Efficient Data Confidence Fabrics With Compact Annotations

Asfa Khalid*, Seán Óg Murphy, Cormac Sreenan, Utz Roedig
School of Computer Science and Information Technology (CSIT)
University College Cork
Cork, Ireland
*Contact Author email: 123111938@umail.ucc.ie

***Abstract*—The need to trust data has become a key requirement in modern distributed systems. To facilitate measurable trust and confidence in data and applications spanning heterogeneous systems, the emerging concept of a Data Confidence Fabric (DCF) offers a compelling solution. Data producers and processors provide metadata, known as annotations, recording trust insertion along the data delivery chain. Thus, it is possible to asses the trustworthiness of data before processing it. While a DCF, such as the Alvarium framework, enables this management of trust, there is a cost in terms of the overheads associated with security annotations themselves. To improve efficiency of a DCF, we therefore propose a set of techniques for making annotations more compact and to reduce the number of DCF transactions. Our work shows that transaction efficiency gains of up to 93% in our considered use cases can be achieved.**

***Index Terms*—Data compression, Data Annotation, Trust, Zero Trust, Data Confidence Fabric, Security**

I. Introduction

In large-scale heterogeneous distributed systems, the problem of data trustworthiness is especially challenging. Can the source of data be trusted? Who created data? Who has handled data between creation and consumption? Data may even be traded and change ownership before it is used. An example is traffic data collected by a large amount of sensors in a smart city. The data is stored for some time during which it changes ownership and after a period the data is finally used to train an AI model that is used to predict traffic conditions. When training such a model it is essential to understand the trustworthiness of the data and perhaps to prioritise depending on its collection history.

Data Confidence Fabrics (DCFs) have been proposed to address the aforementioned problem space. A DCF is used to record the presence or absence of particular security characteristics along the data processing path. Each processing element (sensor, edge device, storage system, ...) handling data records relevant security information in the DCF. For example, who is the sensor provider and was a Trusted Platform Module (TPM) present at data creation; which edge device forwarded the data and was this done securely; which storage systems stored the data or who pre-processed data before storage. These security characteristics are recorded as so called *Annotations* within the DCF, by devices handling the data on its journey through the network. We note that the DCF does not store the data itself, it only maintains the annotations. A requirement is that any data consumer must be able to evaluate access recorded annotations it is necessary to store these in a manner that is open, accessible and secure. This goal can be achieved by recording annotations within a distributed ledger and thus, DCFs usually make use of blockchain technology. A prominent example of a DCF is the Linux Foundation's project Alvarium [1], [2], on which we base our research. Formerly using IOTA Tangle [3] as its distributed ledger, as of May 2024 Alvarium makes use of the Hedera hashgraph ledger [4] as its source of immutable transaction history.

One significant challenge of a DCF is the overhead associated with annotations. Firstly, annotations documenting trust have to be communicated and stored, and therefore the annotation size matters. Secondly, as annotations are usually stored in a distributed ledger, transaction costs are incurred. The two aspects *Annotation Size* and *Annotation Frequency* are of particular importance when small amounts of data are generated with a high frequency as it is common for IoT devices. This provides the key motivation for our research.

Examples of data generation in applications where security annotations might be used might include:

- periodic temperature sensor readings
- motion-activated camera video
- audio capture

For some applications, the volume of communication and processing is dominated by the data itself (as in the case of images or video). In other applications, the data itself may be small (e.g. a simple timestamped temperature reading) but generated frequently and possibly from very many sources. In such cases, the annotation overhead would dominate the resource use relative to the data itself.

The specific contributions of this paper are in reducing annotation size, reducing transaction frequency, and evaluating the relative performance improvements of these changes. The remainder of this paper describes Data Confidence Fabrics in detail, the specifics of the Alvarium DCF, the significance of annotation events, and our suggested format changes to improve efficiency. Finally we present experimental evaluations of the performance of these changes and the consequences of choosing each change on DCF use.

This paper focuses on reducing the size and frequency of annotations while maintaining functionality of annotations, and evaluates the performance improvements. In Section II,

we will review background material, exploring the role of Data Confidence Fabrics (DCF) in a Zero Trust environment and compare it to similar technologies, such as the Coalition for Content Provenance and Authenticity's (C2PA), which operates in a similar way to DCF. In Section III, we will focus on the Alvarium DCF and annotations. This section will cover annotation events, annotation fields, and their classification based on their impact on data security and trust. In Section IV we will propose methods and schemes to generate annotations that are smaller and more streamlined than those produced by the standardized method. In Section V, we will evaluate the results of our proposed changes by measuring the performance improvements in terms of annotation size and transaction frequency. Section VI will review the related research, as published in the literature, regarding efficient metadata and annotations. Finally, in Section VII we will conclude with the finding that our revised annotation formats effectively reduce overhead, making Data Confidence Fabrics more efficient.

II. Data Confidence Fabrics

A Zero Trust (ZT) [5] security environment is a network and computing environment with many heterogeneous participants, none of which are implicitly trusted simply because of their participation. Devices and actors may or may not be credentialed, their identities may be unconfirmed and security technologies may be present or absent in different parts of the environment. In order to properly process and share information and access in a ZT environment, contextual information is required to evaluate participants, equipment and service access. An important tool for security evaluation in ZT is the use of a Trust Algorithm or Risk Scoring scheme [6] which takes contextual information (security technologies, participant history, application requirements etc.) to determine the level of access given to participants as well as data forwarding and processing decisions.

A Data Confidence Fabric [7] [8] is a technology for associating security metadata with data produced, processed and communicated throughout a network environment following ZT principles. This metadata is associated with the data either through a common key (e.g. the hash digest of the data) or incorporated with the data itself in the form of a metadata digest.

Existing DCFs include the Coalition for Content Provenance and Authenticity's (C2PA [9]) Content Authentication Initiative (CAI) [10], a standard for generating and signing metadata for media content at time of creation or editing. At time of creation (e.g. taking a photo), the configuration of the device and its user are examined and used to populate a JSON manifest documenting the state of the device at that time.

Alvarium [8] is a Linux Foundation [2] initiative to produce an open-source DCF for security and trust in large distributed networks. The principal design difference between Alvarium and manifest-based approaches like CAI is that security metadata ("annotations") generated for data is not attached directly to that data, but instead is communicated via a different network path to a common source of distributed truth

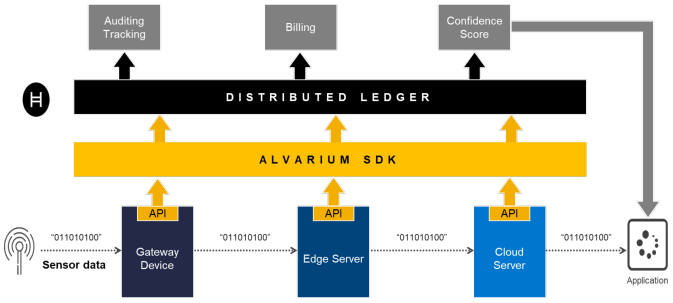

Fig. 1. Data Confidence Fabrics Workflow

such as an immutable ledger (e.g. a blockchain or hashgraph, Figure 1). While data is free to traverse the network to where it is needed, the security features of the device creating, transiting or altering that data travels separately. Decoupling the metadata from the data allows for a truly distributed data confidence framework – participants in the network can access security metadata from an immutable ledger when evaluation of data trustability is required simply by requesting annotations matching the hash digest of the data itself.

Use cases for Alvarium include many applications involving individual discrete sensor devices, machines, energy producers, medical devices or even security cameras. Depending on the device or application, a reading or report might be in the form of a small message (e.g. a temperature reading, device ID and timestamp) or something much larger (a video feed digest).

III. Alvarium DCF and Annotations

The Alvarium DCF is present on devices in the network where security metadata needs to be monitored, particularly where data is generated, altered, amalgamated or stored. Furthermore, where data is transiting through a node in the network, if Alvarium is present at that node, the security configuration at that device can also be annotated if assurance is required that eavesdropping or other attacks are avoided.

A. Annotation Events

annotationevents) The events which cause annotations to be attached to a piece of data when it is processed at a given device are:

- **create()** - *when data is generated/originated*
- **mutate()** - *when data has been edited, merged, or is otherwise associated with another datum*
- **transit()** - *data originating externally to the device is passing through and being relayed further, e.g. at network gateways*
- **publish()** - *general purpose function to generate annotations if required for local use*

Alvarium uses a collection of annotators to generate annotations when one of these events occurs at the device. Each annotator is reponsible for verifying that a given security feature, technology or history is correct on that device at the time of the annotation event. Existing annotators include Transport

```
{
"id":"01HSSG...AJW18HB",
"key":"104...1FAA",
"hash":"sha256",
"host":"csYFCV9Z3",
"kind":"tpm",
"signature":"8034FB...072B08",
"isSatisfied":false,
"timestamp":"2024-03-25T00:38:57.83230Z"
}
```

Listing 1: TPM Annotation example

Layer Security (TLS) verification, Public Key Infrastructure verification, Trusted Platform Model verification and codebase checks such as Checksum and Software Bill of Materials (SBOM) [11].

B. Annotations

Annotations (as in the example in Listing 1) are represented as JSON (JavaScript Object Notation) Objects in a JSON Array with the following fields:

- **id** - *unique identifier, distinct label for the annotation. Generated using a Universal Lexigraphical ID generator (ULID) [12]*
- **key** - *result of applying a hash algorithm on the data*
- **hash** - *name of the hash algorithm that was used to make the key*
- **host** - *identifier of the device generating the annotation*
- **kind** - *security feature being annotated*
- **signature** *cryptopgraphic signature applied to the annotation JSON using the device's private key*
- **isSatisfied** - *indicates whether the criteria defining the annotation was fulfilled*
- **timestamp** - *time when annotation is created*

The array of annotations is combined with the name of the annotation event (e.g. "create", "mutate" etc) and any other important information (such as the hash "key" of the previous data in the case of a "mutate" event, allowing for previous data to be linked to the new data or vice versa). This is then passed along to a Publisher which communicates the annotations upstream to the message broker or ledger depending on the DCF configuration. The Publisher is responsible for encrypting and serializing the annotations, and may also apply data compression [13] (e.g. GZip) prior to sending. In our work, we propose changes to the format and structure of the JSON object prior to its being passed to the Publisher (which will also apply after decompression before entry into a ledger/broker), and hence we leave information entropy-based compression approaches to the purview of the Publisher.

C. Annotation Field Classification

To classify the fields based upon their characteristics for efficient compression of annotations, we adopt a scheme inspired by Tomoskozi et al [14] for systematically characterising metadata or header fields based on their intended usage, relative redundancy and variability. We classify each field of the standard Alvarium annotation format under one of four classes: "Static", "Rarely Changing", "Irregular" and "Pattern". We use these classifications to motivate our compact and efficient adjustments to the annotation format (Table I).

TABLE I
ANNOTATION FIELD CLASSIFICATION

Field Name	Classification	Analysis
id	Irregular	semi-random string
key	Irregular	hash digest
hash	Static	hash algorithm name
host	Static	device name
kind	Static	annotation type
signature	Irregular	crytpographic hash digest
isSatisfied	Rarely Changing	security test result
timestamp	Pattern	periodically incrementing

Static Fields: Effectively constant, with a change in field value unlikely to occur outside of a reboot, update or system migration – static fields are typically the same each time annotations are generated. Within a single annotation event the Key is static, as each annotator within that event is operating on the same piece of data (hence the key, which is the hash of that data, will be common to each annotation within the given operation).

Rarely Changing: Values that are unlikely to change from one operation to the next, as they rely on changeable functionality or external factors, so cannot be guaranteed to hold static values. Typical examples in Alvarium format would include the "isSatisfied" value for most annotations – that is to say that the security status and configuration of the device on which Alvarium is operating are unlikely to differ from one moment to the next (though it is still possible).

Irregular: Exhibit no pattern and do not repeat. As Alvarium makes use of hashing and cryptographic signing, the typical Irregular fields are the hash digests and signatures within the annotations. As the id is generated by ULID, it is a largely a random string exhibiting high entropy appended to a timestamp value to support sorting by order of creation via a single field.

Pattern Fields: Have values which may not repeat but do exhibit a relationship with prior values either within that field or in context with others. Within the standard annotation format, the timestamp field is a fruitful candidate for this classification, as time proceeds in a sequential manner we need not refer to the absolute time on each occasion.

IV. REVISED ANNOTATION FORMATS

In practical use-cases, such as small-payload IOT sensors (refintroduction), it is possible that the annotation payload for a piece of data might dwarf the data itself. We also observe that when communicating annotations to immutable ledgers, there is typically an upper limit on the size of such transactions as well as a cost associated with each transaction. As such, there

is a motivation to improve the efficiency of the annotation overhead to achieve:

- Reduced network bandwidth use
- Reduced number of ledger transactions
- Maximized work per transaction

It is also critical that in pursuing revised formatting, we preserve the utility of the annotation scheme. As such, any changes should not remove important functionality such as the ability to retrieve annotations for a piece of data with just the data itself, or to retrieve annotations for a given device with a particular time-frame. To accomplish these, we propose three methods: Concise, Compact, and Super-Compact as outlined below. We also propose a batching approach (Section IV-D) where a collection of annotations are accumulated over a period of time prior to transmission to the ledger, making more efficient use of a given ledger transaction at the expense of some delay on annotation communication while the batch is built up.

A. Concise

The Concise annotation format achieves performance improvements through using alternative schemes for hashing data, the omission of a distinct ID field and representing the timestamp using the Unix Time. In our evaluation, we substituted the default SHA256 hashing scheme with MD5, providing a shorter hash digest at the expense of potentially higher frequency of hash collisions. MD5 is not as cryptographically strong as SHA256 but in many situations the size reduction may outweigh the possible impact on security. Where the rate and volume of data generation is low, this may be a good choice for improving efficiency.

As a specific ID field is absent, the annotation can instead be retrieved through the value of the signature (which is effectively a random string) and can be sorted via the timestamp, maintaining the utility of ULID without requiring one. Representing time using the UNIX time value rather than the timestamp string removes human readability, but conversely may improve retrieval and processing efficiency as it avoids a timestamp conversion step both at generation and digestion. Given that annotations are typically to be consumed by algorithms rather than human users, this should not have any impact on utility.

B. Compact

The Compact approach involves taking all annotations generated in a single annotation event (Section III-B) and discarding redundant fields. Some annotation fields have values determined on initialization either due to static values or DCF configuration, and have identical values on the *host* (device name) and *hash* (chosen hashing algorithm name) fields. As a single annotation event is specific to a given piece of data, the hash digest is identical for each annotation and can be reduced to one field rather than on per security test. A further saving is possible by defining the timestamp as the relative difference from the time of the first test – a shorter value (e.g. hundreds of ms) than the full timestamp.

C. Super Compact

In the Super Compact format, some additional savings are possible at the expense of losing the information required to convert into original, standard annotations (i.e. losing backwards compatibility). These savings are achieved by:

- Signing the whole super compact object once instead of each individual constituent security annotation
- Using a single timestamp for the super compact object, without the time offsets for each security test
- Assuming a security test's "isSatisfied" result is "true" unless explicitly stated otherwise

D. Batching Annotations

By accumulating annotations over a period of time/annotation events prior to transmission, it's possible to reduce the annotation overhead substantially. As many fields in annotations rarely change (Section I), a series of annotations over a period of time are likely to have identical values. This can be exploited by batching annotations with common values and including those values just once. While these fields rarely change, there is a possibility that from one annotation event to the next that the value changes. As an example, the TLS verification at one moment may fail at the next due to a certificate expiry or losing connection to verification services. In the event that the state of the annotations changes from one event to the next, an existing batch needs to be completed early and a new batch started with the new correct state. In the case that a batch can continue to accumulate due to lack of change in circumstances, criteria for completing are:

- A given period of time has passed (user configurable)
- The batch has reached a targeted size (e.g. the upper limit on a single ledger transaction)

V. Evaluation

To evaluate the relative impact of each of our revised annotation formats, we examined both the byte-cost for individual annotation events (Table II) where three security test annotators are used (TPM,TLS and PKI), as well as the number of annotation events that could be combined into a single transaction of 6 kilobytes (approximating the current maximum transaction size using the Hedera hashgraph ledger) (Figure 2). The experiments represent a typical use case where endpoints such as sensors, machines, or medical devices produce data and provide security measures such as TPM, PKI and TLS.

In these results we find that, when using unmodified ("standard") Alvarium annotations, up to 6 events could be combined in a single Hedera transaction. Where the concise formatting is applied, this rises to 8, and 10 when compact formatting is used in the case of three security tests in each annotation event. A significant increase to 19 annotation events per transaction is possible using the super-compact format. The most substantial result is that of the batch format, provided the security state of the device has not changed from the start of the batch and transmission. In this case 156 annotation events can be accumulated in a single transaction, though in

TABLE II
ANNOTATION EVENT SIZES

Annotation Format	Annotation Size
Standard	973
Concise	710 bytes
Compact	545 bytes
Super-Compact	310 bytes
Batch	5838 bytes (157*3 annotations)

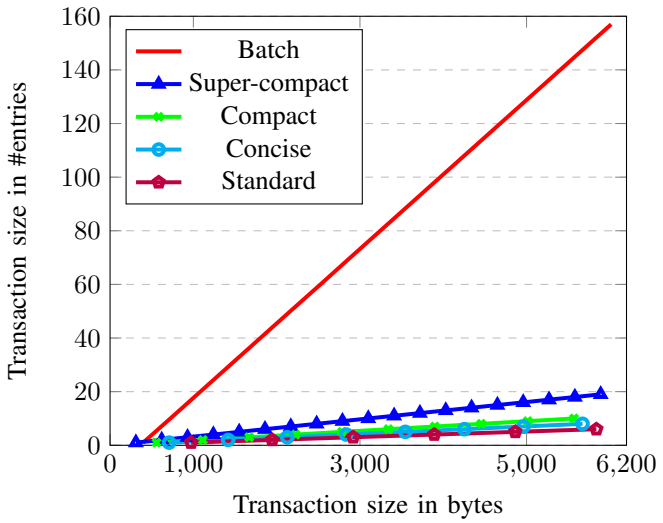

Fig. 2. Transaction size per number of entries for 3-annotation operations in each format

practice this may introduce a substantial delay if these events have a low frequency. Where this is an issue, the alternative batch completion criteria of maximum duration would finish the batch at a smaller size, though in our results we find that where the batch contains at lease two annotation events, it still outperforms the other formats. Thus, if annotations change with every event, batching is not useful. However, such scenario would be a rare occurrence.

VI. Related Work

Improving efficiency by reducing the size of metadata has been studied in other contexts, notably, for packet header and data representation formats. Approaches that seek to mitigate the overheads of small storage transactions has attracted considerable attention, most recently in regard to blockchain.

Situations in which the metadata may be larger than the data itself are common in many contexts, but most notably attract attention for the Internet of Things and sensor networks [15], where embedded devices send small updates, and often need to do so while operating with minimal energy. To this end, compression of protocol packet headers has attracted considerable attention [14]. A common approach is to maintain state between the transmitter and receiver, initially sending the full header, and subsequently transmitting only changes to the values of header fields [16]. The 6LoWPAN-GHC standard defines a header compression for IPv6 over Low-Power Wireless Personal Area Networks [17]. These solutions are designed for use with a sender and single receiver, each maintaining the relevant state. These are unsuitable for our purpose, with large device numbers generating annotations.

In Sections II and IV we described how ledger transaction maximum sizes can be a limiting factor on how efficient and productive an annotation scheme can be. For blockchains this general topic has been investigated previously, e.g. in [18], [19], [20] [21]. Their focus is on enhancing transaction efficiency irrespective of the transaction content. In contrast, our approach involves a more adaptive optimization strategy taking specific annotation structure and usage into account. As we already know the specific contents that will be included in the ledger, we can optimize the process even further by using batching to handle those particular types of transactions in a manner that respects application context.

In Section III-B we described how the Alvarium Publisher component can be configured to apply information-theory-based data compression to annotations prior to network transmission. These representations are designed to be human-readable, and consequently are quite verbose. Techniques specific to JSON or XML formatted metadata [22] have been developed which may give network bandwidth savings due to specific exploitation of the common features of metadata representing in this protocols.

VII. Conclusion

We described Data Confidence Fabrics and their use in establishing a security and trust context that is reliable and distributed. We identified the potential inefficiency of annotation structures and formats for practical applications with frequent data generation and with limited distributed ledger transaction constraints. In our work we suggest four modifications to the annotation strategies to reduce the overhead of maintaining a distributed confidence fabric, with relative improvements in each format, though sometimes at the expense of backwards-compatibility, human-readability, or immediacy of annotation communication.

Acknowledgment

This work was conducted as part of the CLEVER project, EU Grant Number 101097560 and EI No: IR-2022-0065, and supported in part by a research grant from Science Foundation Ireland (SFI), Grant Number 13/RC/2077 P2.

References

[1] J. Lovato, "New Linux Foundation effort to focus on data confidence fabrics to scale digital transformation initiatives," https://www.linuxfoundation.org/press/press-release/new-linux-foundation-effort-to-focus-on-data-confidence-fabrics-to-scale-digital-transformation-initiatives, 2019, [Online; accessed 26-March-2024].

[2] Linux Foundation, "Project Alvarium," https://lfedge.org/projects/alvarium/, [Online; accessed 26-March-2024].

[3] IOTA Foundation, "IOTA Tangle," https://www.iota.org/get-started/what-is-iota/, [Online; accessed 24-May-2024].

[4] L. Baird, M. Harmon, and P. Madsen, "Hedera: A public hashgraph network & governing council," *White Paper*, vol. 1, no. 1, pp. 9–10, 2019.

[5] V. Stafford, "Zero trust architecture," *NIST special publication*, vol. 800, p. 207, 2020.

[6] S. Teerakanok, T. Uehara, and A. Inomata, "Migrating to zero trust architecture: Reviews and challenges," *Security and Communication Networks*, vol. 2021, pp. 1–10, 2021.

[7] "Data Confidence Fabric and the Importance of Vetted Data — Dell — dell.com," https://www.dell.com/en-ie/perspectives/data-confidence-fabric-and-the-importance-of-vetted-data/, [Accessed 21-04-2024].

[8] Steve Todd, Dell Technologies Inc., "Project Alvarium: The future of edge data," https://education.dell.com/content/dam/dell-emc/documents/en-us/2020KS_Todd_Project_Alvarium-The_Future_of_Edge_Data.pdf/, 2020, [Online; accessed 26-March-2024].

[9] L. Rosenthol, "C2pa: the world's first industry standard for content provenance (conference presentation)," in *Applications of Digital Image Processing XLV*, vol. 12226. SPIE, 2022, p. 122260P.

[10] C. A. Initiative, "Working with Manifests: Open-source tools for content authenticity and provenance," https://opensource.contentauthenticity.org/, [Accessed 21-04-2024].

[11] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, "An empirical study on software bill of materials: Where we stand and the road ahead," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2630–2642.

[12] Universally Unique Lexicographically Sortable Identifier, "The canonical spec for ulid," https://github.com/ulid/spec, [Online; accessed 31-May-2024].

[13] J. C. Viotti and M. Kinderkhedia, "A survey of json-compatible binary serialization specifications," *arXiv preprint arXiv:2201.02089*, 2022.

[14] M. Tömösközi, M. Reisslein, and F. H. Fitzek, "Packet header compression: A principle-based survey of standards and recent research studies," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 698–740, 2022.

[15] S. R. Zahra and M. A. Chishti, "Packet header compression in the Internet of Things," *Procedia Computer Science*, vol. 173, pp. 64–69, 2020.

[16] A. Reinhardt, S. M. Parag, T. Koenig, and R. Steinmetz, "SFHC.KOM: Stateful header compression for wireless sensor networks," in *IEEE Local Computer Network Conference*, 2010, pp. 598–605.

[17] C. Bormann, "6LoWPAN-GHC: Generic header compression for ipv6 over low-power wireless personal area networks (6LoWPANs)," RFC 7400, Nov. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7400

[18] Y. Wang and Y. Tang, "Poster: Enabling cost-effective blockchain applications via workload-adaptive transaction execution," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3483–3485.

[19] P. Ruan, D. Loghin, Q.-T. Ta, M. Zhang, G. Chen, and B. C. Ooi, "A transactional perspective on execute-order-validate blockchains," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 543–557.

[20] B. Ding, L. Kot, and J. Gehrke, "Improving optimistic concurrency control through transaction batching and operation reordering," *Proceedings of the VLDB Endowment*, vol. 12, no. 2, pp. 169–182, 2018.

[21] R. Oliveira and A. Gavrilovska, "In-network compression for accelerating IoT analytics at scale," in *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2023, pp. 15–24.

[22] G. P. Tiwary, E. Stroulia, and A. Srivastava, "Compression of XML and JSON API responses," *IEEE Access*, vol. 9, pp. 57 426–57 439, 2021.