IntDB: Towards a Spatiotemporal Database for In-band Network Telemetry

Lizhuang Tan[1], Xin Dong[1], Nguyen Van Tu[2], James Won-Ki Hong[3]
[1]Key Laboratory of Computing Power Network and Information Security, Ministry of Education
Shandong Computer Science Center (National Supercomputer Center in Jinan)
Qilu University of Technology (Shandong Academy of Sciences), Ji'nan, China
[2]MangoBoost, Seoul, Korea
[3]Department of Computer Science and Engineering,
Pohang University of Science and Technology, Pohang, Korea
Email: tanlzh@sdas.org, 10431240209@stu.qlu.edu.cn, tunguyen@postech.ac.kr, jwkhong@postech.ac.kr

***Abstract*—In-band Network Telemetry (INT) provides fine-grained, path-level network visibility by embedding measurement data into live packets. However, the nested, spatiotemporal nature of INT data poses significant challenges for traditional telemetry storage systems, which are primarily optimized for time series data. In this paper, we present IntDB, a spatiotemporal database for INT. IntDB introduces a path-oriented data model that preserves end-to-end flow structure and supports efficient multi-hop correlation and querying. Experimental results show that IntDB outperforms existing solution in both writing scalability and query performance, making it a practical foundation for INT-based network measurement.**

***Index Terms*—Network Measurement; In-band Network Telemetry; Spatiotemporal Data; Telemetry Database**

I. Introduction

Network measurement forms the foundation of network management. Traditional methods, such as SNMP [1] and NetFlow [2], have been extensively used in large-scale network operations [3]. With the emergence of software-defined networking (SDN) and data plane programmability, network devices like switches have gained enhanced functionality and flexibility, giving rise to a variety of network telemetry techniques [4]. As a new paradigm, network telemetry is commonly categorized into out-of-band network telemetry (ONT) [5] and in-band network telemetry (INT) [6]. ONT extends conventional protocols to collect measurement data outside the data path, whereas INT leverages programmable data planes to embed telemetry information directly within packets as they traverse the network.

As modern networks grow in scale and complexity [7], point-level measurement approaches—such as SNMP, NetFlow, and ONT—have become increasingly insufficient for comprehensive network management, particularly in diagnosing and analyzing faults [8]. In contrast, path-level measurement techniques like INT capture multi-hop, end-to-end telemetry data, enabling accurate fault localization and transmission quality assessment. These have seen practical deployment in data center networks [9] and wide-area networks [10].

However, as illustrated in Tab. I, the challenge of efficiently storing and managing massive volumes of telemetry data—particularly from path-level measurements—remains largely unresolved. Because point-level measurement has long been the dominant paradigm, existing storage architectures for network telemetry are primarily designed around time series data. Time series databases (TSDBs), exemplified by InfluxDB [11], and data stream processing platforms like Apache Kafka [12], are inherently optimized for scalar, time-indexed data streams and fall short in supporting the complex requirements of path-level telemetry.

In particular, when applied to INT, current data storage solutions represented by TSDBs exhibit the following limitations:

1) ***Data Model Misalignment.*** INT telemetry data is inherently path-oriented and consists of nested, multi-hop records. Traditional TSDBs, designed for flat, metric-centric time series, fail to represent the spatiotemporal semantics of INT data, leading to the loss of end-to-end path structure.
2) ***Inefficient Write Operations.*** INT data is typically generated in batches per flow, with non-uniform intervals and complex nested formats. This deviates from the high-frequency, scalar, and uniform data patterns that TSDBs are optimized for, resulting in degraded write throughput and poor storage efficiency.
3) ***Limited Query Expressiveness.*** TSDBs primarily support simple time-windowed queries over individual metrics, lacking native support for path-level queries such as hop-by-hop correlation, bottleneck detection, or flow reconstruction. This severely restricts their usability for INT-specific analysis tasks.

In this paper, we conduct an in-depth analysis of the data storage and query requirements of INT, and propose ***IntDB***, a telemetry database specifically designed to address the unique characteristics of INT data. IntDB introduces a path-oriented

This work was supported by the Shandong Provincial Natural Science Foundation under Grant No. ZR2023QF025, the Institute of Information & Communications Technology Planning & Evaluation (IITP) (RS-2024-00392332), the National Natural Science Foundation of China under Grant 62402257, and the Korea-China Young Scientists Exchange Program grant funded by the National Research Foundation of Korea (NRF-24109)

TABLE I: Comparison of measurement types and their corresponding storage systems

Measurement Type	**Representative Method**	**Data Characteristics**	**Representative Storage System**
Point-level	SNMP, NetFlow, ONT	Time series (per-node metrics)	InfluxDB
Path-level	INT	Spatiotemporal sequence (per-flow multi-hop metrics)	Not well supported

data model, supports spatiotemporal correlation, and enables expressive, INT-native queries over multi-hop flows. This paper makes the following key contributions:

- We identify the fundamental limitations of existing telemetry storage systems in handling in-band network telemetry, including data model misalignment, inefficient writes, and weak query expressiveness.
- We design and implement IntDB, a spatiotemporal database tailored for INT, which preserves path semantics and supports multi-hop correlation and query. We present a unified data model and query abstraction for INT that enables expressive, flow-level spatiotemporal analysis.
- We open-source IntDB[1], evaluate its performance and effectiveness, and compare it with traditional telemetry storage architectures.

II. Related Work and Motivation

As shown in Listing 1, INT is a network measurement paradigm that integrates both spatial and temporal dimensions. Existing data storage technologies fall short of meeting its requirements. TSDBs destroy the inherent path structure of INT data by flattening measurements into scalar time series, while Graph or Spatiotemporal Databases, though capable of representing relationships, often incur poor read/write performance and are not optimized for high-throughput telemetry. Furthermore, current query mechanisms for INT are fragmented and lack unified design principles.

Gupta *et al.* [14] proposed Sonata, a query-driven telemetry system that dynamically partitions streaming queries between programmable switches and stream processors. While effective for packet-level analytics, its data model is not designed to capture multi-hop path semantics, limiting its support for INT-specific tasks such as end-to-end path reconstruction or hop-level correlation. Liu *et al.* [15] developed a Kafka-based high-throughput telemetry platform deployed in ESnet. Although it demonstrates impressive ingestion performance and supports stream processing via Kafka APIs, its topic-based storage architecture lacks structured representation of flow paths, making path-aware querying difficult.

Sgambelluri *et al.* [16] introduced a Kafka-based framework for optical network telemetry, combining real-time filtering with visualization in InfluxDB. However, similar to other TSDB-based solutions, it cannot preserve the end-to-end path structure of INT data, and thus struggles with spatiotemporal queries. Tovarňák *et al.* [17] presented a cloud-native analytics platform based on the Data Lakehouse model using components like Apache Kafka, Iceberg, and Spark. Their system is optimized for traditional point-level telemetry such as IPFIX

[1]https://github.com/lzhtan/Intdb

Listing 1: Example of an In-band Telemetry JSON Record

```
{
  "flow_id": "17343111536",
  "telemetry": [
    {
      "switch_id": "s1",
      "timestamp": "2025-04-21T10:00:00Z",
      "queue_util": 0.72,
      "delay_ns": 600
    },
    {
      "switch_id": "s2",
      "timestamp": "2025-04-21T10:00:01Z",
      "queue_util": 0.64,
      "delay_ns": 580
    },
    {
      "switch_id": "s3",
      "timestamp": "2025-04-21T10:00:02Z",
      "queue_util": 0.01,
      "delay_ns": 510
    }
  ]
}
```

and Syslog, but does not address the data modeling or query needs of INT. Zhou *et al.* [18] proposed PCAT, an evolvable telemetry framework that emphasizes monitoring intent evolution and configuration change management. Although it adopts a heterogeneous backend including SQL and key-value stores, its focus is on system adaptability rather than modeling and querying path-level telemetry data.

Collectively, these systems emphasize either scalability, configurability, or query flexibility. None of them offers a unified, path-oriented data model or an INT-native query abstraction that preserves and utilizes the spatiotemporal semantics of multi-hop telemetry data. Our work addresses this gap by proposing IntDB, a telemetry database designed specifically to support efficient storage, modeling, and querying of INT.

III. Design of IntDB

In this section, we describe the design and architecture of IntDB, a spatiotemporal database built specifically for INT. IntDB is designed to address the unique characteristics of INT data, including its path-oriented semantics, multi-hop nested structure, and complex query requirements.

A. System Overview

The architecture of IntDB is tailored to the unique characteristics of INT, which captures per-packet measurement data across multiple hops along a network path. Unlike point-level

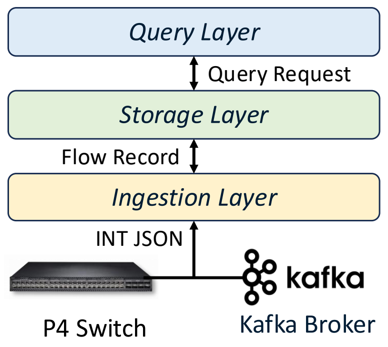

Fig. 1: System architecture of IntDB.

telemetry systems, which store and query independent time series, INT requires path-aware data ingestion, spatiotemporal preservation of flow structure, and fine-grained query capabilities across hop sequences.

As shown in Fig. 1, IntDB adopts a modular, three-layer architecture comprising an *Ingestion Layer*, a *Storage Layer*, and a *Query Layer*.

- **Ingestion Layer**: This layer ingests telemetry records exported from programmable switches (e.g., P4 targets) or Kafka brokers carrying INT payloads. Unlike traditional append-only ingestion in TSDBs, the ingestion layer reconstructs each telemetry flow into a coherent path structure by grouping hop-level records using the flow identifier. It ensures ordering, deduplication, and path-completeness before committing records to storage. For example, a flow traversing switches s1–s2–s3 will be represented as a nested array with ordered hop entries.
- **Storage Layer**: At the core of IntDB is a storage engine that preserves the multi-hop per-flow structure of INT data. Each record stores not just individual metrics but the spatial and temporal relationship between hops, enabling cross-hop correlation and path reconstruction. The system avoids flattening INT data into scalar time series, instead supporting native storage of path-oriented nested records. The layer also builds spatiotemporal indices that enable efficient retrieval of flows traversing specific switches, paths, or time intervals.
- **Query Layer**: The query layer exposes a declarative API tailored for INT. It allows users to issue path-level queries such as *find all flows where queue utilization exceeded 90% on any hop*. Such queries require preserving and traversing hop sequences, which is infeasible in flattened TSDB.

This design tightly integrates INT-specific semantics—such as hop order, inter-hop timing, and per-flow path identity—into each layer of the system, as shown in Tab. II. As a result, IntDB supports scalable telemetry analysis that retains end-to-end fidelity and enables powerful spatiotemporal queries over in-band telemetry streams.

IntDB supports basic CRUD operations, including Create/Insert, Read/Query, Update, and Delete. In addition, it also supports status and statistics monitoring, as well as data source integration with Prometheus [19] and Grafana [20].

TABLE II: Comparison of databases in handling INT data

Feature	InfluxDB	IntDB
Data Granularity	Point-level Scalar metrics per timestamp	Path-level nested telemetry per flow
Path Structure Preservation	Flattened metrics, no hop ordering	Preserves hop-by-hop order and structure
Spatiotemporal Semantics	Only time dimension	Supports both spatial and temporal dimensions
Query Support	Basic metric filtering, time-window aggregation	Native path-aware queries
Storage Model	Time-indexed columnar series	Nested store with path indexes
Indexing Mechanism	Time, tag fields (limited)	Flow ID, hop index, path digest, spatiotemporal multi-key
Suitability for INT	Poor	Fully supports

B. Storage Layer

The core innovation of IntDB lies in its specialized spatiotemporal data model designed specifically for INT, which preserves the complete semantics of network paths while supporting efficient spatiotemporal query operations. This section details the design philosophy, specific structures, and fundamental differences from traditional time series databases.

1) Fundamental Limitations of Traditional Time Series Databases: We first analyze the fundamental deficiencies of traditional time series databases when processing INT data. Time series databases represented by InfluxDB adopt a flattened data model that forcibly decomposes the multi-hop path data of INT into independent scalar time series.

Consider the INT record in Listing 1. Traditional TSDBs would store it as the flattened structure shown in Table III. This storage approach leads to three critical problems: (1) Path structure loss—the logical ordering and dependencies between hops are destroyed; (2) End-to-end semantic absence—the complete path characteristics of flows cannot be effectively expressed; (3) Query complexity explosion—path reconstruction requires complex JOIN operations, causing dramatic performance degradation.

TABLE III: Flattened storage of INT data in InfluxDB

Measurement	Switch	Timestamp	Field	Value
queue_util	s1	2025-04-21T10:00:00Z	value	0.72
delay_ns	s1	2025-04-21T10:00:00Z	value	600
queue_util	s2	2025-04-21T10:00:01Z	value	0.64
...	...	...	...	...

2) IntDB: The fundamental challenge in INT data management arises from the mismatch between the inherently path-centric nature of telemetry flows and the scalar-oriented architecture of traditional TSDBs. While TSDBs excel at storing and querying isolated metric streams, they fundamentally lack the ability to preserve the spatial-temporal correlations and structural dependencies that define network path semantics.

Consider an INT flow F that traverses a sequence of switches $P = \langle s_1, s_2, ..., s_n \rangle$. At each hop s_i, the switch generates a telemetry measurement $h_i = (\text{timestamp}_i, \text{queue}_i, \text{delay}_i)$ at time t_i. In conventional TSDBs, this structured data is decomposed into disjoint time series, effectively flattening the end-to-end path context into fragmented, hop-level records. Reconstructing the original path semantics requires expensive correlation operations that, for a dataset containing k flows with average path length h, may require $O(k \cdot h \cdot \log n)$ operations under typical B-tree indexing schemes, where n represents the total number of stored measurements.

The core technical challenge is to design a storage model that preserves the structural relationship among the path (P), its constituent switches, and per-hop measurements (H), while enabling efficient queries across temporal, spatial, and topological dimensions. This entails addressing three key subproblems: (1) maintaining hop-level ordering within each flow without excessive storage overhead; (2) supporting efficient path-pattern queries, such as prefix and subpath matching; and (3) enabling scalable spatiotemporal range queries over high-volume telemetry datasets.

IntDB addresses these challenges through a unified data model that treats flows as first-class entities while carefully managing the storage-performance trade-off inherent in path preservation. The model is formally defined as follows:

A INT flow F is represented as a 4-tuple (ID, P, H, M), where:

- ID is a globally unique flow identifier.
- $P = \langle s_1, s_2, ..., s_n \rangle$ represents the ordered sequence of switches traversed by the flow.
- $H = \{h_1, h_2, ..., h_n\}$ denotes the set of hop-level telemetry records.
- M captures flow-level metadata including temporal bounds and derived path characteristics.

Each hop record h_i explicitly encodes the switch identifier, measurement timestamp, and raw telemetry metrics. This structure ensures that the original INT data from Listing 1 retains its complete path semantics during both storage and query execution, enabling direct path-level operations without reconstruction overhead.

In general, the core storage structure of IntDB can be expressed as shown in Listing 2.

To address storage efficiency concerns, IntDB employs several optimization strategies. Path templates are extracted for frequently occurring switch sequences, allowing delta-compression of similar flows. Additionally, hop-level aggregation is performed during ingestion to reduce the storage footprint of high-frequency telemetry streams while preserving essential path characteristics.

The model incorporates optional spatial enhancement through a metadata extension framework. In deployments where network topology information is available, spatial coordinates may be associated with switch identifiers as $coord_i = (x_i, y_i, zone_i)$. However, recognizing the deployment complexity and maintenance overhead of comprehensive spatial mapping, this enhancement is designed as a progressive feature that can be selectively enabled based on available infrastructure capabilities.

Listing 2: The Data Structure of IntDB

```
flow_id: "17343111536"
path: NetworkPath {
    switches: ["s1", "s2", "s3"]
    path_hash: "a1b2c3d4e5f6789..."
}
start_time: 2025-01-18T10:00:00Z
end_time: 2025-01-18T10:00:05Z
hops: [
    {hop_index: 0, switch_id: "s1",
        timestamp: "...", metrics: {...}}
    {hop_index: 1, switch_id: "s2",
        timestamp: "...", metrics: {...}}
    {hop_index: 2, switch_id: "s3",
        timestamp: "...", metrics: {...}}
]
```

C. Query Layer

The query processing subsystem in IntDB is specifically designed to exploit the path-aware data model for efficient telemetry analysis. Unlike traditional TSDBs that rely on expensive join operations to correlate related measurements, IntDB's query engine operates directly on preserved path structures, enabling complex network analysis queries with significantly reduced computational overhead.

IntDB supports complex queries that combine temporal, spatial, and metric constraints through a unified QueryBuilder interface. A typical INT analysis query involves multiple condition types:

- **Temporal conditions:** Time range filters (e.g., last 24 hours), sliding windows.
- **Spatial conditions:** Path pattern matching (e.g., `spine` $\rightarrow$ `leaf` $\rightarrow$ `server`), switch-specific filtering (e.g., flows traversing designated switches).
- **Metric conditions:** Performance thresholds (e.g., delay $> 500\,\text{ns}$), statistical constraints (e.g., queue utilization $> 90\%$).

Listing 3 is a query example in IntDB. The purpose of this query is to find all high-latency telemetry flows that pass through switch s2, have a path length between 3-5 hops, are within the last 30 minutes, and have a total latency greater than 1000 nanoseconds.

The query processor employs a cost-based optimization strategy to determine the optimal execution order for multi-dimensional filters, minimizing intermediate result set sizes.

The subsystem supports three primary types of path queries: exact path matching, prefix/suffix matching, and regular expression-based pattern matching.

(1) Exact Path Matching: For exact path queries, IntDB utilizes flow ID or a hash-based path signature mechanism. Each path $P = \langle s_1, s_2, ..., s_n \rangle$ is mapped to a signature $\sigma(P)$ using the hash function.

Listing 3: A Query Example of IntDB

```
{
  "time_conditions": [
    {"type": "within_minutes", "value": {"
        minutes": 30}}
  ],
  "path_conditions": [
    {"type": "through_switch", "value": {"
        switch_id": "s2"}},
    {"type": "length_range", "value": {"min"
        : 3, "max": 5}}
  ],
  "metric_conditions": [
    {"type": "delay_gt", "value": {"
        threshold": 1000}}
  ],
  "limit": 100,
  "include_flows": true
}
```

(2) Prefix/Suffix Matching: For path prefix and suffix queries, IntDB maintains a radix tree (compressed trie) that enables efficient prefix lookup in $O(|prefix|)$ time, independent of the total number of stored paths.

(3) Pattern Matching: For complex path patterns involving wildcards and regular expressions, IntDB employs a finite state automaton (FSA) approach. Path patterns are compiled into FSAs, and path matching is performed through state transitions, achieving $O(|path| \cdot |pattern|)$ worst-case complexity.

The path-preserving design introduces storage overhead proportional to the path length, compared to flattened TSDB storage. However, this overhead enables significant improvements in query performance for path-oriented operations:

- *Path reconstruction*: $O(\log k{+}h)$ in IntDB vs. $O(h \log n)$ in TSDB.
- *Single-flow aggregation*: $O(h)$ linear scan in IntDB vs. $O(h \log h)$hop correlation and sorting in TSDB.
- *Path pattern matching*: $O(\log k + m)$ using path indexing vs. $O(k \cdot h)$ exhaustive search, where m is the result set size.

Here, h denotes the average path length, k is the number of flows, and n is the total number of telemetry records.

IV. Experimental Results and Analysis

We evaluated the performance of IntDB against InfluxDB on a MacBook Air (2020, Apple M1, 8-core CPU, 16GB RAM) under representative telemetry workloads. IntDB v0.2.0 and InfluxDB v2.7.11 were deployed locally.

A. Write Performance

We evaluated the baseline responsiveness of both systems using a lightweight HTTP stress test. The benchmarking test tool is siege[2] v4.1.7. The tested endpoints were the respective health-check interfaces of each database.

[2]https://github.com/JoeDog/siege

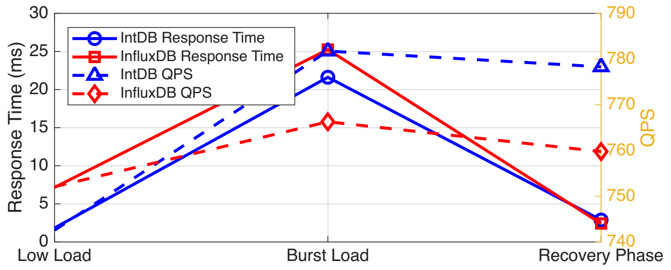

Fig. 2: The handling capability of Burst load.

Fig.2 evaluates the resilience of IntDB and InfluxDB to burst load under varying load phases, including normal load, burst, and recovery. While both databases experience a drop in throughput during the burst phase, IntDB sustains a slightly higher queries per second (QPS), especially during the recovery period. InfluxDB stabilizes more quickly in response time after a burst, whereas IntDB requires a longer recovery duration despite regaining high throughput. In conclusion, InfluxDB demonstrates better real-time responsiveness and burst load tolerance. IntDB excels in throughput recovery.

The results of concurrent scalability are shown in Fig.3. IntDB consistently achieves higher throughput (QPS), making it more suitable for scenarios with heavy data ingestion or high transaction volume. However, InfluxDB demonstrates superior latency characteristics, including lower average response times and better control of peak latencies under high concurrency, which is critical for latency-sensitive applications. In addition, both databases maintained 100% availability and exhibited negligible error rates in all tests, confirming their robustness and stability under stress. While IntDB provides better raw throughput, InfluxDB exhibits higher concurrency efficiency, processing more requests per concurrent thread.

B. Query Performance

We employed an incremental load testing methodology with three different concurrent query scales: 1K, 5K and 10K. And each experiment involves two typical network telemetry query patterns:

1) **Path Pattern Matching Query**: Finding all flows passing through specific switches.
2) **Path Aggregation Query**: Statistical aggregation of network flows within specified time windows.

Figure 4 presents three core performance comparison results. Under path pattern matching, the average response time of IntDB is improved by 87.7%. Under path aggregation, the average response time of IntDB is improved by 82.4%.

V. Conclusion

In this paper, we presented IntDB, a spatiotemporal database designed for in-band network telemetry. To the best of our knowledge, this is the first work that studies in-band network telemetry data storage. IntDB addresses the storage challenges of INT through a path-oriented data model and query engine. The storage and query of in-band network telemetry data is a

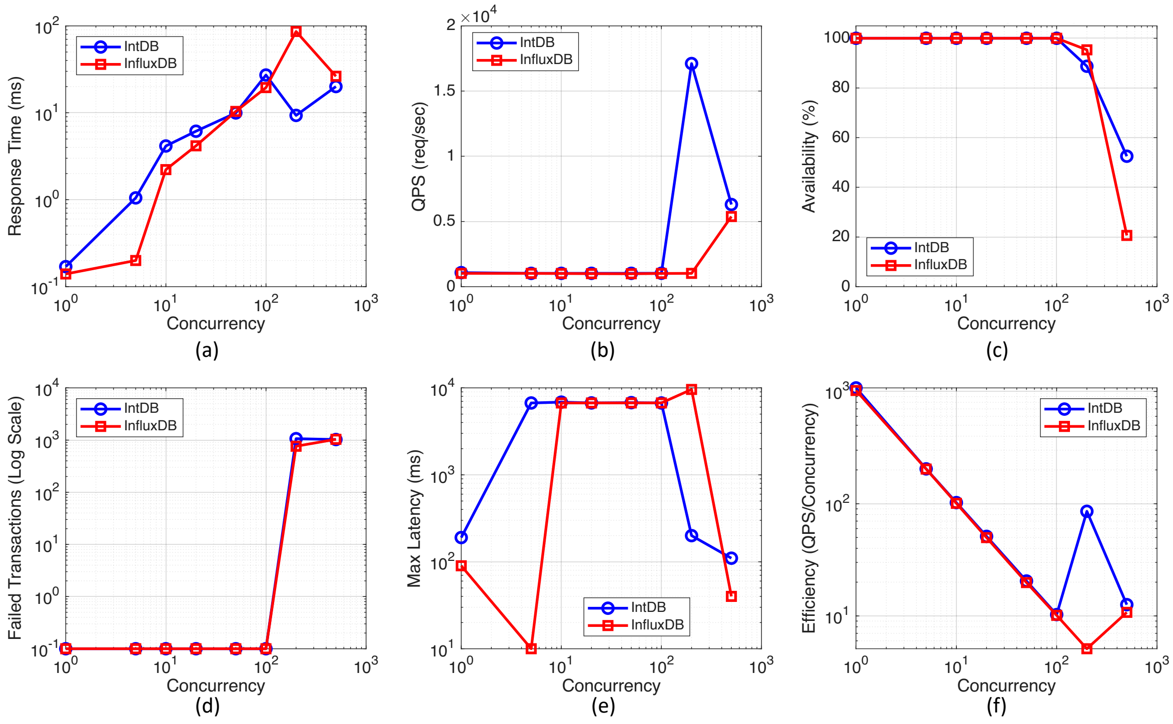

Fig. 3: The result of concurrent writing scalability.

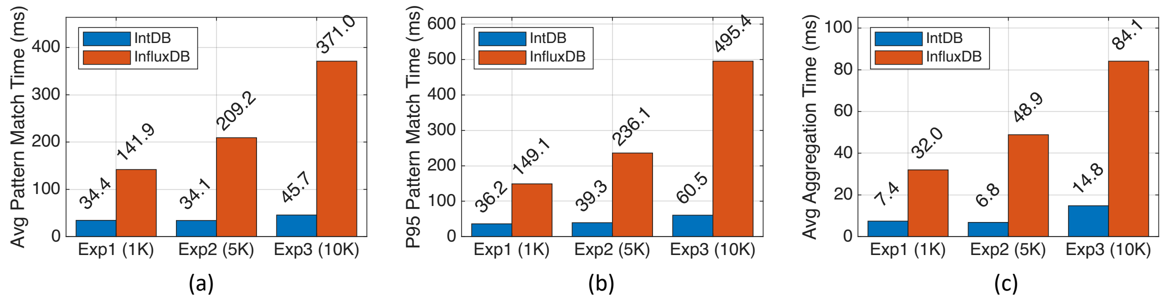

Fig. 4: The result of query performance.

highly customized problem, and IntDB will continue to evolve towards universality and efficiency in the future.

References

[1] J. D. Case, M. Fedor, M. L. Schoffstall, et al., ”Simple Network Management Protocol (SNMP),” RFC 1157, IETF, May 1990.

[2] B. Claise, Ed., ”Cisco Systems NetFlow services export version 9,” RFC 3954, IETF, Oct. 2004.

[3] Y. Tian and J. Gao, Network Analysis and Architecture. Springer, 2024.

[4] H. Song, F. Qin, P. Martinez-Julia, et al., ”Network Telemetry Framework,” RFC 9232, IETF, June 2022.

[5] M. Yu, ”Network telemetry: Towards a top-down approach,” ACM SIGCOMM Comput. Commun. Rev., vol. 49, no. 1, pp. 11–17, 2019.

[6] L. Tan, W. Su, W. Zhang et al., ”In-band network telemetry: A survey,” Computer Networks, vol. 186, p. 107763, 2021.

[7] J. Meza, T. Xu, K. Veeraraghavan et al., “A large-scale study of data center network reliability,” in IMC’18, 2018, pp. 393–407.

[8] X. Yu, K. Ye, D. He et al., “MLPing: real-time proactive fault detection and alarm for large-scale distributed IDC network,” in ICDCS’24, 2024, pp. 913–924.

[9] Y. Zhou, C. Sun, H. H. Liu, et al., ”Flow event telemetry on programmable data plane,” in SIGCOMM’20, 2020, pp. 76–89.

[10] S. Troia, E. Gregorini, J. P. Asdikian, et al., ”Real-time delay measurement in SD-WAN based on in-band network telemetry,” in HPSR’24, 2024, pp. 149–154.

[11] N. Van Tu, J. Hyun, G. Y. Kim, et al., ”Intcollector: A high-performance collector for in-band network telemetry,” in CNSM’18, 2018, pp. 10–18.

[12] Z. Liu, B. Mah, Y. Kumar, et al., ”Programmable per-packet network telemetry: From wire to Kafka at scale,” in SNTA’20, 2020, pp. 33–36.

[13] S. Zhu, J. Lu, B. Lyu, et al., “Zoonet: A proactive telemetry system for large-scale cloud networks,” in CoNEXT’22, 2022, pp. 321–336.

[14] A. Gupta, R. Harrison, M. Canini, et al., ”Sonata: Query-driven streaming network telemetry,” in SIGCOMM’18, 2018, pp. 357–371.

[15] Z. Liu, B. Mah, Y. Kumar, et al., ”Programmable per-packet network telemetry: From wire to Kafka at scale,” in SNTA’21, 2021, pp. 33–36.

[16] A. Sgambelluri, A. Pacini, F. Paolucci, et al., ”Reliable and scalable Kafka-based framework for optical network telemetry,” Journal of Optical Communications and Networking, vol. 13, no. 10, pp. E42–E52.

[17] D. Tovarňák, M. Raček, and P. Velan, ”Cloud native data platform for network telemetry and analytics,” in CNSM’21, 2021, pp. 394–396.

[18] Y. Zhou, Y. Zhang, M. Yu, *et al.*, ”Evolvable network telemetry at Facebook,” in NSDI’22, 2022, pp. 961–975.

[19] Prometheus, https://prometheus.io/.

[20] Grafana, https://grafana.com/.