EAI Endorsed Transactions
on Energy Web
Research Article EAI.EU

Lightweight and scalable IoT architecture for supply chain: integration of Hyperledger Fabric and Node-RED delegated nodes

Yassin Elgountery[1*], Mohamed Aghroud[1], Meryem Lasaad[2] and Mohmed Oualla[1]

[1]Department of Computer Science, CVDIS team, CESTec laboratory, FST Errachidia, Moulay Ismail University, Errachidia, Morocco
[2]Department of Computer Science, GLISI team, CESTec laboratory, FST Errachidia, Moulay Ismail University, Errachidia, Morocco

Abstract

Monitoring environmental conditions such as temperature and humidity across logistics chains is a major concern for the pharmaceutical and agri-food sectors. Running full blockchain nodes directly on constrained IoT devices is still unrealistic because these devices have limited computing power and energy capacity. This paper presents a decentralized hybrid architecture built on Hyperledger Fabric, in which the cryptographic workload is delegated to intermediate edge gateways orchestrated with Node-RED. An Express.js middleware API mediates transactions between IoT flows and the distributed ledger. Experimental tests using a JavaScript harness show a maximum throughput of 657 transactions per second (TPS) with 1,000 simultaneous sensors, an average latency of 853 millisecond (ms), and a 100% success rate. These results support the idea that a delegated-node architecture can combine lightweight edge operation, immutable traceability, and horizontal scalability.

Keywords: Internet of Things, blockchain, architecture, Hyperledger Fabric, edge computing, Node-RED, delegated node, supply chain

Received on 03 August 2026, accepted on 09 November 2026, published on 22 November 2026

Copyright © 2026 Yassin Elgountery *et al.*, licensed to EAI. This is an open access article distributed under the terms of the CC BY-NC-SA 4.0, which permits copying, redistributing, remixing, transformation, and building upon the material in any medium so long as the original work is properly cited.

doi: 10.4108/eetiot.14312

1. Introduction

Perishable products require continuous monitoring to preserve their quality and safety. IoT sensors facilitate this process by measuring temperature, humidity, and gas composition in real time. These data play an essential role in cold chain management [1]. Recent reviews have highlighted the growing use of IoT, blockchain, and predictive analytics to improve the monitoring and management of cold chains in the dairy, pharmaceutical, and food sectors [2]. Blockchain further strengthens this process through data immutability and transparency. It ensures product traceability from the source to the final consumer [1]. Consequently, recent studies have combined AI, IoT, and blockchain to enhance traceability in perishable goods logistics. This integration relies on real-time monitoring, predictive analytics, and the immutable recording of events [3].

However, integrating blockchain with IoT presents several challenges. First, IoT devices often have limited computational resources. This restricts their ability to perform cryptographic operations and participate in consensus mechanisms [4], [5]. Second, the proof-of-work (PoW) mechanism remains unsuitable for IoT environments owing to its high energy consumption and latency [6]. Furthermore, even permissioned blockchain

*Corresponding author. Email: y.elgountery@edu.umi.ac.ma

platforms, such as Hyperledger Fabric, depend on available hardware resources. Evaluations conducted on heterogeneous devices show that performance varies according to computational capacity. As the workload increases, latency increases, and throughput decreases [7]. Moreover, transaction endorsement, ordering, and validation can become performance bottlenecks under heavy workloads [8]. Therefore, achieving scalability in blockchain–IoT architectures, particularly for applications requiring real-time processing, remains a major challenge [9].

This study aimed to design and evaluate an IoT–blockchain architecture for cold chain traceability. The proposed architecture seeks to (i) keep IoT sensors lightweight, (ii) ensure data immutability and auditability through a permissioned distributed ledger, and (iii) support large-scale environmental data streams. To achieve these objectives, we adopted an edge computing approach. Edge devices locally process time-sensitive operations, whereas more resource-intensive tasks are offloaded. This approach reduces latency compared to centralized blockchain processing [10].

This work makes four main contributions, which are divided into two categories: architectural and experimental contributions.

Architectural contributions: First, we propose a lightweight three-layer architecture for IoT–blockchain integration, comprising the IoT/Edge, Middleware, and Blockchain layers. This architecture separates sensor data acquisition from blockchain transaction management, thereby reducing the responsibilities assigned to resource-constrained IoT devices. A key design characteristic is the delegation of blockchain-related operations and associated cryptographic operations to the middleware. This approach reduces the need for IoT devices to handle blockchain-specific processes and cryptographic tasks directly. Second, we designed an integration mechanism based on Node-RED and Express.js. Node-RED manages sensor data collection and event orchestration, while the Express.js gateway acts as an intermediary between the IoT environment and Hyperledger Fabric. It provides an interface for transaction submission and supports the management of incoming sensor data.

Experimental contributions: Third, we conduct a quantitative evaluation of the proposed architecture under different workload conditions. This evaluation examines the transaction throughput (TPS), latency, and resource consumption. It includes a comparison of different transaction submission approaches, as well as load tests involving an increasing number of simultaneous simulated sensors, ranging from 10 to 1,000. It also investigates the impact of transaction batching and ledger database selection on system performance. Fourth, we evaluate system resilience through fault-injection experiments involving failures of key components. These experiments examine data loss, service recovery time, and system availability under the tested conditions. Based on the experimental observations, we also analyze the architecture's limitations, including potential bottlenecks and single points of failure (SPOFs). Finally, we discuss several possible improvements, such as a microservices-based deployment and the integration of lightweight embedded artificial intelligence techniques.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature and related studies. Section 3 presents the problem statement and establishes the research positioning of the proposed approach. Section 4 describes the proposed architecture, while section 5 details its technical implementation. Section 6 introduces the performance evaluation methodology. Section 7 presents the experimental results. Section 8 provides a comparative analysis and discusses the results. Section 9 addresses the limitations of the proposed architecture. Section 10 outlines the future research directions and potential improvements. Finally, section 11 concludes the paper by summarizing its main contributions.

2. Related work

2.1. Blockchain and IoT in the supply chain

The integration of blockchain and IoT for logistics traceability has received growing research attention in recent years. Karaduman and Gülhas [11] proposed a systematic review of blockchain-enabled supply chain management architectures, showing that permissioned distributed ledgers offer a better security/performance trade-off than public chains in an industrial context. Valapu et al. [12] presented a blockchain infrastructure with a reward mechanism for 3D integrated circuit traceability, demonstrating that economic incentives improve the reliability of submitted data (Garbage In, Garbage Out problem [13]). Truong et al. [14] proposed an agricultural product traceability system combining LoRa-based IoT sensors with a private Ethereum blockchain (GoQuorum) to monitor environmental and geographical data during transportation. Although the prototype demonstrated the feasibility of real-time monitoring, its evaluation was limited to a 1.6 km route, leaving its scalability insufficiently explored. Similarly, Sharma et al. [15] integrated Ethereum, IoT sensors, GPS tracking, and smart contracts to improve transparency and traceability in the diamond supply chain. Despite its potential for supporting ethical sourcing and fraud prevention, the system faces challenges related to network congestion, transaction throughput, and scalability.

Architecturally, Liu, Feng and Sakurai [16] leverage the multi-channel mechanism of Hyperledger Fabric to isolate sensitive data in dedicated channels, which improves inter-organizational confidentiality in the Internet of Vehicles. This transactional partitioning approach is transferable to multi-actor supply chains (producers, carriers, distributors).

2.2. Food and pharmaceutical traceability frameworks

In the cold chain, Zhou, Feng and Sakurai [17] propose a food traceability system based on Hyperledger Fabric incorporating an incentive trust mechanism. The authors show that the combination of the tamper resistance provided by blockchain with reputation scores calculated by regulatory authorities and buyers helps reduce the GIGO (Garbage In, Garbage Out) problem. Their evaluation using Hyperledger Caliper reveals stable throughput and low latency under high transactional load [17].

In addition, a study on midstream oil logistics [18] shows how a gateway/SDK layer between IoT devices (temperature and pressure sensors) and the Fabric network. This intermediate layer performs protocol translation (MQTT, OPC UA, and HTTP [19]), semantic alignment of measurements, and their initial validation before on-chain submission. The authors note that despite the variety of data flows, the use of standardized chaincodes ensures the uniformity of the immutable data model [18].

2.3. Integration of edge computing and zero-trust security

The study [20] analyzed the convergence of AI, blockchain, and edge computing for zero-trust security in IoT. The authors identified three major challenges: (i) scalability–massively distributed IoT networks generate data volumes that saturate traditional blockchains; (ii) resource constraints–heavy cryptographic protocols are incompatible with sensor capabilities; and (iii) interoperability–the lack of uniform standards complicates cross-sector integration.

Edge computing is presented as a structured response: by moving blockchain verification and AI analysis to the edge node level, it minimizes latency, reduces single point of failure risks, and preserves end-device resources [20]. This vision aligns with the delegated node architecture proposed in our work, where Node-RED acts as a trusted edge node.

2.4. Lightweight blockchain solutions for the IoT

Given the limitations of Fabric on constrained hardware, recent work has explored alternative consensus mechanisms. A BFT-DAG (Byzantine fault tolerant directed acyclic graph) architecture proposed in [21] achieves 49,039 TPS with a latency of 433 ms in realistic IIoT scenarios. Although this approach performs well, it requires a complete redesign of the consensus protocol. By contrast, our work maintains the mature ecosystem of Hyperledger Fabric while reducing the load on sensors through edge delegation, an approach that is more pragmatic for industrial deployment.

Abbasi et al. [22] introduced proof-of-resource (PoR), a lightweight consensus mechanism that selects validators based on their available resources. Simulations involving 15 IoT devices demonstrated its potential to reduce computational requirements. However, its scalability, throughput, and resilience to complex attacks require further investigation. In contrast, our architecture does not introduce a new consensus mechanism. Instead, it reduces the processing burden on constrained IoT devices by delegating the blockchain interactions to Node-RED gateways.

A further study on IoT blockchain scalability [23] distinguishes three optimization strategies: (i) the use of private/permissioned blockchains; (ii) consensus localization; and (iii) off-chain solutions (payment channels, sidechains). Our architecture combines strategies (i) and (ii) by using Fabric (permissioned) and localizing transaction signing at the level of Node-RED gateways.

2.5. Comparison with existing IoT–blockchain integration architectures

Three architectural approaches are closely related to our proposed solution. IBM's Watson IoT-based solution integrates Node-RED with Hyperledger Fabric through REST interfaces [24]. However, it relies on simulated devices and provides no performance evaluation in the documentation examined. Duman and Aydoğan [25] proposed a cold-chain architecture in which an ESP32 gateway forwards BLE sensor readings through an MQTT broker to a Node.js-based REST API, which invokes the chaincode. This architecture incorporates JWT-based device authentication and server-side management of Fabric identities. The authors evaluated ledger performance using Hyperledger Caliper. However, each reading is submitted individually as a transaction, with no aggregation or filtering mechanism at the gateway level. In contrast, Pajooh et al. [26] adopted a different approach by adapting Fabric to the ARM64 architecture. Their solution enables Raspberry Pi cluster heads to host Fabric peers, while ordinary sensors are authenticated locally. Their results showed that these nodes can process approximately 12 transactions per second, a throughput considerably lower than that achieved by a desktop-based configuration.

By contrast, our architecture maintains a lightweight sensor operation by introducing a Node-RED gateway between the edge layer and the Express.js middleware. This gateway performs data aggregation and filtering, as well as signing on behalf of the sensors, before forwarding the readings to the middleware. The latter then submits transactions to the Hyperledger Fabric network using the Fabric SDK and Wallet. Finally, unlike the aforementioned studies, our evaluation covers the entire processing pipeline under workloads ranging from 10 to 1,000 concurrent sensors, rather than focusing solely on blockchain ledger performance.

3. Problem statement and positioning

3.1. Constraints of IoT devices

IoT devices used in supply chains face several limitations related to computational resources, energy consumption, and connectivity. Their limited memory and processing power restrict their ability to perform cryptographic operations and consensus mechanisms [27], [26]. Moreover, battery-powered operation makes computationally intensive tasks particularly energy-consuming [26], [28]. Finally, intermittent wireless connections and low bandwidth complicate communication with blockchain networks [27], [28]. These constraints justify the use of edge gateways capable of managing blockchain interactions on behalf of IoT sensors.

3.2. Limitations of existing approaches

Direct on-chain approaches [17,18], where each sensor reading is immediately recorded on the ledger, show three main drawbacks: (1) network saturation: a maritime container equipped with 50 sensors sampling every 10 seconds generates 18,000 transactions per hour. Multiplied by a fleet, this volume quickly exhausts the capacity of Fabric orderers [16-18]. (2) Cryptographic overhead: each transaction requires an X.509 signature, an endorsement proposal, a broadcast to peers, and a block commit. These operations are impossible on microcontrollers without a cryptographic coprocessor [18-20]. (3) Storage cost: blockchain immutability implies a monotonic growth of the ledger. Storing high-frequency raw measurements makes the ledger unmanageable in the long term [5-16-17].

Despite their contributions, each of these approaches has limitations that motivate our architectural design. IBM's solution [24] relies on proprietary Watson IoT services and Fabric integration based on the Hyperledger Composer. Furthermore, its documented examples use simulated devices and provide no performance evaluation. Duman and Aydoğan [25] submitted each sensor reading as an individual transaction through a centralized API. They also centralize Fabric identity management within this API layer and evaluate ledger performance in isolation using Hyperledger Caliper without considering the entire sensor-to-ledger pipeline under increasing sensor workloads. Pajooh et al. [26] relied on deploying full Fabric peers on Raspberry Pi-class cluster head nodes, which imposed a considerable computational burden at the network edge. Their measurements indicated a throughput of approximately 12 transactions per second on these nodes, compared with approximately 58 transactions per second on virtual machines. Moreover, their approach does not address resource-constrained microcontroller-based sensors.

Other studies also reveal limitations from different perspectives. Truong et al. [14] evaluated their LoRa–GoQuorum agricultural traceability system over a limited 1.6 km route, leaving its scalability under larger deployments insufficiently explored. Abbasi et al. [22] validated their PoR consensus mechanism through simulations involving only 15 IoT devices, leaving its performance at larger scales and its resilience to complex attacks open to further investigation. Similarly, Sharma et al. [15] focused on blockchain-based transparency in the diamond supply chain, while challenges related to network congestion, transaction throughput, and latency remain relevant to large-scale deployments.

Overall, although existing studies have made significant contributions to supply chain traceability, consensus mechanism optimization, and blockchain–IoT integration, they do not jointly address the different aspects considered in this research. Our architecture aims to bridge this gap by combining a Node-RED-based middleware, the delegation of blockchain interactions to intermediate nodes, and off-chain data management. This approach separates sensor data acquisition from blockchain transaction processing and is subjected to an end-to-end experimental evaluation under workloads of up to 1,000 sensors.

3.3. Research hypothesis

H_1 - An architecture based on delegated nodes, where edge gateways (Node-RED) aggregate, filter, and sign IoT data before submission to Hyperledger Fabric, makes it possible to achieve a transactional throughput exceeding 500 TPS with a latency of less than 1 second, while maintaining zero footprint on the sensors.

4. Proposed architecture

4.1. Overview and design principles

As illustrated in Figure 1, the proposed architecture follows a separation of concerns across three functional layers, consistent with distributed systems best practices [29].

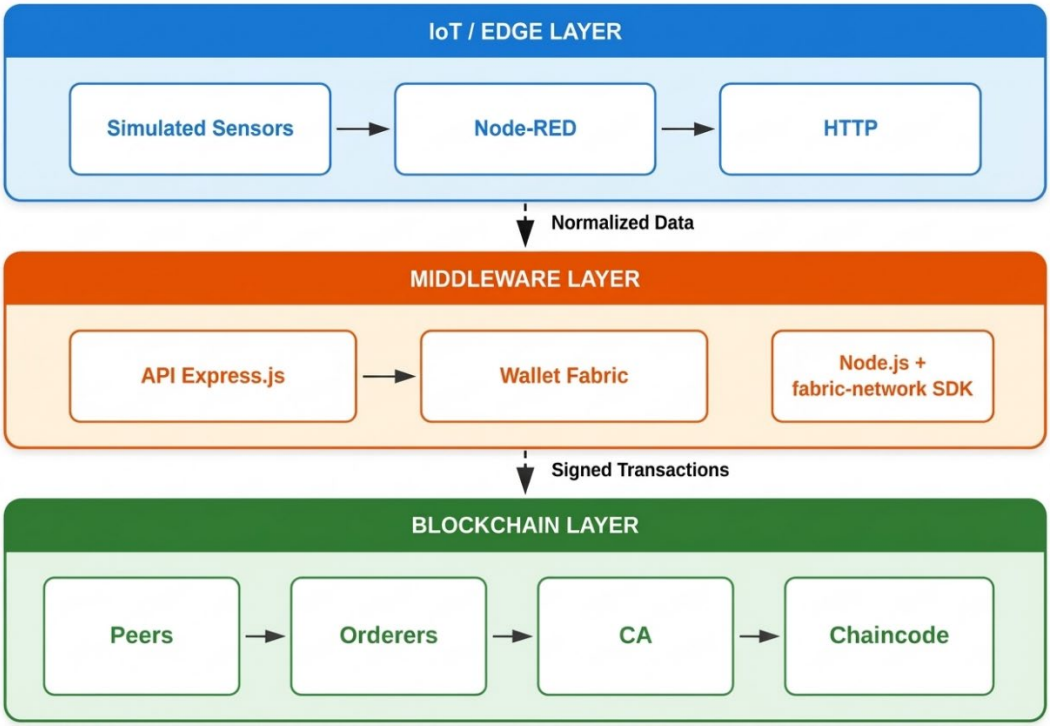

Figure 1. Architecture functional layers

Table 1 summarizes the components within each layer, allocation of responsibilities, and corresponding technologies employed.

Table 1. Distribution of the functional layers of the architecture

Layer	Components	Responsibility	Technology
IoT / Edge	Simulated sensors, Node-RED	Acquisition, buffering, filtering, normalization	HTTP, Node-RED
Middleware	API Express.js, Wallet Fabric	Cryptographic signing, transactional mediation, identity management	Node.js, fabric-network SDK
Blockchain	Peers, Orderers, CA, Chaincode	Consensus, immutability, auditability	Hyperledger Fabric 2.x

4.2. Peripheral layer (IoT / Edge layer)

This layer materializes the concept of delegated nodes. It is composed of temperature and humidity sensors (simulated for this study) and a Node-RED instance. This layer acts as a buffer. Instead of requiring each sensor to cryptographically sign its transactions, Node-RED aggregates the events and transmits them via the HTTP/REST protocol.

The Node-RED instance can perform several functions: (i) temporal aggregation: event windowing (batching) to reduce the number of blockchain transactions. (ii) Plausibility filtering: rejection of outliers (e.g., temperature > 80 °C or < −40 °C in a standard container). (iii) Semantic enrichment: addition of node_id, ISO 8601 timestamp, geolocation (lat/lng), and container identifier. (iv) Local buffering: SQLite/Redis persistence in case of network disconnection to the middleware, ensuring traceability continuity even in areas without connectivity [30].

4.3. Middleware layer (API gateway layer)

The middleware API, developed in Node.js [31] using the Express.js framework, serves as a critical integration bridge. It holds the identity wallet (Wallet) of the organization operating the edge gateways (appUser). Upon receiving a JSON payload from Node-RED (via HTTP POST), the API: (1) authenticates the request; (2) validates the JSON schema (temperature, humidity, units, timestamps); (3) signs the transaction through the fabric-network SDK using the X.509 identity stored in the Wallet; (4) submits the transaction proposal to the organization's endorsing peers; and (5) returns the transaction identifier and validation status to Node-RED.

This approach centralizes the cryptographic complexity: only the API Gateway possesses the private keys and

interacts with the Fabric Certificate Authority (CA) [17]. The sensors and Node-RED handle only lightweight payloads.

4.4. Blockchain layer (Ledger layer)

The Hyperledger Fabric 2.x [32] network was configured with two organizations: Org1-IoT (owners of sensors and containers) and Org2-Logistics (logistics operators and carriers).

Chaincode (Smart Contract) [32]–Written in JavaScript, the chaincode exposes the following functionalities: RecordSensorData(ctx, sensorId, temperature, humidity, location, timestamp), which validates the presence of all required fields, converts temperature and humidity into floating-point numbers using parseFloat, and then stores the JSON object (containing docType, sensorId, temperature, humidity, location, and timestamp) in the World State through ctx.stub.putState(). The ReadSensorData(ctx, sensorId) function queries the current state of a sensor using ctx.stub.getState() and returns the corresponding JSON data (or an error if the sensor does not exist). The GetSensorHistory(ctx, sensorId) function enables the traversal of the complete history of all modifications related to a sensor using ctx.stub.getHistoryForKey(), returning for each historical state the transaction identifier (txId), validation timestamp (timestamp), a deletion or modification indicator (isDelete), as well as the complete JSON record data.

Consensus uses the Raft algorithm [33] (Byzantine fault tolerance for orderers), which offers fast finality and predictable latency compared to earlier versions of Kafka [17].

5. Technical implementation

5.1. Technology stack and test environment

Table 2 lists all the software and hardware components used for deployment and evaluation.

The local deployment uses Docker Compose with a test profile: 2 peers (1 per organization), 2 Raft orderers, 1 CA, 1 CouchDB (World State), and 1 Node-RED instance. The runtime environment was a laptop equipped with an Intel Core i7-3610QMx8 processor, 8 GB of DDR3 RAM, and an NVMe SSD, running Ubuntu 24.04.4 LTS.

5.2. Detailed Node-RED flow

The Node-RED flow shown in Figure 2 operates sequentially to simulate the sensor and transmit the data to the blockchain. The flow is represented by five detailed sequential stages:

A[Step 1: Inject / Trigger] -->|Every 10s| B[Step 2: Payload Formatting Function]
B -->|JSON Data| C[Step 3: Debug Log Payload]
B -->|JSON Data| D[Step 4: HTTP POST Request]

Table 2. Technology stack and test environment

Component	Version	Role
Hyperledger Fabric	2.5.4	Permissioned blockchain network
Fabric CA	1.5.7	Certificate authority and MSP management
Node-RED	3.1.3	Edge gateway and flow orchestration
Node.js	18 LTS	Edge gateway and flow orchestration
Express.js	4.18.2	REST API framework
fabric-network	2.2.18	Client SDK for transactional submission
Iot-fabric-benchmark	1.0.0	Load testing and benchmark harness

D -->|API Response| E[Step 5: Debug Blockchain Response].

The Node-RED flow starts with an “Inject” node that triggers the sending of a timestamp every 10 seconds. A “Function” node executes a JavaScript script to generate realistic data and formats it into JSON. This data is simultaneously routed to a “Debug” node for visualization in the console and to an “HTTP Request” node that sends it via a POST request to the local API gateway. The final “Debug” node displays the API gateway response, confirming the signing and recording of the transaction on the Hyperledger Fabric ledger.

5.3. API gateway middleware

The API gateway forms the core of transactional mediation between IoT devices and the Hyperledger Fabric network. Listing 1 presents an excerpt of the implementation of the REST endpoints (POST /api/sensors for writing and GET /api/sensors/:id for reading), which are responsible for validating required parameters (sensor ID, temperature, humidity, location, timestamp), managing the digital wallet and cryptographic identities through loading the connection profile (connection-org1.json), signing transactions by the authorized user (appUser), and atomically submitting transactions to the Smart Contract (RecordSensorData for immutable recording, ReadSensorData for querying the current or historical state).

```
// POST Endpoint: IoT Data Ingestion (Invoked by Node-RED)
app.post('/api/sensors', async (req, res) => {
    try {
 const { sensorId, temperature, humidity, location, timestamp }
= req.body;

        if (!sensorId || !temperature || !humidity) {
```

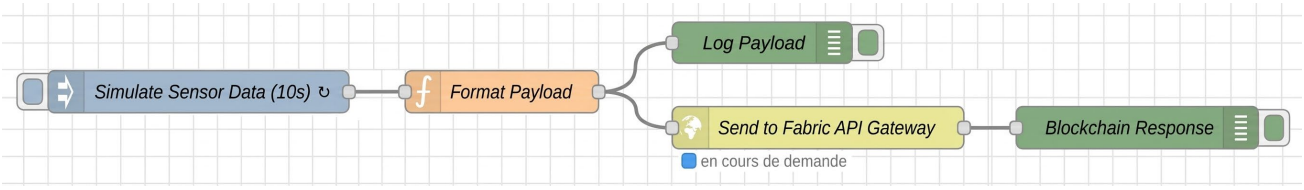

Figure 2. Node-RED flow

```
        return res.status(400).json({ error: 'Missing required
parameters' });
    }

    const contract = await getContract();

// Submitting the Transaction to the Fabric Network
    const result = await contract.submitTransaction(
      'RecordSensorData',
      sensorId,
      temperature.toString(),
      humidity.toString(),
      location || 'Unknown',
      timestamp || new Date().toISOString()
    );

    res.status(200).json({
      message: 'Transaction has been submitted',
      payload: JSON.parse(result.toString())
    });
  } catch (error) {
    console.error(`Failed to submit transaction: ${error}`);
    res.status(500).json({ error: error.message });
  }
});

// GET Endpoint: Retrieve Data from a Specific Sensor
app.get('/api/sensors/:id', async (req, res) => {
  try {
    const contract = await getContract();
    const result = await
contract.evaluateTransaction('ReadSensorData', req.params.id);
    res.status(200).json(JSON.parse(result.toString()));
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
app.listen(PORT, () => {
  console.log(`API Gateway is running on
http://localhost:${PORT}`);
});
```

Listing 1. Excerpt of the implementation of the REST endpoints of the API Gateway.

5.4. Chaincode (smart contract)

The supplychain.js chaincode was developed in JavaScript for the Hyperledger Fabric Node.js runtime. It encapsulates the business logic for validating, persisting, and ensuring the traceability of temperature and humidity sensor data. Listing 2 presents an excerpt of supplychain.js; the RecordSensorData function allows recording or updating the value associated with a key (in this case, the sensorId). The ReadSensorData function retrieves the most recent value associated with the key.

```
// Excerpt from supplychain.js
// A. Writing to the Ledger
async RecordSensorData(ctx, sensorId, temperature, humidity,
location, timestamp) {
 if (!sensorId || !temperature || !humidity || !location ||
!timestamp) {
      throw new Error('All fields are required');
    }
    const sensorData = {
      docType: 'sensorData',
      sensorId: sensorId,
      temperature: parseFloat(temperature),
      humidity: parseFloat(humidity),
      location: location,
      timestamp: timestamp
    };

 // Ledger State Recording (Key–Value Pair Stored as a JSON
Buffer)
    await ctx.stub.putState(sensorId,
Buffer.from(JSON.stringify(sensorData)));
    return JSON.stringify(sensorData);
  }
// B. Reading the Current State
async ReadSensorData(ctx, sensorId) {
    const sensorJSON = await ctx.stub.getState(sensorId);
    if (!sensorJSON || sensorJSON.length === 0) {
 throw new Error(`The sensor data for ${sensorId} does not
exist`);
    }
    return sensorJSON.toString();
  }
```

Listing 2. Excerpt from supplychain.js

6. Performance evaluation and results

6.1. Experimental protocol

A load testing benchmark was developed in JavaScript to simulate bursts of asynchronous requests to the API gateway. The experimental protocol followed the following reproducibility criteria:

Independent variables: number of concurrent sensors (10, 100, 500, 1000). Dependent variables: throughput

(TPS), average latency (ms), and success rate. Duration: 60 seconds per burst, preceded by a 10-second warm-up phase. Payload: 240-byte JSON (sensorID, nodeID, 5 aggregated readings, signature). Network configuration:

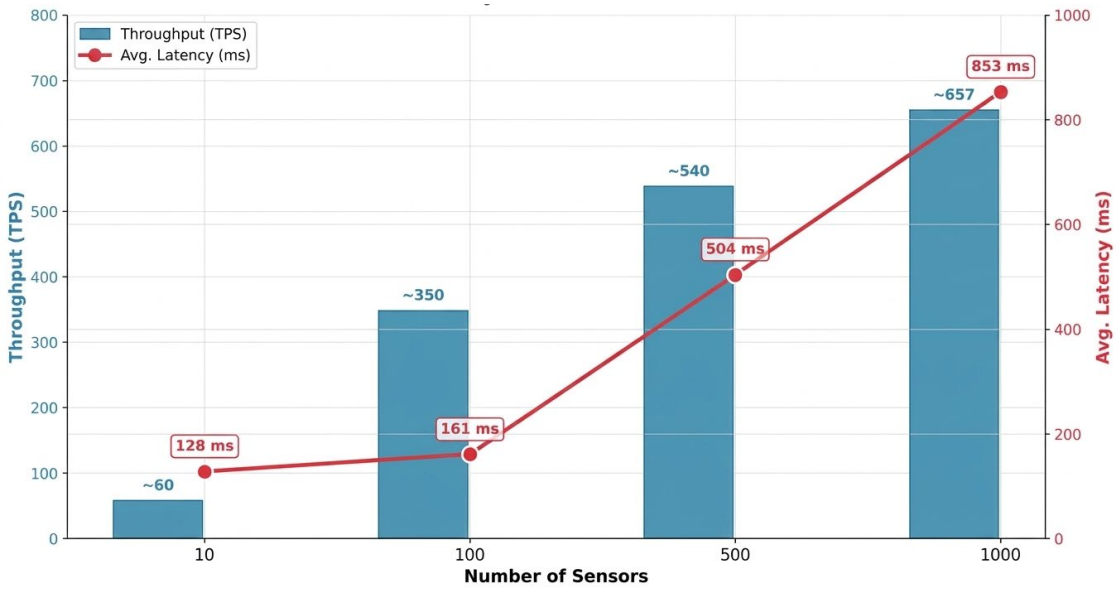

Figure 3. Performance evaluation: Throughput vs. Latency

localhost (negligible network latency), thereby isolating the performance metrics of the blockchain system itself.

6.2. Quantitative results

Table 3 summarizes the performance metrics obtained for each load level in this study. Each measurement corresponded to the average of five independent runs.

Table 3. Performance results under increasing loads (average of 5 executions)

Number of Sensors	Throughput (TPS)	Avg. Latency (ms)	Success Rate
10	~60	128	100 %
100	~350	161	100 %
500	~540	504	100 %
1 000	~657	853	100 %

The combined graph illustrated in Figure 3 presents the system performance data. It is clearly observed that the system handles the load increase well up to 100 sensors (moderate latency at 161 ms). However, beyond 500 sensors, the latency increases significantly (504 ms then 853 ms), indicating a bottleneck.

6.3. Analysis of results

Horizontal scalability - The system demonstrates near-linear scalability up to 500 sensors, where the throughput increases from 60 to 540 TPS. Beyond this point, the performance curve inflects (657 TPS at 1,000 sensors), revealing a saturation phenomenon in the Raft orderer and the underlying gRPC connection layer. This observation aligns with the findings of [17], where the authors note that Hyperledger Fabric reaches a performance plateau due to the sequential endorsement mechanism and the size of the SDK connection pool.

Latency - The average latency remained below 200 ms for moderate loads (< 100 sensors), which is consistent with the real-time requirements of the cold chain (immediate alerts in case of temperature excursion).

Edge lightweightness - The lightweight nature is demonstrated by the fact that IoT sensors experience no cryptographic overhead. No cryptographic processes were executed on the simulated sensors, thereby validating hypothesis H_1 of the full delegation of the blockchain workload to the upper layers.

7. Experimental evaluation

7.1. Comparative evaluation of the architectures

Objective and description of the experimental protocol

This experiment aimed to evaluate the performance of the proposed architecture in comparison with two conventional approaches to interacting with Hyperledger Fabric. Three configurations were investigated. The first (A1) relied on

direct transaction submission by a Node.js client using the complete fabric-network SDK. The second (A2) used the modern @hyperledger/fabric-gateway library, which delegates certain operations to the Gateway service integrated into the Fabric peer. Finally, the third configuration (A3) corresponded to the proposed architecture, in which IoT devices communicated with an Express.js REST API responsible for managing interactions with the blockchain. To ensure a fair comparison, all three configurations were evaluated on the same Hyperledger Fabric 2.x network, consisting of two organizations, a common channel, and a Raft ordering service configured with a 2-second batching timeout and a maximum of 10 transactions per block. The same RecordSensorData chaincode function and payloads of approximately 240 bytes were used. For each architecture, three independent runs of 50 transactions were performed, with 10 concurrent requests and a warm-up phase consisting of 10 transactions. The evaluated metrics included throughput, average latency, P50, P95, and P99 percentiles, transaction validation rate, and client-side CPU and memory consumption.

Experimental results

Table 4 presents a comparative evaluation of the three transaction submission architectures (A1: Direct Client Submission, A2: Fabric Gateway Client, and A3: Proposed Delegated Middleware Architecture). It considers their throughput, transaction latency, resource consumption, and success rate under identical experimental conditions. This comparison aimed to assess the impact of the proposed delegated middleware on the blockchain interaction performance.

The results reveal a significant performance difference among the three architectures. A1 achieves an average throughput of 1.87 transactions per second (TPS), with an average latency of 5,337.88 ms. A2 slightly improves throughput, reaching 2.01 TPS, while reducing average latency to 4,599.82 ms. In contrast, A3 achieves the best performance, with a throughput of 7.07 TPS and an average latency of 1,412.16 ms. Compared with A1, this architecture increases throughput by 278.1% and reduces average latency by 73.5%. Compared with A2, the improvements reach 251.7% and 69.3%, respectively. The 95th-percentile latency also decreases to 2,119.60 ms with A3, compared with 8,097.80 ms for A1 and 10,080.38 ms for A2. Furthermore, all three configurations achieve a 100% transaction validation rate, with no transaction loss. Client-side memory consumption is 113.95 MB for A1, 83.33 MB for A2, and 55.63 MB for A3. CPU utilization follows the same trend, reaching 4.03%, 3.83%, and 2.73%, respectively.

Discussion

These results highlight the benefits of delegating blockchain interactions to a middleware layer. In A1, the client directly handles several operations related to peer discovery, endorsement, and transaction submission. This increases the communication complexity and resource requirements. A2 reduces part of this workload through the Fabric Gateway service; however, the client remains responsible for managing its cryptographic identity and

Table 4. Comparative performance evaluation of the three transaction submission architectures

Performance Metric	A1 — Direct Client	A2 — Fabric Gateway	A3 — Proposed Architecture	Improvement (A3 vs. A1)	Improvement (A3 vs. A2)
Overall Throughput (TPS)	1.87 TPS	2.01 TPS	7.07 TPS	+278.1% (×3.78)	+251.7% (×3.52)
Average Validation Latency	5,337.88 ms	4,599.82 ms	1,412.16 ms	−73.5% (÷3.78)	−69.3% (÷3.26)
Median Latency (P50)	8,038.08 ms	4,796.07 ms	2,021.44 ms	−74.8%	−57.8%
95th-Percentile Latency (P95)	8,097.80 ms	10,080.38 ms	2,119.60 ms	−73.8%	−79.0% (÷4.76)
99th-Percentile Latency (P99)	8,115.83 ms	10,096.77 ms	2,125.20 ms	−73.8%	−79.0% (÷4.75)
Total Duration (50-transaction burst)	26.70 s	25.22 s	7.07 s	3.78× faster	3.57× faster
Success Rate and Ledger Validation	100.0% (150/150)	100.0% (150/150)	100.0% (150/150)	Equal (100% integrity)	Equal (100% integrity)
Client RAM Footprint (Isolated)	113.95 MB (full SDK)	83.33 MB (Gateway v1)	55.63 MB (native HTTP)	−51.2%	−33.2%
Client CPU Utilization	4.03%	3.83%	2.73%	−32.3%	−28.7%

private keys. A3 further simplifies processing on the IoT device side. The client sends its data through HTTP, while the middleware centralizes the operations required to record them on Fabric. Session reuse and asynchronous request processing may also contribute to the observed improvements in throughput and latency. From a hardware perspective, the reduction in memory and CPU consumption confirms the benefits of this separation of responsibilities. However, the measured client memory footprint of 55.63 MB corresponds to the client used in the experiment and does not directly represent memory consumption on a microcontroller, such as an ESP32. Similarly, private-key protection depends on the effective security of the middleware. Finally, the possibility of horizontally replicating the API does not, by itself, demonstrate the elimination of all single points of failure. These aspects should therefore be considered architectural properties that require further evaluation under broader deployment conditions.

7.2. Evaluation of the impact of batching

Objective and description of the experimental protocol

The objective of this experiment was to analyze the influence of batching on the performance of the proposed architecture. More specifically, it aimed to determine the extent to which grouping multiple IoT readings into a single Fabric transaction improves throughput and reduces the validation cost associated with each data record. Four configurations were investigated: individual processing without batching (Batch = 1), followed by three configurations grouping 10, 50, and 100 readings per transaction, respectively. In the individual-processing mode, each measurement was submitted separately to the blockchain network. In the other configurations, several measurements were grouped into a single JSON payload and recorded through an atomic transaction. The tests were conducted using the proposed architecture, following a warm-up phase and three independent repetitions for each configuration. Submission and validation times were collected separately through instrumentation of the API gateway. Additional metrics were also recorded, including IoT data throughput, blockchain transaction throughput, processing latency, success rate, and middleware resource consumption.

Experimental results

Table 5 summarizes the results obtained with different batching configurations, ranging from 1 to 100 IoT readings per blockchain transaction. The evaluation focuses on data throughput, transaction throughput, latency, success rate, and resource consumption to examine how batching affects the system performance and transaction processing efficiency.

The results indicate a substantial improvement in effective throughput as the batch size increases. The individual-processing mode achieves 7.06 readings per second, compared with 14.20 readings per second for Batch = 10, 74.78 readings per second for Batch = 50, and 133.09 readings per second for Batch = 100. Thus, grouping 100 measurements achieves an effective throughput approximately 18.85 times higher than that of individual processing. At the same time, the number of blockchain transactions required decreases significantly: it falls from 7.06 transactions per second in individual mode to approximately 1.33 transactions per second with Batch = 100. The average end-to-end latency of a batch reaches approximately 3,350.93 ms in the latter configuration. However, the amortized latency per reading decreases from 1,415.00 ms to 33.51 ms, representing a reduction of 97.6%. Submission latency gradually increases with batch size, rising from 92.90 ms for Batch = 1 to 225.85 ms for Batch = 100. Validation latency remains relatively stable at approximately 2.3 seconds. Finally, all configurations achieve a 100% success rate. Middleware memory consumption remains between 78.3 and 82 MB, while CPU utilization remains low, ranging from 0.9% to 2.6%.

Discussion

These results demonstrate that batching reduces the number of interactions required with the blockchain and distributes their cost across multiple readings. In the individual-processing mode, each measurement requires a separate transaction, multiplying the endorsement, submission, and validation operations. Conversely, grouping several measurements into a single transaction allows these operations to be shared. This approach explains the increase in effective throughput, particularly for batch sizes of 50 and 100 readings. The results also show that the main improvement concerns the time cost per measurement rather than the overall validation time of a batch. Indeed, the overall latency increases as more readings are grouped together because several data records must be prepared and processed within a single transaction. The stability of validation latency at approximately 2.3 seconds suggests that this phase remains mainly influenced by Fabric's ordering and validation mechanisms. Therefore, selecting an appropriate batch size requires balancing throughput and responsiveness. For cold-chain monitoring applications, Batch = 50 or Batch = 100 may be suitable when data volumes are high and a batch validation delay of a few seconds is acceptable. Nevertheless, it is important to distinguish IoT reading throughput from blockchain transaction throughput: an increase in the former does not necessarily imply an increase in the number of transactions processed by the network.

7.3. Performance comparison of LevelDB and CouchDB state databases

Objective and description of the experimental protocol

This experiment aimed to measure the influence of the state database on the performance of the Hyperledger Fabric network. Two solutions were compared: LevelDB, which

Table 5. Performance evaluation under different batching configurations

Performance Metric	Batch = 1 (Disabled — Baseline)	Batch = 10 (Enabled — Small)	Batch = 50 (Enabled — Medium)	Batch = 100 (Enabled — Large)	Relative Change (B=100 vs. B=1)
Effective IoT Data Throughput (Readings/s)	7.06 TPS	14.20 TPS	74.78 TPS	133.09 TPS	+1,786.0% (×18.85)
Blockchain Transaction Throughput (Transactions/s)	7.06 tx/s	1.42 tx/s	1.50 tx/s	1.33 tx/s	100-fold batching
Amortized Latency per Reading	1,415.00 ms	351.75 ms	67.13 ms	33.51 ms	−97.6% (÷42.2)
Average End-to-End Latency per Batch	1,415.00 ms	3,517.44 ms	3,356.14 ms	3,350.93 ms	Moderate increase (+136.8%)
Submission Latency (Tsub)	92.90 ms	70.54 ms	163.77 ms	225.85 ms	Increase due to payload size
Validation / Raft Latency (Tval)	329.45 ms	2,324.12 ms	2,298.31 ms	2,298.24 ms	Nearly constant (~2.3 s)
Median Batch Latency (P50 E2E)	2,025.41 ms	4,018.04 ms	3,698.58 ms	3,681.50 ms	High reproducibility
95th-Percentile Batch Latency (P95 E2E)	2,100.83 ms	4,028.41 ms	3,707.41 ms	3,708.38 ms	No significant jitter
Success Rate and Ledger Integrity	100.0% (150/150)	100.0% (300/300)	100.0% (1,500/1,500)	100.0% (3,000/3,000)	100% integrity
Gateway Memory Footprint (RAM RSS)	78.3 MB	79.9 MB	80.8 MB	82.0 MB	Stable (<85 MB)
Gateway CPU Utilization	2.6%	0.9%	0.9%	1.1%	Low resource consumption

is integrated directly into the peer process, and CouchDB, an external document-oriented database. The evaluation was conducted using Hyperledger Caliper v0.6.0 on the mychannel channel, with the supplychain-cc smart contract. Several workload levels were applied to investigate the behavior of both databases under different operating conditions. The tests included write operations at 10, 25, and 50 TPS, a read test at 15 TPS, a mixed workload consisting of 70% write operations and 30% read operations at 20 TPS, and a saturation test at 100 TPS. The main evaluation metrics included write throughput, read and write latency, transaction success rate, and CPU and memory consumption. Resource usage was monitored using Docker container statistics, taking into account that CouchDB requires an additional container compared with LevelDB.

Experimental results

Table 6 reports the performance of LevelDB and CouchDB under different workloads. The comparison covers write throughput, read latency, CPU utilization, memory consumption, and transaction success rate. These metrics were used to evaluate the suitability of each state database for the proposed IoT-based supply chain architecture.

The results show that LevelDB generally provides a better write performance. At a target workload of 10 TPS, its throughput reaches 9.6 TPS, compared with 8.9 TPS for the CouchDB. At 25 TPS, the measured throughput values were 24.1 and 22.2 TPS, respectively. The difference becomes more significant at 50 TPS: LevelDB achieves 41 TPS, whereas CouchDB is limited to 24.1 TPS. During the 100 TPS saturation test, the maximum observed throughput was 49.9 TPS for LevelDB and 25 TPS for CouchDB. In contrast, both databases exhibit similar read latency for direct key-based operations, estimated at approximately 20 ms. Regarding resource consumption, LevelDB uses approximately 220–230 MB of memory per peer, whereas

Table 6. Comparative performance evaluation of LevelDB and CouchDB

Criterion / Metric	Test Round	LevelDB	CouchDB	Performance Gain / Difference	Summary and Analysis
Write Throughput (@ 10 TPS)	R1 (10 TPS)	9.6 TPS (latency: 0.77 s)	8.9 TPS (latency: 0.93 s)	+7.9% throughput for LevelDB	LevelDB is more responsive and faster
Write Throughput (@ 25 TPS)	R2 (25 TPS)	24.1 TPS (latency: 0.52 s)	22.2 TPS (latency: 1.09 s)	+8.6% throughput; 52% lower latency for LevelDB	CouchDB latency is approximately twice as high
Write Throughput (@ 50 TPS)	R3 (50 TPS)	41.0 TPS (latency: 6.85 s)	24.1 TPS (latency: 10.73 s)	+70.1% throughput for LevelDB	CouchDB reaches saturation at 24.1 TPS
Read Latency	R4 (15 TPS)	0.02 s (20 ms)	0.02 s (20 ms)	Equal performance	Identical direct key-value lookup latency
Mixed Workload (70% Write / 30% Read)	R5 (20 TPS)	19.5 TPS (latency: 0.49 s)	18.3 TPS (latency: 0.62 s)	+6.6% throughput; 21% lower latency for LevelDB	LevelDB performs better under mixed workloads
Stress Test (Saturation @ 100 TPS)	R6 (100 TPS)	49.9 TPS (maximum throughput)	25.0 TPS (maximum throughput)	+99.6% maximum throughput for LevelDB	LevelDB nearly doubles the processing capacity
Transaction Success Rate (Nominal Workloads)	R1–R3, R5	100% (6,908/6,908)	100% (2,208/2,208)	100% for both databases	Full transaction integrity for both engines
Memory Consumption (Peers)	Overall	~220–230 MB / peer	~132–144 MB + 77 MB CouchDB	LevelDB has a lower overall footprint	CouchDB requires two additional dedicated containers
CPU Utilization Under Load	Overall	~7.79% (peer0.org1)	~12.41% (peer0) + 7% (CouchDB)	~140% higher total CPU usage for CouchDB	Additional overhead from local TCP communication and JSON parsing

the CouchDB configuration requires between 132 and 144 MB for the peer, in addition to approximately 77 MB for the CouchDB container. CPU utilization is also lower with LevelDB, at approximately 7.79% for the peer, compared with 12.41% for the peer and an additional 7% for CouchDB. Finally, both solutions achieved a 100% transaction success rate under the reported nominal test conditions.

Discussion

The results indicate that LevelDB is better suited to workloads dominated by write operations and direct key-based reads. Its integration into the peer process avoids additional communication with an external service, which may explain the observed improvements in throughput and CPU consumption. CouchDB introduces additional overhead because it operates as a separate service and processes JSON documents. This difference becomes particularly noticeable as the workload increases, especially during the 50 and 100 TPS tests. However, both databases provide comparable performance for simple read operations based on sensor identifiers. Therefore, the choice between the two solutions depends on the application's requirements. For a cold-chain monitoring application that mainly records readings associated with unique sensor identifiers, LevelDB is an attractive option because of its write throughput and resource efficiency. CouchDB remains relevant when the application requires rich queries and document indexing operations. Nevertheless, the read results presented here concern direct key-based access and do not establish equivalence between the two databases for complex queries. Such queries require a separate evaluation because the capabilities of the two databases are not identical.

7.4. Evaluation of resilience and fault tolerance

Objective and description of the experimental protocol

This experiment aimed to evaluate the ability of the proposed architecture to maintain continuous data acquisition and recording in the presence of failures. More specifically, it examined the system behavior when connectivity between the edge devices and the gateway was interrupted, when the Express.js API was stopped, when a Fabric peer failed, and when the CouchDB state database became unavailable. For each scenario, failures were injected during workload execution, and system behavior was monitored until the service was restored. A comparison was also conducted between the two edge configurations: one without a buffering mechanism and another using a Store-and-Forward temporary storage mechanism in Node-RED. Each scenario was repeated three times using a workload of 20 transactions. The measured indicators included downtime, recovery time, data loss percentage, success rate, and maximum number of pending transactions. The objective was to determine whether data generated during an interruption could be retained and subsequently transmitted to the blockchain once connectivity was restored.

Experimental results

Table 7 presents the results of the resilience experiments conducted under different failure scenarios. These scenarios include edge connectivity interruption, API gateway failure, Fabric peer failure, and CouchDB failure. The evaluation considers downtime, recovery time, data loss, transaction success rate, and pending transactions to assess the system's ability to recover from component failures.

The results reveal different behaviors depending on the affected component. During an edge connectivity interruption without buffering (S1-A), the system experienced 6 seconds of downtime and a data loss rate of 50%, corresponding to 10 out of 20 transactions. With the buffering mechanism enabled (S1-B), data loss decreased to 0%, with a recovery time of 24.71 seconds and a maximum of 10 pending transactions. When the Express.js API gateway was restarted (S2), the downtime was 9.42 seconds and the recovery time was 44.34 seconds. The system retained up to 16 pending transactions, with no data loss. In the Fabric peer failure scenario (S3), the recovery time was 15.31 seconds, with a maximum of two pending transactions and a 100% success rate. Finally, stopping CouchDB (S4) results in an average recovery time of 46.82 seconds, with a standard deviation of ±20.2 seconds. Eight transactions are observed as pending, and no data loss is reported. The recovery scenarios therefore achieved a 100% success rate, unlike the Edge scenario without buffering, which reached only 50%.

Discussion

These results highlight the importance of recovery mechanisms and temporary storage in a blockchain–IoT architecture. The comparison between S1-A and S1-B demonstrates that Edge-level buffering plays an essential role when connectivity is intermittent. Without this mechanism, measurements generated during an interruption may be lost. With Store-and-Forward enabled, data are stored locally and retransmitted once the connection is restored. The Express.js API gateway restart scenario also demonstrates the benefits of queuing and retry mechanisms on the Edge side. Similarly, the failure of a Fabric peer is compensated for, under the tested configuration, by the availability of another peer capable of participating in transaction processing. The CouchDB scenario shows that state database unavailability may temporarily interrupt operations, but service can resume after the affected components are restarted. Overall, the results confirm the value of combining Edge-level buffering, supervised middleware, and blockchain-level

Table 7. Resilience evaluation under different failure scenarios

Experimental Scenario	Downtime (s)	Recovery Time (s)	Data Loss (%)	Success Rate (%)	Peak Pending Transactions
S1-A: Basic Edge Node (Without Buffering)	6 ± 0 s	0 ± 0 s	50% (10 tx)	50 ± 0%	0 tx
S1-B: Proposed Edge Node (With Buffering)	6 ± 0 s	24.71 ± 0.19 s	0% (0 tx)	100 ± 0%	10 tx
S2: Express.js API Gateway Crash	9.42 ± 0.06 s	44.34 ± 0.25 s	0% (0 tx)	100 ± 0%	16 tx
S3: Fabric Peer Crash (Failover)	6 ± 0 s	15.31 ± 0.04 s	0% (0 tx)	100 ± 0%	2 tx
S4: CouchDB State Database Crash	6 ± 0 s	46.82 ± 20.2 s	0% (0 tx)	100 ± 0%	8 tx

recovery mechanisms. However, these observations must be interpreted in the context of the topology actually tested. The failure tolerance of a peer does not automatically demonstrate the fault tolerance of the Raft ordering service. Likewise, the results obtained with a single unavailable peer are insufficient to establish network resilience against a general partition or the loss of the ordering service quorum. Additional tests involving multiple orderers and different network configurations would provide a more comprehensive assessment of these properties.

8. Discussion and comparative analysis

8.1. Comparison with the state of the Art

Table 8 positions our work in relation to several recent blockchain-IoT solutions published in the literature. The performance metrics were reported as published by the authors, although the experimental conditions differed (hardware, network topology, block size).

The chart shown in Figure 4 provides a performance comparison with the studies referenced in Table 8. It is evident that Study [21] outperforms the others in terms of throughput (49,039 TPS) while exhibiting the lowest latency (~433 ms). In contrast, our study achieved a throughput of 657 TPS with a latency of approximately 853 ms.

Our architecture, although less performant in absolute terms than the BFT-DAG solution [21], has the decisive advantage of relying on a mature industrial ecosystem (Hyperledger Fabric, Node-RED, Docker) without requiring the deployment of a proprietary consensus protocol. The throughput of 657 TPS far exceeds the requirements of pharmaceutical supply chains (typically < 100 TPS per regional warehouse), while still providing immutable and interoperable traceability. Compared to [17], our Node-RED delegated-node approach offers greater protocol flexibility (MQTT, HTTP, Modbus) and a reduced memory footprint on edge gateways.

8.2. Advantages of the delegated node approach

Protocol interoperability: Node-RED natively supports MQTT, Modbus, LoRaWAN, HTTP, and OPC UA, enabling the integration of heterogeneous sensors without modifying the chaincode [18]. This property is essential in the supply chain, where sensor fleets come from multiple manufacturers with proprietary protocols.

Network resilience: Local SQLite buffering ensures service continuity in areas with reduced coverage (tunnels, isolated warehouses, maritime zones), a major challenge for the international cold chain [30]. Data is stored locally and synchronized as soon as connectivity is restored, guaranteeing continuous temporal traceability.

Separation of responsibilities: Identity management (Wallet) is centralized at the middleware level, simplifying X.509 certificate rotation and revocation in the event of a gateway compromise. This architecture respects the principle of least privilege by limiting private key exposure to a single secure component.

9. Architecture limitations and threats

9.1. Single point of failure (SPoF)

The API gateway and the Node-RED instance constitute a centralized bottleneck. In the event of hardware failure, distributed denial-of-service (DDoS) attack [34], or filesystem corruption, the data flow between the edge and the blockchain is interrupted. This limitation is inherent to gateway-centric architectures and has been identified as a major risk in zero-trust IoT frameworks [20]. The lack of active redundancy forms the main operational threat for a production deployment.

9.2. GIGO problem

Although the blockchain guarantees post-submission immutability, it cannot validate the authenticity of sensor data at the source. A faulty, miscalibrated, or physically compromised sensor will transmit erroneous data that will be irrevocably recorded on the ledger. This issue is central to the work of [17], who propose an incentive-based trust

Table 8. Performance comparison with recent blockchain-iot works

Work	Blockchain Technology	Throughput (TPS)	Latency	Edge Gateway	Domain
Study [17]	Hyperledger Fabric + multi-channel	~400	~600 ms	Not specified	Agri-food
Study [18]	Hyperledger Fabric + sidechain	~250	~900 ms	SDK Gateway (OPC UA)	Oil & gas
Study [21]	Custom BFT-DAG	49 039	433 ms	Dedicated edge nodes	Generic IIoT
Our work	Hyperledger Fabric + Node-RED	657	853 ms	Node-RED (delegated)	Cold chain

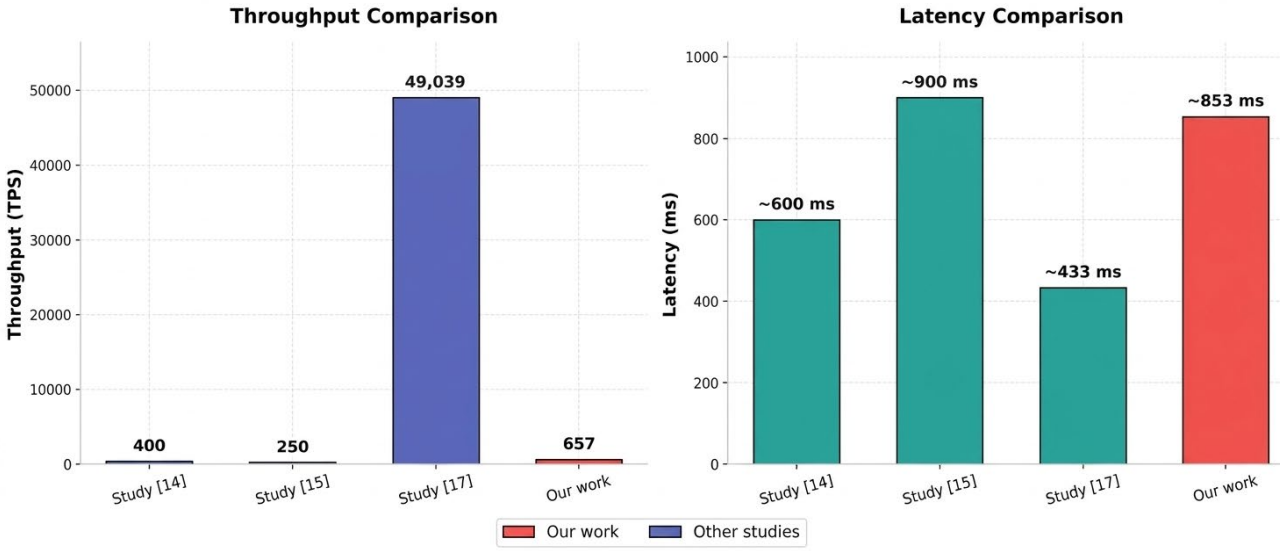

Figure 4. Performance comparison across studies

score mechanism to encourage actors to submit reliable data. Our architecture, although it implements basic plausibility filtering at the edge, does not yet have an advanced anomaly detection mechanism (e.g., isolation forest, LSTM (Long Short-Term Memory) auto-encoder).

9.3. Ledger saturation

At 657 TPS, the Fabric ledger grows by approximately 56 MB per hour (2 MB blocks, 2-second batch timeout). Over one year, this represents approximately 490 GB of raw data, requiring a pruning or off-chain archiving strategy (S3/IPFS object storage with on-chain hashing) for system sustainability [5]. Without a retention policy, the synchronization time for new peers would become prohibitive (> 24 hours), compromising operational scalability.

10. Future work and perspectives

10.1. Decentralization of gateways

To address the SPoF, a distributed gateway architecture (containerized microservices under Kubernetes) with load balancing and active redundancy will be deployed [35]. Each Node-RED instance will have a distinct Wallet under the same MSP (Org2-LogisticsMSP), enabling horizontal transactional sharding. A Kubernetes orchestrator will ensure automatic failover and auto-scaling based on incoming MQTT load. This approach builds upon work on blockchain microservice architectures [29].

10.2. Artificial intelligence at the edge

The integration of a lightweight Machine Learning module (TensorFlow Lite) within Node-RED will enable real-time anomaly detection (e.g., abnormal thermal drift, faulty sensor) before blockchain submission [36]. This AI-at-the-Edge approach, advocated in [20], will reduce the volume of erroneous data recorded on-chain and trigger predictive alerts. An LSTM model trained on historical time series will identify sensor failure patterns with accuracy exceeding 95%, in line with recent benchmarks in IoT anomaly detection.

10.3. Deployment on physical hardware

The next experimental phase will replace simulated sensors with physical devices (ESP32 + DHT22) communicating via MQTT over Wi-Fi/LoRaWAN with Node-RED. Real-world tests (refrigerated container, pharmaceutical warehouse) will validate the resilience of SQLite buffering and compliance with EN 12830 standards [30]. The integration of GPS modules (NEO-6M) will enable automatic enrichment of readings with geographic coordinates, reinforcing the spatial traceability of the cold chain [37].

10.4. Multi-chain interoperability

Exploration of Hyperledger Fabric's multi-channel mechanism [16] will enable the isolation of sensitive commercial data (prices, contracts, supplier identities) from environmental readings, offering granular access governance across organizations. Each supply chain actor (producer, carrier, distributor, regulator) will access only the channels relevant to their role, in accordance with the need-to-know principle [36].

11. Conclusion

This paper presents the design, implementation, and evaluation of an IoT-blockchain architecture dedicated to cold chain traceability. By offloading cryptographic complexity to Node-RED edge gateways, the system reconciles the lightweight constraints of sensors with the immutability requirements of the Hyperledger Fabric distributed ledger. The experimental results validate the initial hypothesis: a throughput of 657 TPS and controlled latency (< 1 s) under a load of 1,000 sensors, with a 100% success rate.

This architecture forms a solid proof of concept for future industrialization, subject to addressing the identified limitations (SPoF, GIGO, ledger growth) through the avenues of decentralization, embedded artificial intelligence, and multi-channeling outlined in future work. The combination of edge computing, permissioned blockchain, and flow-based visual programming offers a promising path for the responsible digitalization of supply chains sensitive to environmental conditions.

Acknowledgements

We would like to express our sincere gratitude to Mr. Jakimi, Head of the Department of Computer Science at the Faculty of Sciences and Techniques of Errachidia, for his constant support and for providing the logistical and educational resources necessary to carry out this study. His assistance contributed significantly to the smooth progress and successful completion of this work.

References

[1] Li S, Li H, Kong T. Digital technologies for sustainable food supply chains: a scoping review of impacts on food safety, loss reduction, and implications for nutritional security. Front Nutr. 2026;13:1796752. doi:10.3389/fnut.2026.1796752.

[2] Habazin J, et al. A review of artificial intelligence applications in cold chain and reverse logistics. Promet Traffic Transp. 2025;37(6):1409-1440. doi:10.7307/ptt.v37i6.1332.

[3] Villegas-Ch W, Gutierrez R, Govea J, García-Ortiz J. Integrated AI, IoT, and blockchain for enhancing security and traceability in perishable logistics. Emerg Sci J. 2025;9(5):2471-2496. doi:10.28991/ESJ-2025-09-05-011.

[4] Jusak J, Kerrison S. Integration of IoT and distributed ledger technologies: a survey, challenges, and future directions. ACM Comput Surv. 2026;58(9):233. doi:10.1145/3789255.

[5] Gholami M, Ghaffari A, Derakhshanfard N, İbrahimoğlu N, Pourhaji Kazem AA. Blockchain integration in IoT: applications, opportunities, and challenges. Comput Mater Contin. 2025;83(2):1561-1605. doi:10.32604/cmc.2025.063304.

[6] Abdullah, Hafeez N, Shabbir M, Ather MA, Rodríguez JLO, Sidorov G. Security, privacy, and scalability trade-offs in blockchain-enabled IoT systems: a systematic analytical review. Appl Sci. 2026;16:3638. doi:10.3390/app16083638.

[7] Zulkarnain MM, Ramli N, Nordin AN. Performance benchmarking of Hyperledger Fabric on heterogeneous hardware for IoT applications. IIUM Eng J. 2025;26(3):156-170. doi:10.31436/iiumej.v26i3.3610.

[8] Siddiqui S, Neha, HBK. ZK-rollup for Hyperledger Fabric: architecture and performance evaluation [Preprint]. arXiv:2602.08870. 2026 Feb. doi:10.48550/arXiv.2602.08870.

[9] Rhidoy HM, Miraz MH, Salam I. Optimising blockchain scalability for real-time IoT applications. Int J Adv Sci Eng Inf Technol. 2026;16(1):311-318. doi:10.18517/ijaseit.16.1.21523.

[10] Bentayeb Y, Chaoui K, Badir H. Integrating blockchain and edge computing: a systematic analysis of security, efficiency, and scalability. Int J Adv Comput Sci Appl. 2025;16(1). doi:10.14569/IJACSA.2025.0160160.

[11] Karaduman A, Gülhas G. Blockchain-enabled supply chain management: a review of security, traceability, and data integrity amid the evolving systemic demand. Appl Sci. 2025;15:5168. doi:10.3390/app15095168.

[12] Valapu ST, Saha A, Krishnamachari B, Menon V, Guin U. Reward-based blockchain infrastructure for 3D IC supply chain provenance. In: Proceedings of the 2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST); 2025; San Jose, CA, USA. p. 370-380. doi:10.1109/HOST64725.2025.11050057.

[13] Powell W, Foth M, Cao S, Natanelov V. Garbage in garbage out: the precarious link between IoT and blockchain in food supply chains. J Ind Inf Integr. 2022;25:100261. doi:10.1016/j.jii.2021.100261.

[14] Truong T, Phan MC, Nguyen HQ. Towards an IoT and Blockchain-based System for Monitoring and Tracking Agricultural Products. EAI Endorsed Trans IoT. 2023;9(2). doi:10.4108/eetiot.v9i2.1903.

[15] Sharma A, Babu LG, Buradkar MU, Shanmathi M, Vinisha J, Udhayamoorthi M. Integration of Blockchain and IoT for Enhanced Transparency in Diamond Supply Chain. EAI Endorsed Trans IoT. 2025;11. doi:10.4108/eetiot.7145.

[16] Liu Y, Feng Y, Sakurai K. Hyperledger Fabric-based multi-channel structure for data exchange in Internet of Vehicles. Electronics. 2025;14(3):572. doi:10.3390/electronics14030572.

[17] Zhou Z, Feng Y, Sakurai K. Design and implementation of a trusted food supply chain traceability system with incentive using Hyperledger Fabric. Computers. 2026;15:108. doi:10.3390/computers15020108.

[18] Nisar F, Mustafa E, Rehman F, Zaman SKU, Maray M, Malik A, et al. Hyperledger-enabled blockchain framework for secure data integration in midstream oil supply chains. PeerJ Comput Sci. 2026;12. doi:10.7717/peerj-cs.3491.

[19] Petrescu I, Niculae E, Vulturescu V, Dimitrescu A, Ungureanu LM. Transport and application layer protocols for IoT: comprehensive review. Technologies. 2025;13:583. doi:10.3390/technologies13120583.

[20] Khan IU, Khan FM, Haider ZA, Alturise F. Integrating AI, blockchain, and edge computing for zero-trust IoT security: a comprehensive review of advanced cybersecurity framework. Comput Mater Contin. 2025;85(3):4307-4344. doi:10.32604/cmc.2025.070189.

[21] Stodt F, Alshawki M, Reich C, Ligeti P, Theoleyre F. Securing the future: lightweight blockchain solutions for IIoT and IoT networks. Security and Privacy. 2025;8. doi:10.1002/spy2.70070.

[22] Abbasi M, Prieto J, Plaza-Hernandez M, Corchado JM. Proof-of-resource: A resource-efficient consensus

mechanism for IoT devices in blockchain networks. EAI Endorsed Trans IoT. 2024;10. doi:10.4108/eetiot.6565.

[23] Alramouni S, Alkassim N. Architecture design of scalable blockchain in IoT smart cities. In: Proceedings of the 16th International Multi-Conference on Complexity, Informatics and Cybernetics (IMCIC 2025); 2025. p. 48-53. doi:10.54808/IMCIC2025.01.48.

[24] IBM. IoT Blockchain Service: using REST and IBM Watson IoT Platform Service with IoT Blockchain Service [Internet]. IBM Documentation [cited 2026 Nov 5]. Available from: https://www.ibm.com/docs/SSCG66/iot-blockchain/getting_started/advanced_exercise.html

[25] Duman E, Aydoğan E. Enhancing traceability and reliability in cold chain logistics through Hyperledger Fabric and IoT. Appl Sci. 2025;15(22):12149. doi:10.3390/app152212149.

[26] Pajooh HH, Rashid M, Alam F, Demidenko S. Hyperledger Fabric blockchain for securing the edge Internet of Things. Sensors. 2021;21(2):359. doi:10.3390/s21020359.

[27] Reyna A, Martín C, Chen J, Soler E, Díaz M. On blockchain and its integration with IoT. Challenges and opportunities. Future Gener Comput Syst. 2018;88:173-190. doi:10.1016/j.future.2018.05.046.

[28] Kösemen C, Dalkılıç G, Öksüzer Ş. Performance analysis of lightweight Internet of Things devices on blockchain networks. Turk J Electr Eng Comput Sci. 2022;30(2):328-343. doi:10.3906/elk-2103-110.

[29] Chouhan C, Saxena M. A blockchain and IoT-enabled framework for automated and secure supply chain management: enhancing transparency, efficiency, and trust. In: Sridaran R, Priti S, editors. AI & ML - Frontiers in Cross Disciplinary Applications & Case Studies. Cham: Springer; 2026. (Communications in Computer and Information Science; vol. 2866). doi:10.1007/978-3-032-17300-3_11.

[30] Ela Innovation. Cold supply chain: 2026 complete guide [Internet]. 2026 [cited 2026 Nov 5]. Available from: https://elainnovation.com/en/cold-supply-chain-guide-2026/

[31] Sakti MDI, Dirgantara D, Ramadhan AK, Ibrahim MA. Optimizing asynchronous performance in Node.js with Express and PostgreSQL. Procedia Comput Sci. 2025;269:172-181. doi:10.1016/j.procs.2025.08.270.

[32] The Linux Foundation. Key concepts. In: Hyperledger Fabric Documentation. Release 2.5 [Internet]. [cited 2026 Nov 3]. Available from: https://hyperledger-fabric.readthedocs.io/en/release-2.5/key_concepts.html

[33] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: Proceedings of the USENIX Annual Technical Conference (USENIX ATC); 2014; Philadelphia, PA, USA. p. 305-319.

[34] Kumar R, Kumar P, Tripathi R, Gupta GP, Garg S, Hassan MM. A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled IoT network. J Parallel Distrib Comput. 2022;164:55-68. doi:10.1016/j.jpdc.2022.01.030.

[35] Marchese A, Tomarchio O. Enhancing the Kubernetes platform with a load-aware orchestration strategy. SN Comput Sci. 2025. doi:10.1007/s42979-025-03712-z.

[36] Sayar A, Kurtbaş MC, Sancaktar Ç, Kabak RD, Çakmak Ş. Application of Hyperledger blockchain technology to logistics supply chain with IoT. Procedia Comput Sci. 2025;252:814-823. doi:10.1016/j.procs.2025.01.042.

[37] Kamyod C, Arwatchananukul S, Aunsri N, Saengrayap R, Tontiwattanakul K, Prahsarn C, et al. IoT-based system for real-time monitoring and AI-driven energy consumption prediction in fresh fruit and vegetable transportation. Sensors. 2025;25:7475. doi:10.3390/s25247475.