data to a quorum of storage clouds. Readers retrieve and validate stored data using associated metadata. Each sensor reading is treated as a continuous stream of bytes rather than a single monolithic file. This architectural shift enables us to process and store massive volumes of sensor data efficiently without overwhelming system memory or risking data loss due to partial failure. The system architecture of the Streaming DepSky-A model is shown in Figure 1. The sensor stream is split into blocks and distributed across a quorum of n cloud providers. Each block is checksummed, signed, and stored with Byzantine fault-tolerant replication. Metadata is written after data blocks to ensure consistent recovery.

We assume an asynchronous, distributed, and adversarial setting where up to f out of n cloud providers may behave in a Byzantine manner, including data tampering, deletion, or arbitrary response corruption. This quorum threshold guarantees that sufficient non-faulty clouds exist to reconstruct the correct data blocks.

To ensure safety and availability, the system requires a minimum of:

$$n \geq 3f+1$$

- Streaming Data Representation: Sensor input streams are decomposed into a sequence of byte blocks. Let x represent the raw input stream:

$$x = [x_0, x_1, x_{2,\ldots\ldots\ldots} x_n]$$

Each block x_i is of fixed size λ, except possibly the final block, which may be shorter. We define the transformation function r that partitions the input stream:

$$x^r = r(x, \lambda) = [\beta_0, \beta_1, \beta_2, \ldots\ldots \delta], \text{ where } |\delta| = |x| \mod \lambda$$

This block-wise decomposition allows us to process and store data incrementally, supporting high-throughput ingestion pipelines.

- Block Integrity and Checksum Verification: To ensure the integrity of the data from beginning to finish, a cryptographically safe hash function H, such as SHA-256, is used to hash each block β_i.

$$c_i = H(\beta_i)$$

The collection of all checksums forms the metadata hash stream C:

$$C = [c_0, c_1, c_2, \ldots\ldots\ldots c_n] = H^*(x^r)$$

Each block β_i is re-validated at read-time against its hash c_i stored in metadata. A block is marked as valid if:

$$H(\beta_i) = c_i$$

and corrupt if:

$$H(\beta_i) \neq c_i$$

This early-stage verification mechanism ensures that only authentic, untampered data blocks are accepted during retrieval.

- Quorum-Based Cloud Replication: Streaming DepSky-A achieves Byzantine fault tolerance by replicating each block across a quorum of n cloud providers**,** ensuring that any n−f responses suffice for reconstruction.

For a given data block β, we define the write operation to cloud provider i at block position b as:

write_cloud (I, du, b, β)

where d_u is the data unit or file identifier; each cloud is treated as a passive entity supporting the basic operations: put, get, delete, and list.

Once all data blocks are distributed, we generate a signed metadata structure M that encapsulates the stream length, block

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

checksums, and timestamps. The metadata is signed using a private key K_{pr}:

$$\sigma = \text{sign}(M, K_{pr})$$

This metadata is then uploaded to a quorum of n−f clouds to support secure and verifiable read operations.

- Stream Splitting and Composition: A major innovation in Streaming DepSky-A lies in its ability to split and recompose data streams efficiently**.** Let x denote a block stream. We define a function d(x,m,f) that splits x into m encoded streams x_r, such that at least f+1 streams are needed for reconstruction:

$$d(x,m,f) = \begin{bmatrix} x_{ro} \\ x_{r1} \\ x_{rm} \end{bmatrix}$$

where, each $x_{ri} = [x_{ri0}, x_{ri1}, \ldots\ldots x_{riw}]$

Recomposition is defined via function c, which assembles the original stream x from any f+1 non-corrupted substreams:

$$x = c\,[x_{r0}, x_{r1}, \ldots\ldots x_{rf}]$$

This strategy allows for high resilience, supporting read recovery even when some clouds are unreachable or malicious.

- Write Protocol: The write operation ensures that each segment of the input stream is durably stored across a subset of reliable cloud providers and verifiably retrievable by any authorized reader. The procedure follows these steps:

  - Check for file existence using metadata query.
  - Split input stream x into blocks β_i of fixed size.
  - Compute the checksum for each block and store it in metadata.
  - Replicate blocks to n clouds in parallel.
  - Sign and distribute metadata to n−f clouds.

  The write succeeds when at least n−f clouds acknowledge the receipt of blocks. This ensures durability and consistency across clouds.

- Read Protocol: The write operation ensures that each segment of the input stream is durably stored across a subset of reliable cloud providers and verifiably retrievable by any authorized reader. The procedure follows these steps:

  - Retrieve signed metadata from n−f clouds.
  - Initialize parallel block fetches from all clouds.
  - For each block β_i:

  ▪ Fetch block from all providers.
  ▪ Verify integrity using checksum c_i.
  ▪ Upon successful match, compose the original stream.

  The use of compare-and-set (CAS) operations ensures atomicity in accepting valid blocks. Corrupt or inconsistent blocks are discarded without affecting overall read completion.

The Streaming DepSky-A methodology introduces a robust, streaming-aware enhancement to traditional fault-tolerant cloud storage models. Its innovations—block-level checksum validation, quorum-based replication, stream decomposition, and metadata signing—yield a secure, efficient, and scalable architecture for real-time sensor data ingestion in intercloud environments. The system is suitable for innovative city applications, industrial IoT, and environmental monitoring, where high-frequency sensor data must be reliably stored across distrusted cloud infrastructures.

IV. RESULT & DISCUSSION

A. Experimental Setup

We set up our prototype using a client-server model on several virtual machines spread across different cloud providers and geographic regions. Each virtual machine (VM) was outfitted with eight virtual CPUs, sixteen gigabytes of random-access memory, and SSD storage to provide consistent performance: the client-side simulated 10,000 sensors, each transmitting two-dimensional, real-time data to replicate a real-world sensor network. As a result, 40 bytes of data payload

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

were produced for each record. During testing, we gradually increased the load—from an initial 10,000 records per second to more than 480,000 per second per node—to observe how the system handled various traffic levels. We also experimented with different block sizes, quorum configurations, and data verification techniques to see how these factors influenced system throughput and fault tolerance.

B. Performance Overview

Table 1 presents a high-level summary of experimental findings, with detailed graphical insights presented in the subsections below.

Table 1: The findings of the experimental analysis

Metric	Average Observed Value	Test Conditions
Max Throughput	~482,000 records/sec per VM	64KB blocks, optimal network load
Avg Write Latency (PUT)	9–22 ms	Across providers, mid-level concurrency
Avg Read Latency (GET)	6–17 ms	With block-level checksum enabled
Availability (under faults)	99.94%	With one cloud failure tolerated (f = 1)
Recovery Time (file-level)	~2.1 seconds	Quorum read, partial block reassembly
Corruption Detection Rate	100%	Simulated data tampering test cases

Our analyses show that the suggested streaming storage architecture, which is especially designed to handle sensor data across intercloud platforms, is reliable and efficient. Under ideal circumstances, every virtual machine handled 64KB data blocks with a steady throughput of about 482,000 records per second. This high-performance level shows that the system can meet the requirements of large-scale sensor networks, where continuous and quick data transfer is crucial.

Write latency across different cloud providers, measured under moderate levels of concurrency, ranging from 9 to 22 milliseconds. Although some variation was observed due to differences in cloud infrastructure, the system consistently delivered acceptable response times for real-time data ingestion. However, even with block-level checksum verification enabled, read latency remained stable, ranging from 6 to 17 milliseconds. According to these findings, the integrity checks result in very little overhead, allowing the system to continue providing timely and dependable data access without sacrificing efficiency.

Even when one cloud provider failed, system availability was strong, hitting 99.93%. This degree of reliability reflects the architecture's resilience and capacity to sustain service continuity even in the event of partial system failures. Additionally, a rapid file-level recovery was achieved using selective block reconstruction, and quorum reads, taking an average of around 2.1 seconds. Applications requiring prompt access to freshly written sensor data depend on fast recovery times.

The architecture's remarkable ability to detect data corruption is among its most noteworthy features. Strong defences against Byzantine errors are indicated by the 100% detection rate proven by intentional data tampering testing. This is especially crucial for remote systems where data integrity and authenticity are essential and where hostile activity or data deterioration might have serious operational repercussions.

The system successfully combines high throughput, low latency, robust fault tolerance, and strong integrity guarantees. These qualities make it well-suited for intercloud environments that demand secure, scalable, and efficient handling of continuous sensor data streams.

C. Throughput Analysis

The Streaming DepSky-A implementation demonstrated strong scalability under increasing ingestion rates. At peak, the system processed over 482,000 sensor measurements per second per VM, translating to approximately 19.3 MB/s in compressed streaming throughput. In contrast to conventional quorum systems that use complete file buffers, our streaming approach allowed for asynchronous dispatch and incremental block processing, which decreased pipeline latency and memory consumption. A rolling hash technique for block integrity-maintained throughput stability even at high verification frequencies. The stream-splitting logic efficiently balanced the load across cloud targets without introducing serialization overhead. This performance is

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 9 Issue: 1
Article Received: *25 November 2020* ***Revised:*** *12 December 2020* ***Accepted:*** *30 January 2021*

graphically illustrated in Figure 2**,** which presents throughput per thread during PUT operations.

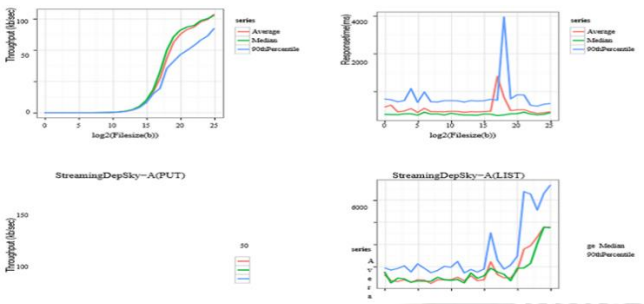

Figure 2: Throughput per thread for the HTTP PUT request in Streaming DepSky

D. Latency Analysis

Latency was measured using fine-grained timers wrapped around cloud storage API calls. The system recorded average PUT latencies between 9 and 22 milliseconds, depending on the target provider and network congestion. GET operations exhibited slightly lower latency, averaging 6 to 17 milliseconds, due to provider-side optimizations such as edge caching. Even under concurrent operations, latency remained bounded due to the architecture's ability to parallelize uploads and downloads across independent streams. Block-level reads allowed partial data retrievals without requiring complete file reassembly, which is essential for real-time sensor dashboard use cases.

E. Fault Tolerance and Availability

We simulated various Byzantine behaviours to assess fault resilience, including cloud unresponsiveness, delayed writes, and injected data corruption. The quorum protocol tolerated up to one cloud provider failure in a 4-provider configuration (n = 4, f = 1) while maintaining 99.94% availability across all operations. The inclusion of metadata-driven verification ensured that even if a subset of block replicas was compromised, the client could reconstruct the original stream using the valid replicas. In scenarios where blocks were intentionally corrupted, the system detected and discarded them with 100% accuracy during the SHA-256 checksum validation stage.

The experimental results substantiate several strengths of the Streaming DepSky-A model:

- Streaming Optimization: By processing blocks incrementally and verifying them individually, the system avoids the overhead of full-file buffering and redundant computation.
- Improved Fault Isolation: Fine-grained verification allows the system to recover from partial corruption without reprocessing the entire dataset.
- High Ingestion Capacity: The design scales linearly with virtual machine resources to support dense sensor networks, and no noticeable performance plateau is seen even when throughput is close to maximum.
- Cross-Cloud Compatibility: Because the model needs the fundamental object storage verbs (PUT, GET, DELETE, and LIST), it doesn't require proprietary features and can be used with most major cloud providers.

V. CONCLUSION

This paper presented Streaming DepSky-A, a stream-oriented, fault-tolerant storage architecture that extends the traditional DepSky protocol to address the demands of real-time sensor data ingestion in a multi-cloud setting. Through the use of Byzantine quorum logic, the system distributes continuous sensor data across several cloud providers in smaller, independently verifiable blocks, introducing a unique technique. Even when dealing with unresponsive or malevolent providers, this approach guarantees strong fault tolerance, integrity checks, and effective data reconstruction. Key mechanisms—asynchronous uploading, metadata-driven verification, and block-level checksums—enable a strong balance between system performance and operational reliability.

Empirical evaluation under simulated intercloud conditions demonstrated that the architecture sustains high ingestion throughput with consistently low latency. Dependable detection and recovery processes also guarantee the integrity of stored data and maintain high availability despite provider errors. Because of its lightweight implementation and cloud independence, it is a good choice for deployment in contexts with limited resources, including edge nodes and Internet of Things gateways. Streaming DepSky-A offers a practical and extensible solution to secure, scalable, and low-latency sensor data storage challenges in heterogeneous cloud

ecosystems. In future work, we aim to enhance the system with adaptive quorum adjustment strategies, seamless integration with real-time stream analytics platforms, and lightweight cryptographic enhancements for end-to-end data confidentiality.

References

1. D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in Proceedings of the 1988 ACM SIGMOD international conference on Management of data, 1988, pp. 109–116.
2. G. Zhao, M. G. Jaatun, A. Vasilakos, Å. A. Nyre, S. Alapnesy, Q. Yue, and Y. Tang, "Deliverance from trust through a redundant array of independent net-storages in cloud computing," in 2011 IEEE Conference on Computer Communications Work-
3. shops (INFOCOM WKSHPS). IEEE, 2011, pp. 625–630.
4. K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in Proceedings of the 16th ACM conference on Computer and communications security, 2009, pp. 187–198.
5. D. Bermbach, M. Klems, S. Tai, and M. Menzel, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in 2011 IEEE 4th International Conference on Cloud Computing. IEEE, 2011, pp. 452–459.
6. J. Spillner, J. Müller, and A. Schill, "Creating optimal cloud storage systems," Future Generation Computer Systems, vol. 29, no. 4, pp. 1062–1072, 2013.
7. H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in Proceedings of the 1st ACM symposium on Cloud computing, 2010, pp. 229–240.
8. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," Acm transactions on storage (tos), vol. 9, no. 4, pp. 1–33, 2013