DPPS: A Decentralised Publish-Process-Subscribe Middleware with Verifiable Computations

***Abstract*—In the digital era, supply chains have evolved into complex, Internet of Things (IoT)-driven networks requiring efficient data sharing and processing. Traditional data-sharing methods face challenges in the current predictive supply chain's need for processed data as insights in near-real-time. However, existing messaging models often suffer from the lack of support for automated processes, or the centralisation issues, such as single points of failure and potential bottlenecks. This paper introduces a Decentralised Publish-Process-Subscribe (DPPS) middleware to overcome these limitations, offering byzantine fault-tolerant data processing and verifiable insights without relying on a central entity. DPPS integrates data processing with the pub-sub paradigm, enabling automatic raw data processing in a decentralised manner. The paper presents a decentralised data sharing and processing architecture, a byzantine fault-tolerant verification method, and evaluates the proposed system through a simulated supply chain scenario. Results indicate that DPPS achieves decentralised, verifiable insights with no transaction fees and minimal latency trade-offs, addressing the shortcomings of existing messaging middlewares in predictive supply chains.**

***Index Terms*—Blockchain, IoT applications, Verifiable Computations, Publish-Subscribe, Broker, Smart Contract, Ledger, Supply Chain Monitoring**

I. Introduction

Modern supply chains have evolved from simple systems into complex networks of logistics in the age of digitisation and the expanding Internet of Things (IoT). IoT-driven logistics are widespread across supply chains, encompassing a variety of sensors to guarantee real-time monitoring and effective management. These IoT devices have allowed enormous volumes of data to be collected, which, when evaluated, can provide valuable insights and completely transform supply chain processes [1].

Supply chains need suitable data sharing and processing middleware for transferring this massive inflow of data. However, traditional request-response data-sharing strategies have become increasingly ineffective due to their one-to-one communication nature. In this situation, the publish-subscribe (pub-sub) messaging model has proven to be an invaluable candidate for the data-sharing method due to its loosely coupled one-to-many messaging support [2]. Furthermore, in proactive competitive supply chains nowadays, there is a need for processed data to obtain predictive analysis and enable rapid decision-making rather than working with the raw data [1]. Therefore, real-time data processing is a must-have in predictive supply chains that rely on processed data. Data-sharing middleware must integrate data processing features within its model to extract potential insights on the fly to benefit competitive supply chains. To improve the effectiveness and efficiency of data-driven supply chain activities, PPS, a Publish-Process-Subscribe middleware, was proposed. The system addresses automated data processing features through a centralised broker [3]. PPS provides processed data as insights by orchestrating automated data processing as an integrated part of its data-sharing architecture. Then, it delivers resulting insights to the subscribers instead of delivering raw data.

Although PPS [3] provides the mentioned advantages, it unavoidably fosters the centralisation issue as a severe drawback. Centralised systems are exposed to flaws such as single points of failure, potential bottlenecks, and centralised control. To avoid centralisation flaws like tampering with the initially produced data, some existing messaging middlewares [4] offer blockchain-integrated features like using its tamper-proof distributed ledger to store data. Therefore, no central entity is in charge of recording data, and all the processes are traceable. For instance, Trinity [4] proposes a byzantine fault tolerance distributed messaging model that uses blockchain's immutable ledger to increase data sharing transparency, trust and efficiency. However, the lack of real-time data processing features within Trinity and all the other messaging middleware limits real-time insights for predictive supply chains [4], [5]. Furthermore, offloading computations to subscribers leads to costly and time-consuming data collection and insight extraction for consumers. Besides, if the subscribers send computations to a third-party computing provider, the integrity of the resulting insights is not guaranteed.

Distributing the processes and computations among various network entities and relying on them to provide insights is an alternative approach that can cross-validate insights across the different participating entities to avoid concentrating trust on a single entity. Distributed computation, however, must address the trustworthiness of insights from the participating entities. Many studies used Smart Contracts as a solution to address the automation, integrity and trustworthiness of processes [6]–[8]. Smart contracts are self-executing contracts where the agreed terms between participating entities are written directly into code lines and distributed across a decentralised blockchain network [9]. In a predictive supply chain where enormous data is constantly exchanged, smart contracts are inefficient due to their high associated transaction fees, complexity and the network's inability to run arbitrary computations. Every participant in the supply chain network must run the smart contracts separately, leading to unnecessarily high costs [7]. In blockchain systems like Ethereum, the cost of executing a smart contract is directly related to the computational effort required to perform the contract's operations [5]. Therefore, as

the contract becomes more complex, the more computational resources it requires, and thus, the more transaction fees it produces [10]. For instance, distributing the computations that adopt machine learning techniques and complex algorithms or interactions with other contracts in an active supply chain leads to enormous transaction fees [11].

To address the above challenges, we propose a Decentralised Publish-Process-Subscribe (DPPS) data sharing and processing middleware that is byzantine fault-tolerant. DPPS provides verifiable insights by running automated processes and executing computations cost-efficiently. In this model, shared raw data is automatically processed by PPSManagers in a decentralised manner. PPSManagers, so-called brokers in traditional publish-subscribe architecture, are the entities that integrate data processing features with the broker-based functionalities of the pub-sub paradigm. DPPS delivers verifiable insights by validating computational traces and results at each PPSManager's end using the computational records stored on the blockchain. Our automated data processing middleware dynamically schedules processing tasks based on the perceived load on random computers while guaranteeing the trustworthiness of computations without reliance on any central entity. The contributions of this paper are:

- We design DPPS with built-in byzantine fault-tolerant data processing features where published raw data is distributed and processed on the fly at PPSManagers' end.
- We design a cost-efficient data processing scheme with a dynamic and byzantine fault-tolerant task allocation mechanism where the computations execute at randomly selected computers, and tasks are deallocated when there is a bottleneck risk.
- We develop a byzantine fault-tolerant computation verification method to deliver verifiable insights while guaranteeing the correctness of computations.
- We evaluate DPPS using a proof-of-concept through a simulated supply chain use case. Our evaluation results indicate that we provide verifiable processes and insights decentralised with no transaction fees by an insignificant trade-off in latency.

The rest of the paper is structured as follows: Section II introduces the existing messaging middleware and discusses related works. Section III presents the detailed architecture definition for our proposed Decentralised Publish-Process-Subscribe (DPPS) framework, and Section IV presents the safety and liveness analysis of the system. Section V discusses the evaluations and experimental results, and then Security Analysis is presented in Section VI. The last section concludes the paper with future work.

II. State of the Art and Related Work

The publish-subscribe messaging paradigm is frequently used in modern IoT and business deployments because of its resource efficiency and support for loosely coupled messaging. In this paradigm, subscribers register their interest in an event, or a pattern of occurrences, in systems based on the publish-subscribe interaction and are then asynchronously alerted of events created by publishers. Many variations of the publish-subscribe paradigm have been presented, each tailored to a unique application or network architecture. The common denominator underlying these versions is the complete decoupling of the communication entities in time, space, and synchronisation [12]. MQTT [2] is a widely used publish-subscribe messaging architecture that employs a broker to coordinate communication between data producers and consumers. MQTT requires one request to be sent to a broker, and then the broker forwards the data from the producers to the subscribers. However, this centralised architecture is vulnerable to Byzantine failures. Hence, the shared raw data integrity is not guaranteed through the typical MQTT-based pub-sub [13]–[15].

Authors in [4] propose Trinity as a blockchain-based distributed publish/subscribe broker to address the centralised brokers' issues in IoT and supply chain monitoring applications. Trinity comprises three primary parts: a blockchain network, a broker, and users. The blockchain network controls the system's consensus and durable storage. The broker handles the communication between the blockchain network and customers, and the users are the publishers and subscribers. Although Trinity guarantees data integrity by distributing messages across brokers and using the blockchain ledger's immutability, this framework does not offer data processing within its architecture. As near-real-time insights are vital for modern predictive supply chains, the lack of data processing features on top of shared data is a significant shortage in dispersed IoT applications that serve predictive supply chains [16].

Executing computations on the shared raw data provides data consumers with near-real-time insights that benefit the system differently [17] [18]. Authors in PPS [3] aimed to provide insights by executing on-the-fly computations at their architecture's broker's end. PPSManager is the terminology used in PPS for the data processing integrated pub-sub broker. This entity in PPS controls data processing steps like task allocations and resource scheduling. Computing units are also the executive entities that run computations. As discussed before, one single entity in charge of processes is a single point of failure. Therefore, both the PPSManager and the computing units are susceptible to byzantine failures, and there is no guarantee that the computations and insights are trustworthy in their system. The PPSManager is susceptible to malicious attacks and data tampering. Besides, a malicious entity in charge of the execution may try to tamper with the computations and their results to gain benefits. Therefore, there is a need to verify the correctness of the processes and guarantee that all the computations are run precisely and are reliable. However, there must be no central entity in charge of the verification.

Since supply chain participants use blockchain technology and smart contracts to track and control the supply chain independently, the compliance verification processes can also be automated using the smart contracts [16], [19], [20]. To speed up the compliance verification process, authorities might

convert regulatory requirements into one or more smart contracts. Still, the issue with smart contracts is the inefficiency of required re-computations for verification purposes [21]. Such processes may hold long delays on the network, and the overheads caused by them are inevitable, as well as the costly transaction fees for re-executing computations for verifying purposes [5]. Also, the single entity can claim running the computations while not completing the tasks precisely, which puts the trustworthiness of computational results at risk. In

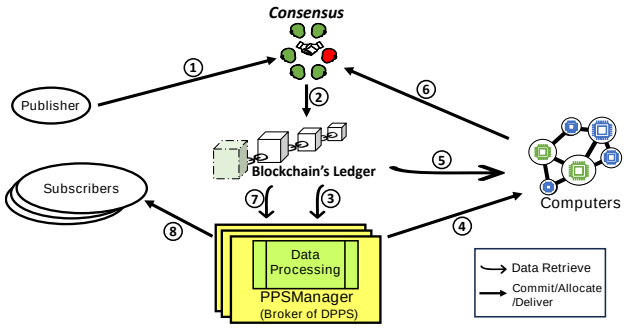

Fig. 1. Overview of the framework

[22], authors created a new version of Python called Verifiable Python (or vPython) by adding the capability for collecting run-time and stack traces to Python (Python3.8). Their study has updated the Python interpreter and run-time to collect stack and run-time traces. They implemented WhistleBlower to detect fake news, and employed a prototype version of vPython to ensure that verifiers have done the validation processes. However, in their study, the central entity controlling the verification and verifiers is susceptible to the single point of failure risks. The node in charge can tamper with the raw data, computations and verification results.

In this study, we aim to design DPPS as a data sharing and processing middleware where no node controls the processes and computations. Decentralising the system raises trust issues regarding the trustworthiness of computations and computational results. Therefore, we provide verifiable insights instead of merely distributing raw data or computational results. We aim to guarantee the correctness of computations and trustworthiness of insights using **verifiable computations** within the proposed DPPS. This is achieved by leveraging the inherent immutability of blockchain technology, which is used to record and secure the processes involved in generating these insights.

III. Architecture Definition and Assumptions

This section discusses the system model of our proposed decentralised PPS middleware, and definitions and assumptions are provided to describe the system.

A. Overview

In DPPS, we propose a publish-subscribe-based data sharing and processing middleware where no centralised entity controls the processes. Figure 1 shows an overview of the system in a domain-based context. Domains in our proposed DPPS architecture are self-contained computational environments, where each domain consists of a PPSManager [3], a consensus node, and potentially a set of publishers and subscribers. Each domain operates semi-independently, facilitating the decentralised nature of the DPPS while also participating in the consensus and processing mechanisms of the system. As shown in Figure 1, for every message published within DPPS, a Byzantine Fault Tolerance mechanism is maintained as step one. In this Byzantine Fault Tolerant (BFT)-based approach, more than two-thirds of validators must reach a consensus before proceeding to the data processing steps. The BFT-based consensus in DPPS leads to instant finality of transactions, and each domain stores its replica of the blockchain as step two. Once consensus is reached, the message is then recorded in the subsequent block as a transaction. Consensus nodes and other block producers create blocks by including transactions from their local mempool in a first-in, first-out (FIFO) fashion. The mempool is a collection of all the transactions that have been submitted to the network but not yet included in a block. Each consensus node maintains its mempool that serves as a waiting area for validating transactions. After validating the transaction, every consensus node alerts the PPSManager in the same domain as step three. Data processing features of PPS [3] are expanded within domains in a decentralised manner. Each PPSManager of each domain checks the transaction existence on the domain's state machine replica and, for validated transactions, initiates data processing [3]. It schedules and allocates tasks on randomly selected computers in step four. Computations are executed off-chain, and the computational traces and results include a signature of the executive computer as computational records. Each computer selected by a PPSManager that has completed computations commits a separate transaction to the blockchain. Other consensus rounds in step six then occur on the computational records committed by each computer. For the computational records, PPSManagers do the same process of checking the transaction's existence on their domain's replica in step seven. PPSManagers verify the integrity of recorded computational results for the same processed raw data and validate the trustworthiness of computations and results.

TABLE I
DPPS APIs To Interact with a Blockchain Network

API	Functionality
DeliverTx(MetaData, Msg)	*if successful;* Return the Block Info *if unsuccessful;* Return Error Code
RetrieveTx(MetaData)	*if successful;* Return the Data *if unsuccessful;* Return Error Code
GetCurrentBlockHeight()	Return the Current Block Height
GetBlock(BlockHeight)	Return the Block using Block Height

B. Architecture Definition and Key Concepts

Figure 2 demonstrates the architecture of our proposed DPPS. Messages are published in the local domain, and a consensus must occur by all the consensus nodes from all domains. Validations occur in the consensus layer for on-chain checkpoints, and all transactions are recorded on the

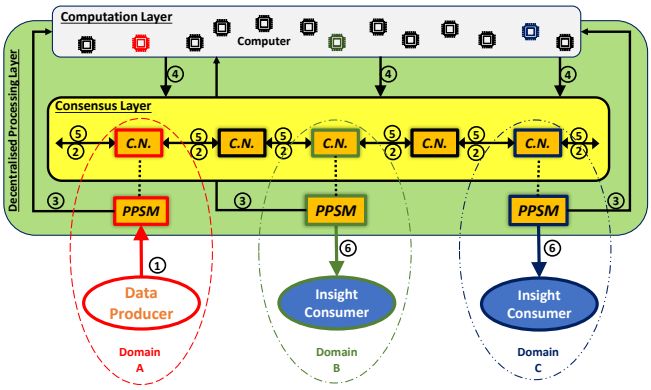

Fig. 2. Architecture of Framework

immutable ledger. PPSManagers run data processing features like task allocations in the processing layer. The connection between the computation layer and the consensus also occurs in the processing layer. The computation layer is for the off-chain event where the computations are executed. The preliminary definitions of the DPPS's architecture key concepts are listed below.

PPSManager: are the brokers of traditional publish-subscribe architecture with integrated data processing features. PPSManagers facilitate the communication between the clients engaging as publishers and subscribers and the blockchain network. Every PPSManager in this domain-based system is connected to a consensus node following the underlying blockchain platform's library. The middleware's interaction with blockchain for the consensus and persistent storage is maintained towards a set of proposed APIs by DPPS. For each published message in a local domain, *DeliverTx* API from Table I delivers the message to the underlying blockchain network and initiates consensus by replicating the message among the other consensus nodes.

Consensus node: are the validators in DPPS architecture where they validate transactions in a BFT-based consensus protocol. After validating a transaction, these nodes initiate the data processing sequence through a specific mechanism facilitated by the **Application Blockchain Interface** (**ABCI**). ABCI provides a standardised method for the consensus nodes to communicate and interact with the application layer, in this case, the PPSManagers. This interaction is achieved through an 'alerting' mechanism, where the consensus nodes use ABCI's set of protocols to signal the PPSManagers. This signalling indicates that a transaction has been validated and is ready for further processing. The use of ABCI ensures a standardised and efficient communication process between the blockchain's consensus layer and the application layer, thereby enhancing the reliability and speed of transaction processing. DPPS's blockchain network includes the consensus protocol, block creation logic, distributed ledger and public key cryptography.

Consensus protocol: is a set of rules and processes for nodes in a decentralised network to agree on the validity of transactions and the state of the shared ledger. Despite the absence of a central authority, consensus nodes collectively make decisions about adding new transactions to the blockchain to ensure all participants in the network have a consistent view. The default consensus protocol obtained for this verification process is the BFT protocol, where most (at least two-thirds) of the network nodes must verify messages. Our system is not limited to the BFT protocols only, but the ability of fault tolerance in a system depends on the consensus protocol.

Public Key Cryptography: The blockchain framework secures transactions using public key cryptography. To participate in the consensus and block formation processes, each consensus node generates a pair of keys and publishes them to the network.

As mentioned, consensus nodes and PPSManagers in a domain are interconnected by an ABCI. Once the raw data reaches consensus, the PPSManager uses block height-related APIs (as detailed in Table I) following the consensus node's alert. Then, the PPSManager shares the metadata and block height with other PPSManagers as a key identifier for each message stored in the blockchain blocks to proceed with task allocations. Then, each PPSManager allocates tasks to a randomly chosen computer from the **Computation Layer**. For task allocations, PPSManagers also share the block height with the computers, and the computers will extract the data stored in each block using the transaction retrieval-related API in Table I for further processes and executions. Using **Hashing** functions, like the SHA-256 hashing algorithm, the integrity of data and blockchain transactions are verified. The same hashing functions are used on the published data by publishers and the computational traces and results by computers.

Computers: are registered identifiable nodes that execute allocated computations within a time-bounded manner [23]. They are selected randomly, and there is no prior knowledge of the tasks or the data they receive. Based on the shared raw data type, PPSManagers provide certain functions to the computers beforehand. For instance, if the input is temporal temperature data, functions related to temperature to calculate the remaining life of a food will be executed by computers. Computers also commit and finalise the computation results and traces to the blockchain and share the block height with the PPSManager. Although these computers are selected randomly, we still verify computations by checking computational results and traces in a BFT-based fashion.

Verifiable Computations: While every PPSManager allocates the same task on a random computer, each computer must execute the same computations on the same data. Therefore, there must be the same computation result and traces per computation. When computers follow the same hashing algorithm, these computational results and traces must contain the same hash. Once the computations are executed and hashes are generated, they are committed to the blockchain and delivered to the consensus layer for at least two-thirds approval. The created block includes the initial data, computational trace and result, and every domain stores a replica on its own blockchain. The PPSManagers will verify that each computer's commits

exist on the blockchain and then validate that all the records match BFT-based. Subscribers of the topic in each domain will receive a validated result from their PPSManager.
There are also some assumptions to maintain the performance of DPPS that are discussed below.

C. Assumptions

Assumption 1: Correct PPSManagers and consensus nodes are always online and do not show byzantine faults.
Assumption 2: No more than one-third of the considered domains are faulty.
Assumption 3: Based on the above assumption, the consensus layer in this architecture implements a Byzantine fault-tolerant atomic broadcast:

- **3.1** All transactions exchanged between correct consensus nodes for validation are delivered precisely the same as those received by other correct consensus nodes.
- **3.2** If a correct consensus node receives a message, all the other consensus nodes must receive the same message in a bounded delay.
- **3.3** Order of the messages is preserved. Therefore, if message m is received before message m' by a correct consensus node, then all the other consensus nodes will receive the same messages in the same order.

Assumption 4: For the consistency of shared messages format, we assume they are all published with JSON message format and JSON payload structure.
Assumption 5: A publisher/subscriber entity is faulty once they publish messages out of the mentioned structure or when they are proved to inject false data into the network. Otherwise, they are a correct entity.
Assumption 6: Honest publishers are always online while auditing a message.
Assumption 7: Based on the incentives and penalties, we assume that the computers will not enter computational results that they are not certain about.
Assumption 8: All the computers use the same hashing algorithm. Therefore, all the same inputs, including the computational functions and input value, must have the same hash.
The next section discusses the system design model using a sequence diagram and detailed description.

D. System Model

Figure 3 shows the sequence diagram of processes happening within DPPS to provide verifiable insights to the subscriber. As mentioned above, we consider a system with a domain in which there is a PPSManager (publish-subscribe broker) $\mathcal{B}$, a validator for consensus $\mathcal{V}$, and possibly a set of publishers $\mathcal{P}$ and subscribers $\mathcal{S}$. For a subscriber $s_m \epsilon \mathcal{S}$ interested in receiving processed data d_x, more than two-thirds of consensus nodes must first validate published raw data r_x [4] in step two, and then each PPSManager of $\mathcal{B}$ must process the data accordingly (step three). The messages that require processing are treated as tasks, and each PPSManager allocates them on an identifiable random computer c_x from the set of registered computers $\mathcal{C}$ (step four). Computers will

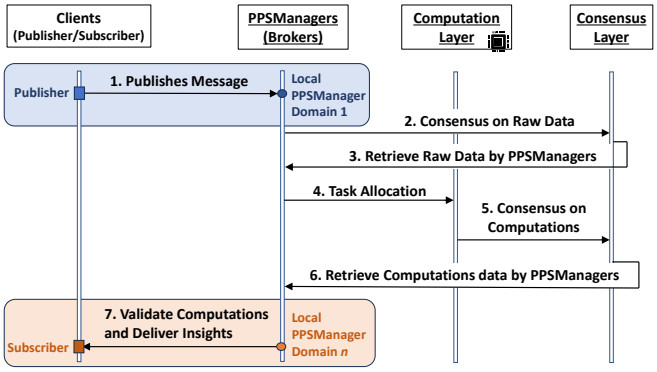

Fig. 3. Sequence Diagram of Processes in DPPS

commit the computational results and traces for another round of consensus in step five. Processed data d_x must also get the consensus of more than two-thirds of consensus nodes $v_i \epsilon \mathcal{V}$. PPSManager of each domain b_m assesses committed computational results and traces to validate the verifiable computations before delivering d_x to its local subscriber s_m. There is another BFT-based validation in PPSManager b_m to ensure more than two-thirds of computational traces and results match (step seven).

IV. Safety and Liveness Analysis

While the liveness property implicitly ensures that the system will advance and that "something positive will always happen," the safety property establishes that "something wrong will never happen" [24]. In this part, we demonstrate the liveliness and safety qualities of DPPS.

A. Safety Properties

Publisher's safety: All correct PPSManagers guarantee the published messages are delivered to topic subscribers.
Messages' safety: All correct subscribers receive the same messages, including the same information and computational results. Also, all the correct subscribers receive the messages in the same order delivered to all other subscribers.
Computations' safety: All correct computers must execute the same functions on the same shared data. Also, all correct computers must use the same hashing algorithm for the verifiable computations' sake on the computational results and traces.

B. Liveness Properties

Message's liveness: Messages published correctly by the correct publishers will be delivered to the message's topic subscribers within a bounded delay.
PPSManager's liveness: PPSManagers of each domain are always online, and once a PPSManager receives a message, all the other consensus nodes must receive the same transaction as the consensus node connected to that PPSManager.
computer's liveness: Once a computer receives a task to

complete, there is a certain time limit for computers to complete the task execution. If there are no results by the limit, a null message will be returned to the PPSManager and the PPSManager must reallocate the task.

V. Implementation and Evaluation

This section discusses the implementation and evaluation of DPPS. The evaluation setup is introduced in Section V-A. Section V-B discusses the baselines and the use case applications. The performance evaluation, considering the end-to-end delay and the network's overhead of decentralised PPS, is presented in Section V-C. Section V-D discusses the security threat models and privacy analysis in detail.

A. Evaluation Setup

We implemented the DPPS using Mosquitto (MQTT) Broker and CometBFT[1] Blockchain. MQTT is one of the widely used publish-subscribe communication protocols in IoT applications. The broker-specific functionalities are implemented on top of MQTT, called PPSManagers in our middleware. For the blockchain, we used CometBFT, an open-source blockchain framework developed on top of the Tendermint Core. CometBFT blockchain framework consists of tools for achieving BFT-based consensus on a distributed network and the creation of blocks. The CometBFT consensus engine allows the application developers to replicate the state of an application across all the CometBFT validators in the network. The state information is fed into the consensus engine using the Application Blockchain Interface (ABCI). To implement the DPPS, we created a broker application using MQTT on top of the CometBFT blockchain framework. We used ABCI to bridge the MQTT application with the blockchain framework. The CometBFT framework uses Byzantine Fault Tolerance (BFT) consensus protocol as mentioned, which means 2/3 of the devices in the network must approve the transactions. When the majority of the devices in the network approve the transaction, the CometBFT framework adds the transaction in a block. All the application-specific software was implemented in NodeJS. We used Raspberry Pi 4 model B for evaluations, and we evaluated our setup on a Raspberry Pi test bed, including four Raspberry Pis. Each Pi is used as a separate domain and runs a validator and a PPSManager.

B. Use Case Application

To evaluate the performance of our DPPS middleware that delivers verifiable insights, we use PPS [3], an existing centralised data-sharing and processing middleware, as one baseline and Trinity [4] as another baseline. In Trinity, the shared data is not processed, but the message is distributed with multiple brokers and written on the immutable ledger. In this experiment, we evaluate the system's functionality using the real-life use case of the Food Supply Chain (FSC) used in PPS [3]. In that scenario, there was a need for near-real-time insights regarding the remaining life of the perishable products as a predictive supply chain to avoid food spoilage, food loss or human contamination. Computations related to this scenario might be simple temperature threshold calculations or complex algorithms to estimate spoilage based on previous records and predict the most efficient new dropping place. The publishers of these temporal messages can be the IoT logistics used in a distribution entity of the supply chain, like the temperature and humidity sensors and GPS data. Subscribers can also be the same truck, the retail store that is the best dropping point, and all the other supply chain participants. In PPS, the authors showed that by integrating data processing features within the data sharing middleware, they trade-off insights with only minor added execution time instead of delivering raw data. However, the limitation of their proposed system is that the produced results are susceptible to malicious tampers. For instance, the entity in charge of executing the computations might tamper with the functions and processes to gain malicious benefits from the results. Both PPSManagers and computers, so-called computing units in PPS, are susceptible to single point of failure concerns like deliberate malicious actions or even unintentional faults that might happen during the computations. As food quality and its expiration and spoilage directly impact human lives, it is necessary to ensure that the computations are executed precisely and results are trustworthy.

C. Performance Evaluation

We compare and contrast the benefits of our proposed DPPS with a smart contract-based framework and the mentioned baselines. Table II declares the benefits of DPPS over a middleware that uses Smart Contracts as a tool for automated processes and verification purposes. The inability to support arbitrary computations and the need to execute them separately on each machine leads to considerable excessive fees that are unnecessary in a supply chain. The cost of deploying a new smart contract for every process and computation is higher than calling a function from a contract that is already deployed [10]. Therefore, as shown in Table II, to compare the best case of using the smart contracts we consider the transaction fee of a function call that equals 0.000131 ETH. Considering an IoT application in a predictive supply chain where the message arrival rate is high, the accumulated transaction fees for each supply chain participant are too high, proving the benefits of DPPS outweigh Smart Contract-based systems. Assuming a supply chain in which ten sensors publish one message per ten seconds (six messages per minute), the cost of one-hour data processing for five domains equals the number of executions multiplied by the transaction fee:
$(10\times6\times60\times5) \times 0.000131$ ETH.
The hourly cost of running supply chain with smart contracts

TABLE II
Data Processing Costs for *n* Nodes per Transaction

Data Processing	Number of Computations	Transaction Fee
Smart Contracts	n	0.000131 ETH
DPPS	$1 <$ computations $<$ n	none

[1]https://github.com/cometbft

in this scenario equals 2.358 ETH ($\simeq$ 4927.49 USD).
To prove that DPPS offers the same decentrality as a smart contract-based system, we need to measure the Gini Coefficient (G) metrics of the consensus nodes and the Fairness Index (F(X)) of PPSManagers in DPPS [25].

$$G = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} |P_i - P_j|}{2 \cdot N \sum_{j=1}^{N} P_j} \tag{1}$$

$$F(X) = \frac{(\sum_{i=1}^{N} P_i)^2}{\sum_{i=1}^{N} P_i^{2'}} \tag{2}$$

The Gini Coefficient is used to measure the inequality of the distributions [26]. The Fairness Index aims to show the fairness level of nodes' decisions and control in allocations [27].

Equations 1 and 2 calculate the Gini Coefficient and Fairness Index, respectively [28]. N is the total number of messages in both equations, and P_i is the fraction of messages that the consensus nodes and PPSManagers receive. In a centralised environment like PPS [3], where one node receives all the messages and does the processes solely, the Gini Coefficient is close to one, and the Fairness Index is close to zero. However, in DPPS, as all the consensus nodes receive the same shared data and the consensus occurs decentralised, the Gini Coefficient equals zero, which means the system is totally decentralised. The Fairness Index for PPSManagers in DPPS also equals one, which quantifies the data processing is completely distributed and the system is decentralised.
Data processing is an absent feature in Trinity [4] as a decentralised data-sharing middleware. Our evaluation results in Figure 4 indicate that with an insignificant rise in delay, we can offer verifiable insights to subscribers rather than transferring the raw data.

D. End-to-End Delay

Figure 4 shows the end-to-end delay results from published raw data to the subscriber's end delivery. As mentioned earlier, there are three different baselines investigated in this experiment. To make it a near-real-life experiment, different numbers of messages are published within each baseline, with the lowest rate starting from one message per ten Seconds to 2 messages per Second. The duration of executing computations in both PPS and DPPS that integrate data processing features in their architecture is equal (251 Milliseconds) to make it an even comparison. Also, both Trinity and DPPS in Figure 4 have four consensus nodes to ensure the minimum required nodes for meeting the BFT-based condition. PPS, which is a centralised data-sharing and processing middleware, tends to show the same trend in end-to-end delays as the number of published messages increases. There are slight increases in Milliseconds for delays, but they are negligible in the scale of Seconds. Trinity also shows the same trend as the number of messages increases. There is a slight change in Milliseconds as the message arrival rate increases, and this delay is caused by the blockchain-based immutability and ordering through BFT consensus [4]. However, there are no data processing features or added computational delays to Trinity. DPPS on the other hand has a rise in delays as the message arrival

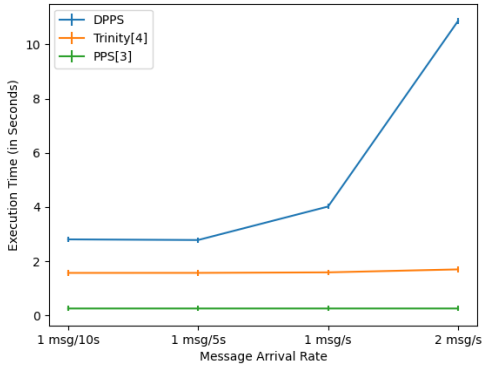

Fig. 4. End-to-End Delay Comparison

rate increases, and this rate is due to the throughput that the PPSManagers and the consensus nodes face as the input increases. Each PPSManager and computer must retrieve data from the blocks. Therefore, it takes time for each transaction to be committed and become available in each domain's replica. Also, as the CometBFT forms block every second, there is a heads-up regarding the mempool connection for transactions. CometBFT sequentially processes an incoming transaction, and if the processing does not return any error, the transaction is accepted into the mempool, and CometBFT starts gossiping about it.
Checking the transaction state should be reset to the latest committed state at the end of every Commit. At the end of the consensus instance, CometBFT locks the mempool and flushes the mempool connection before calling Commit. This ensures that all pending RetrieveTx (as mentioned in Table I) calls are responded to, and no new ones can begin. Therefore, if transactions are committed once the mempool is locked, there might be a loss in the shared messages that are not finalised in a block. Table III compares the features of DPPS with the existing data sharing and processing middleware and declares the contributions of DPPS. For both DPPS and Smart Contract-based systems like [5], the computational overhead due to the data processing facilities is high. For n domains in a system, we still need n PPSManagers to have the same degree of decentralisation for both DPPS and a smart contract-based framework. For each domain, if the computations are executed once by a smart contract, there are n computations executed to gain insights. As mentioned earlier in Table II, the number of computations needed to gain verifiable insights by DPPS is $1 < computations < n$. Therefore, there are nearly the same overheads on the PPSManagers for facilitating the data processing using either smart contracts or DPPS. Although there are some slight additional delays for the final validation and verification that occur at the PPSManager in DPPS, the total overhead difference with smart contracts is insignificant. Besides, there is a huge monetary difference in system execution as there are no transaction fees in DPPS compared to costly smart contracts.
Furthermore, Table III shows that the DPPS benefits of au-

TABLE III
DPPS COMPARISON WITH THE STATE OF THE ART

System	Decentralised	Data Processing	Transaction Cost	Verifiable Computations	Computational Overhead
DPPS	✓	✓	None	✓	High
Smart Contract-based (e.g. [5])	✓	✓	High	✓	High
PPS [3]	×	✓	-	×	Low
Trinity [4]	✓	×	None	×	Medium

tomated decentralised data processing with verifiable insights outweigh its increase in end-to-end delay compared to centralised systems([3]) or decentralised messaging models([4]). In DPPS, the trustworthiness of delivered insights is high as PPSManagers ensure the integrity of computations among all the records on the ledger. Consequently, there is a rise in overhead for this validation. Still, the total added latency is relatively small due to saving the computational effort for insights extraction or verification from the subscribers' end.

VI. SECURITY ANALYSIS

This section briefly investigates the security risks and threat models and discusses our DPPS's resistance towards them.

Threat Model: Attack scenarios and malicious activities that aim to tamper the data, processes, computational results, and data sharing are listed below.

Registering Multiple computers: A malicious participant could aim to register multiple computers to increase the chance of receiving the computations. Through this malicious behaviour, they aim to tamper with the shared data or the computational processes to benefit themselves by generating wrong results. To counter this, our decentralised PPS decreases the chance of benefiting from registering many computing nodes by first randomly selecting the computers and then restricting the computer selection pool to only available nodes. Also, as discussed earlier in Section IV-B as the computer's liveness properties, computations are bound to a certain time limit for execution. Furthermore, all the registered computing units must be identifiable by a unique public key. Besides, we also avoid the tampering opportunity by choosing multiple computers to do the exact executions and performing a validation process at the PPSManager before delivering insights to guarantee the trustworthiness and dedicate penalties to malicious computers.

Man-in-the-middle attacks and **Data Tampering**: PPSManagers in our DPPS middleware act as brokers that, besides receiving messages from publishers, investigate the topics and schedule and allocate tasks and receive the computational results to publish them to subscribers. Decentralisation and replicating messages to different PPSManagers through different domains prevent malicious behaviours where an attacker intercepts and manipulates messages between two parties to reveal or potentially tamper with sensitive information. We encrypt the communications between nodes to prevent eavesdropping and use digital signatures to detect tampering.

For **Sybil attacks**, a malicious actor creates multiple fake identities to manipulate the system and gain control over the nodes, like PPSManagers. As the PPSManagers are expected to provide the same processes on each incoming message, they must generate the same traces for the same input. These traces can also be periodically checked in a BFT-based fashion to ensure the integrity of PPSManagers.

Resource exhaustion attacks happen when an attacker tries to exhaust resources like computers to become unavailable or crash. Also, as mentioned before, with the time-bound discussed in Section IV-B for the computer's liveness, DPPS prevents excessive computer usage and increases the system's resilience to node failures.

Lazy Attacks also happen when a computer claims to provide computational results while not running the computations precisely. Using verifiable computations for validating computational traces and results, DPPS avoids lazy attacks and sets penalties for malicious nodes that do not match the traces. As mentioned earlier, PPSManagers are the entities that forward input-related functions to random computers for execution. Therefore, PPSManagers also can periodically examine a computer's integrity by executing computations at their end and comparing traces in a BFT-based fashion.

VII. CONCLUSION

We proposed DPPS, a Decentralised Publish-Process-Subscribe framework architecture, in which every shared raw data is processed decentralised, and the computations executed on raw data to extract insights are verifiable. Our evaluation results show that we provide insights in a highly trusted fashion with the minimum number of computations in favour of a predictive supply chain with merely a negligible rise in delay. For future work, we focus on privacy-preserving techniques in which we guarantee stakeholders that the data are only shared with whom it is meant to be and the computations and computational results do not risk privacy concerns of the data or data producer.

REFERENCES

[1] L. Cui, M. Gao, J. Dai, and J. Mou, "Improving supply chain collaboration through operational excellence approaches: an iot perspective," *Industrial Management & Data Systems*, vol. 122, no. 3, pp. 565–591, 2022.

[2] O. Standard, "Mqtt version 3.1. 1," *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, vol. 1, 2014.

[3] A. Jabbari, G. Ramachandran, and S. Mailik, "Pps: A publish-process-subscribe middleware for predictive supply chains," in *Proceedings of the 20th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2023), Melbourne, Australia, November, 2023*. Springer, 2023.

[4] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal, "Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 227–235.

[5] L. Li, T. Zhang, G. Sun, D. Jin, and N. Li, "A fair, verifiable and privacy-protecting data outsourcing transaction scheme based on smart contracts," *IEEE Access*, vol. 10, pp. 106 873–106 885, 2022.

[6] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart contract-based product traceability system in the supply chain scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019.

[7] T. M. Hewa, Y. Hu, M. Liyanage, S. S. Kanhare, and M. Ylianttila, "Survey on blockchain-based smart contracts: Technical aspects and future research," *IEEE Access*, vol. 9, pp. 87 643–87 662, 2021.

[8] Y. Wu, J. Li, J. Zhou, S. Luo, and L. Song, "Evolution process and supply chain adaptation of smart contracts in blockchain," *Journal of Mathematics*, vol. 2022, pp. 1–13, 2022.

[9] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[10] S. Mercan, M. Cebe, E. Tekiner, K. Akkaya, M. Chang, and S. Uluagac, "A cost-efficient iot forensics framework with blockchain," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–5.

[11] A. Di Sorbo, S. Laudanna, A. Vacca, C. A. Visaggio, and G. Canfora, "Profiling gas consumption in solidity smart contracts," *Journal of Systems and Software*, vol. 186, p. 111193, 2022.

[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[13] E. Vinka, L. Johansson, and A. Kafka, "Cloudkarafka, 2019," *URL https://www. cloudkarafka. com*.

[14] D. Scott, V. Gamov, and D. Klein, *Kafka in Action*. Simon and Schuster, 2022.

[15] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[16] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. Leung, "Blockchain and machine learning for communications and networking systems," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1392–1431, 2020.

[17] J. Will, L. Thamsen, J. Bader, D. Scheinert, and O. Kao, "Get your memory right: The crispy resource allocation assistant for large-scale data processing," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2022, pp. 58–66.

[18] L. Tan, K. Yu, A. K. Bashir, X. Cheng, F. Ming, L. Zhao, and X. Zhou, "Toward real-time and efficient cardiovascular monitoring for covid-19 patients by 5g-enabled wearable medical devices: A deep learning approach," *Neural Computing and Applications*, pp. 1–14, 2021.

[19] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2021, pp. 1–3.

[20] T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, "Blockchains everywhere-a use-case of blockchains in the pharma supply-chain," in *2017 IFIP/IEEE symposium on integrated network and service management (IM)*. IEEE, 2017, pp. 772–777.

[21] A. Mamageishvili and J. C. Schlegel, "Optimal smart contracts with costly verification," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–8.

[22] G. Ramachandran, D. Nemeth, D. Neville, D. Zhelezov, A. Yalçin, O. Fohrmann, and B. Krishnamachari, "Whistleblower: Towards a decentralized and open platform for spotting fake news," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 154–161.

[23] A. Jabbari, F. Masoumiyan, S. Hu, M. Tang, and Y.-C. Tian, "A cost-efficient resource provisioning and scheduling approach for deadline-sensitive mapreduce computations in cloud environment," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 600–608.

[24] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed computing*, vol. 2, no. 3, pp. 117–126, 1987.

[25] J. W. Heo, G. Ramachandran, and R. Jurdak, "Ppos: Practical proof of storage for blockchain full nodes," in *The Proceedings of the 5th IEEE International Conference on Blockchain and Cryptocurrency, Dubai, UAE, May, 2023*. Institute of Electrical and Electronics Engineers Inc., 2023.

[26] M. Liu, F. R. Yu, Y. Teng, V. C. Leung, and M. Song, "Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3559–3570, 2019.

[27] R. K. Jain, D.-M. W. Chiu, W. R. Hawe *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, vol. 21, 1984.

[28] S. P. Gochhayat, S. Shetty, R. Mukkamala, P. Foytik, G. A. Kamhoua, and L. Njilla, "Measuring decentrality in blockchain based systems," *IEEE Access*, vol. 8, pp. 178 372–178 390, 2020.