CELTIC-NEXT
Eureka Cluster

Project Report

CISSAN

Collective intelligence supported by security aware nodes

D4.4 Updated Network Logging System

Editor: Klaus Chmelina, Geodata

Abstract

This document presents an updated version of CISSAN deliverable D4.2, describing a blockchain-based system for secure transmission and logging of IoT sensor data. Two distinct blockchain systems have been designed to meet different IoT security requirements:

Geodata's system combines the speed and scalability of the Lightning Network with the robustness of the Bitcoin blockchain to ensure consensus on the sequence and validity of IoT events, generating an indisputable and tamper-evident record. The document outlines the system's objectives and architecture, data management approach, an example workflow, and a prototype hardware and software implementation. It also describes a demonstrator built to showcase the system in operation and simulate attack scenarios. The second (confidential) part of the deliverable includes the software developed by Geodata.

Councilbox's Eventchain system uses a dual-blockchain architecture with Byzantine Fault Tolerant consensus based on cryptographically signed validation votes, integrated machine-learning-driven anomaly detection using deterministic models, and a key-value store with namespace isolation for device state management. The document details the Eventchain system's objectives and dual-chain architecture, data handling and consensus process with validation voting, integrated anomaly detection, example workflows, and software implementation. It also presents a blockchain explorer used to demonstrate the system's operation. The implementation features containerized deployment using Docker and comprehensive APIs.

Project CISSAN

Public Report

July 2025

Participants in project CISSAN are (in alphabetic order with the project coordinator first):

- University of Jyväskylä (coordinator)
- Affärsverken Karlskrona
- Arctos Labs Scandinavia AB
- Bittium Biosignals Ltd.
- Bittium Wireless Ltd.
- Blekinge Tekniska Högskolan
- Blue Science Park
- Clavister AB
- Councilbox Ltd.
- Geodata ZT GmbH
- Mattersoft Ltd.
- Mint Security Ltd.
- Netox
- Nodeon Finland Oy
- Savantic
- ScopeSensor Ltd.
- Techinova AB
- Wirepas Oy

CISSAN-Collective intelligence supported by security aware nodes

D4.4 Updated Network Logging System

Editor: Klaus Chmelina, Geodata

Project coordinator: Ilgin Safak, University of Jyväskylä

CELTIC published project result

Executive Summary

This report presents two complementary blockchain-based logging systems developed within the CISSAN project, each addressing different dimensions of IoT data security and integrity.

The Lightning Network Data Transmission System is a CISSAN solution that introduces a novel approach to securely transmitting and logging data in IoT environments. It addresses key challenges in current deployments such as data integrity, security, and efficient data handling through an architecture that combines Infineon security chips with the Lightning Network and blockchain technology.

The report outlines the system's objectives, architecture, data handling, user roles, an example workflow, and a prototype hardware and software implementation. It also describes a demonstrator designed to showcase the system in operation and simulate attack scenarios.

This implementation represents both a technological advancement and a strategic step toward improving operational security and data integrity in IoT applications across multiple sectors.

The Eventchain Network Logging System, developed by Councilbox, employs a dual-blockchain architecture with Byzantine Fault Tolerant consensus requiring a 51% quorum for block validation. Each hub sends cryptographically signed validation votes that are verified and counted by the master node's validation consumer service. The system integrates machine learning based anomaly detection directly into the consensus process, ensuring that all nodes produce identical results. Eventchain includes a Blockchain Key-Value Store (BKVS) with namespace isolation, while a Metadata Application Programming Interface (API) abstraction layer enables seamless integration for device lifecycle management.

Both systems periodically anchor their data to the Bitcoin blockchain to ensure long-term immutability. The Lightning Network solution does this through channel settlement mechanisms, whereas Eventchain anchors block header hashes directly. Together, these approaches demonstrate how blockchain technology can address diverse IoT security needs, from hardware-secured critical infrastructure to software-based enterprise deployments.

List of Authors

In alphabetic order by partner name:

- Rodrigo Martínez, Councilbox Ltd
- Klaus Chmelina, Alois Maierhofer, Geodata ZT GmbH
- Ilgin Safak, University of Jyväskylä

Table of Contents

List of Figures

Abbreviations

API	Application Programming Interface
BFT	Byzantine Fault Tolerant
BKVS	Blockchain Key-Value Store
CO	Construction Company
ECDSA	Elliptic Curve Digital Signature Algorithm
GB	Gigabyte
gRPC	Google Remote Procedure Call
HTLCs	Hash Time-Locked Contracts
IoT	Internet of Things
LN	Lightning network
LND	Lightning Network Daemon
LoRaWAN	Long Range Wide Area Network
ML	Machine Learning
PO	Project Owner
PBFT	Practical Byzantine Fault Tolerant
RAM	Random Access Memory
Regtest	Regression test
REST	Representational State Transfer
SHA	Secure Hash Algorithm
SSD	Solid State Drive
SU	Supervision
UID	Unique Identifier

1. Introduction

This document describes two blockchain-based logging systems, the **Lightning Network Data Transmission System** and **the Eventchain Blockchain Logging System.** Both systems enable the secure transmission, logging and anchoring of Internet of Things (IoT) events.

1.1 Objective of this document

The primary objective of this document is to provide a comprehensive overview of both systems. The document aims to:

- **Describe the Systems' Objectives**
- **Detail the Architectures**: Describe the core components and their roles in ensuring data integrity.
- **Explain the Functionalities**: Outline how data is transmitted and verified through the network, processed and, finally, anchored.
- **Highlight Security Measures**: Detail the distinct security functions and measures (e.g. anomaly detection methods, transmission protocols, security chips) safeguarding data from sensor to client.
- **Define Data Handling**: Explain the standardized format for event data and how it is managed, transformed, and validated through the network to ensure consistency and reliability.
- **Provide Workflow Examples:** Present examples of how events are transmitted, logged, and verified to aid understanding of the system's operation.
- **Present hard- and software prototypes and demonstrators:** Describe the implementation of prototypes and demonstrators used to showcase the system in operation.

1.2 Content and structure of this document

To achieve the above objective, the document is structured to guide readers through the conceptual framework, technical architecture, operational workflows, and the implemented hardware and software prototypes of the systems. The structure includes:

- **Introduction**
  - Objective of this document
  - Content and structure of this document
- **Lightning Network Data Transmission System**
  - Overview of the system's objective, architecture, data handling, prototype, and demonstrator
- **Eventchain Blockchain Logging System**
  - Overview of the system's objective, architecture, data handling, implementation details.
- **Conclusion**

Each section is designed to progressively build upon the reader's understanding, from high-level concepts down to specific technical implementations, ensuring clarity on how the systems operate to securely manage IoT data.

2. Lightning Network Data Transmission System

The Lightning Network data transmission system is designed to enhance the security and efficiency of logging sensor data within IoT ecosystems. Its primary purpose is to provide a **scalable, secure, and verifiable** record of events generated by IoT devices. By integrating the Lightning Network, the system ensures high-speed transactions and a **routing and transaction mechanism** that involves multiple independent nodes, increasing trust and data integrity.

The system addresses challenges in IoT data management, enabling organizations to leverage sensor data securely and efficiently, supporting innovation and operational excellence.

The system leverages both the speed and scalability of the Lightning Network and blockchain technology to ensure consensus on the sequence and validity of IoT events, creating an indisputable and tamper-evident record.

Key to this system is its use of **multi-party channel routing** combined with **advanced signing methods** using INFINEON security chips to secure sensor data before transmission. This system not only ensures the integrity, authenticity, and security of IoT event data but also provides a progressive step in enhancing data management in IoT environments through decentralized channels.

2.1 System Objective

The objectives of using the Lightning Network within an IoT ecosystem focus on **data integrity, enhanced security, and efficient data management**. Below is a detailed description of these objectives:

- **Data Integrity and Verifiability**:
    - **Immutable Records**: By leveraging blockchain technology, data transmitted via Lightning channels can be anchored to the blockchain, ensuring it cannot be altered. The integrity of the data is assured, which is vital for applications where authenticity is paramount.
    - **Consensus through Multi-Party Routing**: By utilizing native Lightning Network mechanisms such as Hash Time-Locked Contracts (HTLCs) and multi-hop routing, data transmission is inherently distributed and cryptographically verifiable. Agreement across multiple participating nodes significantly reduces the risk of manipulation or loss.
    - **Redundant Multi-Path Delivery:** Built entirely on standard, open-source Lightning Network protocols, our architecture employs redundant and parallel routing to increase resilience and fault tolerance. Each record (e.g., sensor data) is sent concurrently across multiple independent paths, ensuring delivery even in case of partial network failure. This setup supports verifiability rules such as "2 out of 3 copies must arrive and match", enabling robust, trust-minimized data validation.
- **Enhanced Security Measures**:
    - **Secure Transmission**: The sensor data is signed using **INFINEON security chips** either at the sensor or at the gateway level before transmission over the Lightning Network.
    - **Multi-Node Routing**: Data is routed through multiple nodes, each controlled by different parties, adding redundancy and making it harder for malicious actors to compromise the entire dataset.
- **Data Management**:
    - **Event Standardization**: All IoT event data follows a standardized format for consistency across devices and efficient data processing.
    - **Data Handling Efficiency**: The system is optimized to handle large volumes of IoT data, ensuring smooth transmission, aggregation, and reassembly at the client end.

- **Atomicity and Trustlessness (Zero-Trust)**:
    - **Reliable Delivery**: By leveraging HTLCs, data can be securely transmitted through multiple Lightning nodes, ensuring either all packets are delivered correctly or the transaction fails. This **atomic** nature guarantees data reliability.
    - **Monitoring**: The system employs watchtower functions to monitor transactions and channels, providing additional security and the ability to intervene in case of malicious behaviour.

2.2 System Architecture

The architecture of the Lightning Network-based system is designed to manage IoT sensor data with **high security, precision, and integrity**. Each component plays a vital role in ensuring data is captured, signed, transmitted, and validated effectively. The components of the system include (see Figure 1 and Figure 2):

2.2.1 Sensor Setup and Signing

- **INFINEON Security Chip Integration**: Each sensor is equipped with an INFINEON security chip, ensuring that the data is signed at the hardware level before transmission. If the sensor cannot directly handle signing, a **gateway** equipped with the chip will manage signing, particularly for sensors e.g. connected via Long Range Wide Area Network (LoRaWAN) or other limited connectivity solutions.

2.2.2 Lightning Network Nodes

- **Multi-Node Data Transmission**: Sensor data, once signed, is extended to transaction data packets and is transmitted via the Lightning Network through multiple nodes. Figure 2 views an implementation example for the use case of a construction monitoring network (e.g., used for monitoring a tunnel under construction). The nodes are:
    - **Monitoring Contractor Node**: First node to receive the data from the sensor or gateway and to create packets for data transmission over Lighting Channels. The node is owned/operated by the monitoring contractor (= the company responsible for tunnel monitoring, e.g. Geodata).
    - **Project Owner Node**: Receives transaction data from Monitoring Contractor Node via Channel C and forwards it through the network to the target node in the backend system. The node is imagined to be owned/operated by and physically located at the project owner (e.g., the ministry or state department ordering the tunnel).
    - **Construction Company Node:** Receives transaction data from Monitoring Contractor Node via Channel B and forwards it through the network to the target node in the backend system. The process is running parallel to the above. The node is imagined to be owned/operated by and physically located at the company responsible for building the tunnel.
    - **Local Construction Supervision Node:** Receives transaction data from Monitoring Contractor Node via Channel A and forwards it through the network to the target node in the backend system. The process is running parallel to the above. The node is imagined to be owned/operated by and physically located at the company responsible for site supervision.
    - **Backend Node**: Receives transaction data from **Project Owner Node, Construction Company Node** and **Local Construction Supervision Node.** Reassembles the data packets and ensures the received data is complete and intact. The node is assumed to be owned/operated by the company responsible for data management (e.g. is part of a cloud platform like GeodataHub of Geodata).

- **HTLCs and Security**: Each data packet passes through the Lightning Network using **Hash Time-Locked Contracts** to ensure the transmission is secured and verifiable.

2.2.3 Target Node Verification

- **Reassembly and Integrity Check**: Once all data packets reach the **Target Node** in the backend system, they are reassembled into their original form. The integrity of the data is verified by checking the hashes, compared in case of multiple copies, ensuring the data received matches the original data transmitted.

2.2.4 System Architecture

Areas and Components

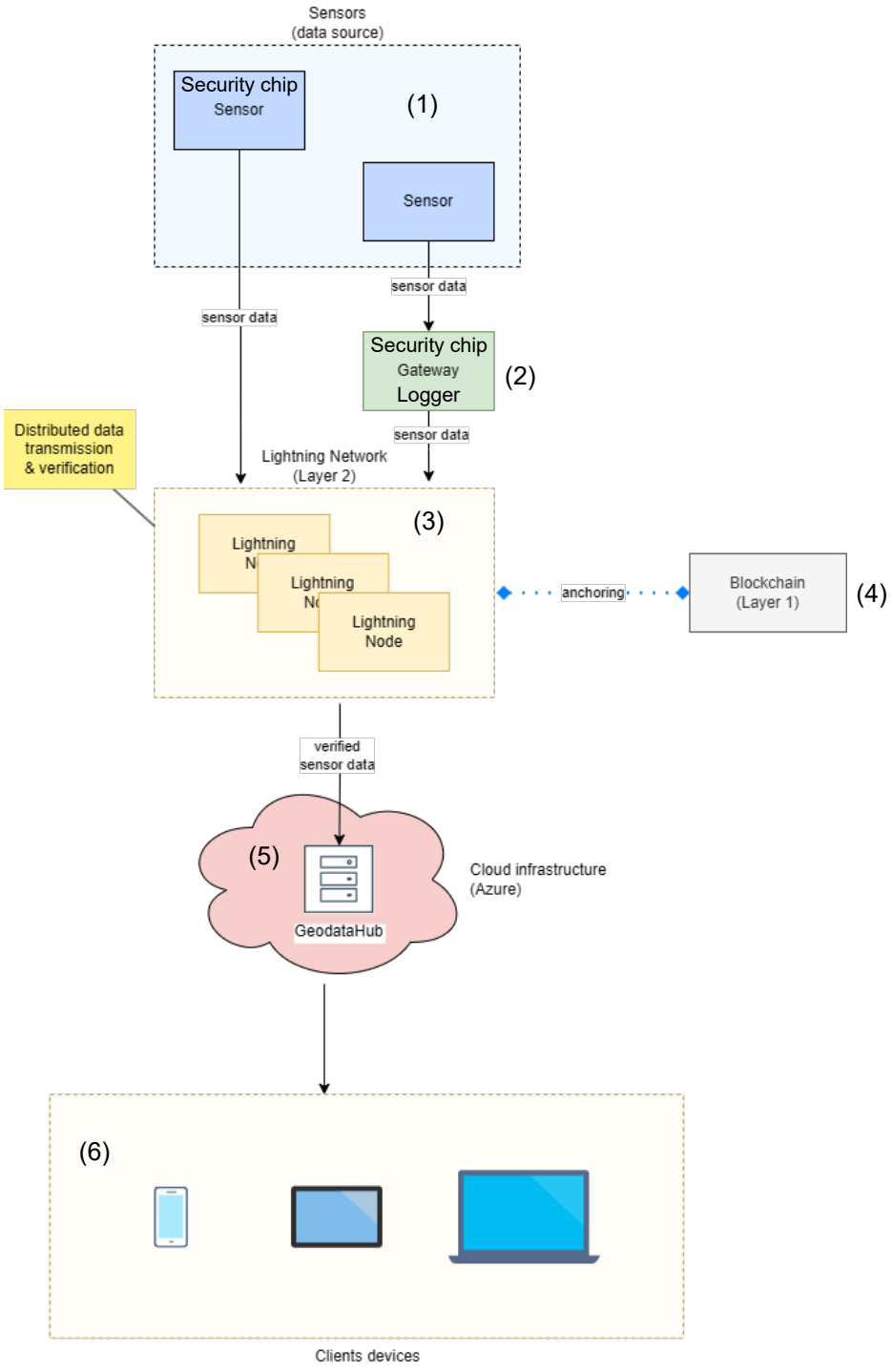

Figure 1. System architecture overview

The system (see Figure 1) consists of numerous (1) sensors (e.g., geotechnical sensors located in a tunnel under construction) and (2) gateways/loggers (e.g. installed at the tunnel portal and used for collecting, signing and transmitting the sensor data). Depending on the capability of the sensor, data signing with security chips is either done at the sensor or at the gateway/logger. The system further consists of several interconnected (3) Lightning nodes enabling the multi-channel routing of the signed data (e.g. established as single board computers located in the offices of the different parties of the tunnel project), (4) a blockchain to which the data is anchored automatically when Lightning Network (LN)-channels are closed, (5) a cloud-infrastructure (cloud database including webservices for data visualization, alarming and other uses) and (6) the client devices of the end users accessing and using the data.

Concept of Implementation

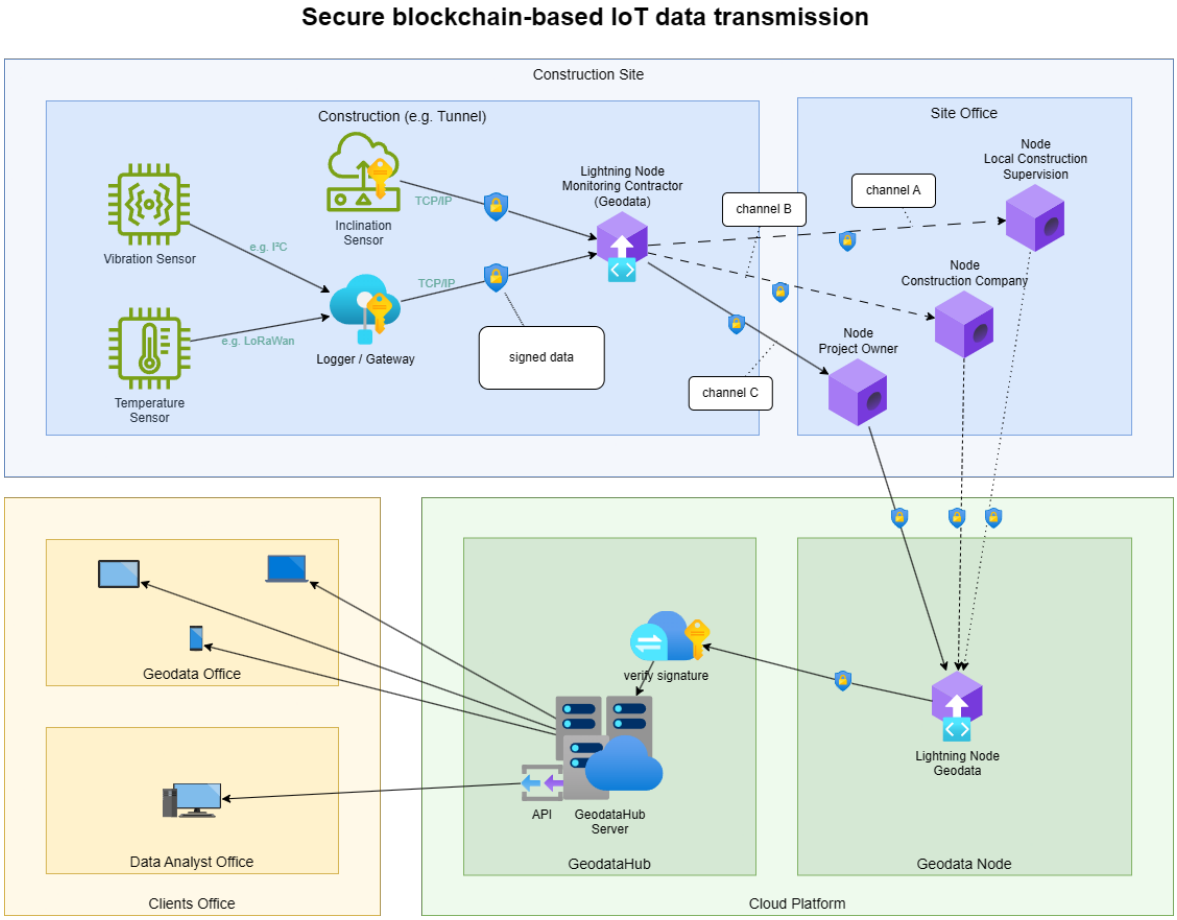

Figure 2. Concrete implementation example for the Tunnel Construction use case

Figure 2 shows an implementation example of the system for the tunnel construction use case. On the construction site are installed a temperature and vibration sensor sending their data via I^2C and LoRaWAN to the nearest logger/gateway where it is signed by a security chip. An additional inclination sensor is assumed to have the electronic capability of signing its data directly. The inclination sensor and the logger/gateway transmit the signed data to the first node of the LN. Here, a service checks the signatures for the first time and, if verified, the data is transformed into an LN transaction and further transmitted over three LN channels to three nodes located at the three main parties of the tunnel project, the Project Owner, the Construction Company and the local Construction Supervision. Each node sends the data further to the final node at the cloud-based data management platform (Geodata Node). Here, the signatures are verified again, and it is checked if the data coming in from the three parties are the same. If this is the case, the data is accepted and imported to the cloud platform/cloud server (GeodataHub). Now, the data can be accessed by the different end users and domain-specific applications.

2.2.5 Lightning Data Transmission

2.2.5.1 Multi-Party Channel Routing

In the Lightning Network, multi-party channel routing refers to a mechanism by which payments/or data are forwarded across multiple participants in the network, without requiring direct channels between the sender and the final recipient. Instead, the transfer is routed through a series of interconnected payment channels, much like a package being passed through multiple relay points on its way to the destination.

Suppose Alice wants to send a payment to Bob, but she doesn't have a direct payment channel with him. However, she does have a channel with Charlie — and Charlie, in turn, has a channel with Bob – e.g., Routing: Alice -> Charlie -> Bob (see Figure 3).

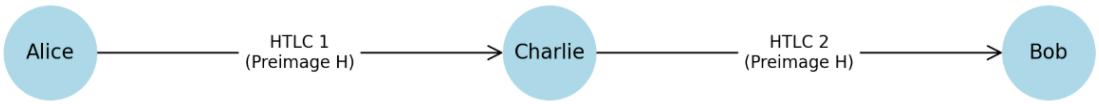

Figure 3. Multi-party channel routing

2.2.5.2 Advanced Multi-Party Channel Routing with redundancy

A redundant, parallel routing setup for sensor data transmission was built on top of the LN (see Figure 4) with the goal of:

- increasing reliability and fault tolerance
- ensuring that each measurement reaches the backend multiple times and independently via different paths

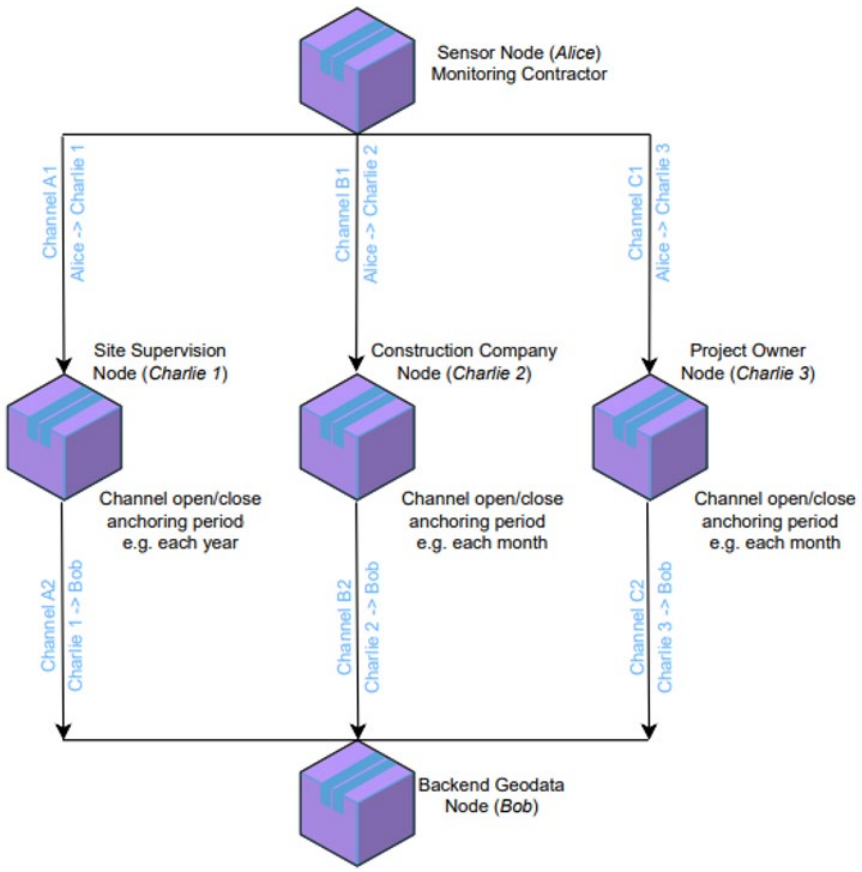

Figure 4. Redundant, parallel routing setup

2.3 Data Handling

2.3.1 Input Data and Transmission

- **Event Formation**: Sensor data is converted into events using a standardized format. Each event contains a unique identifier, timestamp, data items, the **device unique identifier (UID)** and a signature. This ensures that data from different sensors is uniformly represented.

```
{
    "uid": "unique_event_identifier",
    "timestamp": 1696248400000,
    "items": [
        {
            "key": "temperature",
            "value": 23.5
        },
        {
            "key": "humidity",
            "value": 55.6
        }
    ],
    "namespace": "organization/device_group",
    "device_uid": "device_001"
}
```

- **Data Transmission**: Sensor events are transmitted to a gateway or directly to the import service of the Sensor Node within the LN. From there, data is routed from the construction site to the backend system using the standard multi-hop routing mechanisms provided by the LN protocol.

2.3.2 Blockchain Anchoring Process

To complement data transmission over Layer 2, we periodically close and reopen Lightning channels in order to commit aggregated channel activity back to the Bitcoin mainchain (see Figure 5). This **Blockchain Anchoring** Process (Bitcoin Lightning Network Settlement) ensures that the state of the channels is periodically written to the immutable Bitcoin blockchain, providing an additional layer of transparency, auditability, and long-term integrity.

In our example setup, which features **redundant, parallel routing across three independent channels**, we take advantage of the architectural flexibility to configure anchoring intervals individually per channel. This allows us to align the anchoring frequency with application-specific requirements:

- One channel, dedicated to long-term archival, is closed and reopened only once per year.
- The remaining two channels are configured for more frequent anchoring, being settled monthly.

This staggered approach to Blockchain anchoring allows us to balance on-chain footprint, cost efficiency, and compliance with varying retention and verification policies across different data consumers.

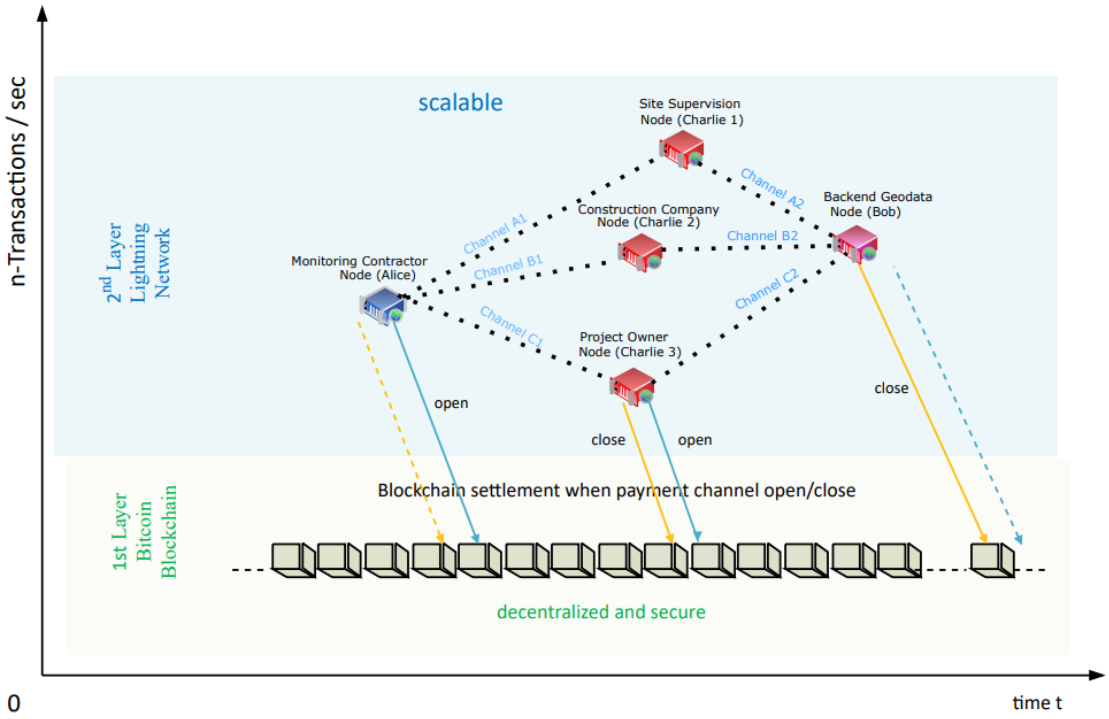

Figure 5. Bitcoin Lightning Network Settlement/Anchoring

Independently from the Lightning Network Bitcoin settlement also the state of the cloud database is periodically written to the immutable Bitcoin blockchain, another time providing an additional layer of transparency, auditability, and long-term integrity.

2.4 Network Nodes

The LN-based IoT logging system includes various user roles to ensure secure and efficient operation (see Figure 5):

2.4.1 Sensor Node [= Monitoring Contractor Node - Alice]

- **Role**: Responsible for maintaining sensors and ensuring secure data collection.
- **Responsibilities**:
  - Ensure that sensors are functioning and signing data with INFINEON chips.
  - Monitor sensors and manage secure connections to gateways or Lightning nodes.

2.4.2 Routing Nodes [= Charlie1 – Charlie3]

- **Role**: Manage the LN nodes through which the data is transmitted.
- **Responsibilities**:
  - Maintain secure connections and facilitate data flow between nodes.

2.4.3 Backend Node [= GeodataHub - Bob]

- **Role**: Ensure the integrity of the final received data.
- **Responsibilities**:
  - Reassemble data from multiple routing nodes and verify the signatures (hashes).
  - Check that data received from the multiple routing nodes match (e.g. applying the rule “2 out of 3 must match”
  - Log received events and ensure their immutability.

2.5 Example Workflow

A typical data transmission workflow using the Lightning Network:

1. **Event Generation**:
   - A sensor detects the temperature and signs the data using an **INFINEON chip**.
   - The signed data is formatted into an event (see below) that is sent as a Lightning Transaction (see Figure 6).

```
{      // event data
       "uid": "7488f8aa-97c7-4c4f-a546-f06a97bb0c63",
       "timestamp": 1696248400000,
       "items": [{"key": "temperature", "value": 25.3}],
       "namespace": "company/weather-stations",
       "device_uid": "weather_station_001"
       "path": "Channel A"
       "signature": "3045022100dff1d77f2a671c5f3a2fc5f5c3c28e6b3d0d9a8d9e
..."
}
```

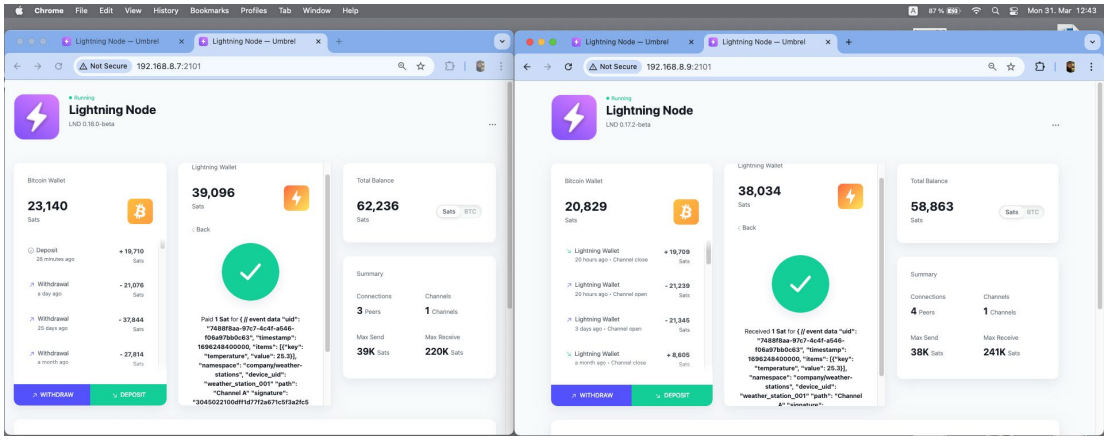

Figure 6. Event Data sent as Lightning Transaction

2. **Blockchain Anchoring**:

   See Section 4.2 for details.

3. **Data Transmission Input**:
   - The event is transmitted from sensor via a gateway to the import node service. This service verifies the signature and creates 3 similar packets for data transmission over 3 channels of the **Monitoring Contractor Node (Alice)**.

4. **Node Processing**:
   - **Monitoring Contractor Node (Alice) and Routing**
     - gets 3 Input Data Packets and creates Lightning Transactions 3Tx (containing the event data as payload).
     - these Lightning Transactions 3Tx will be sent parallel via the Routing-Nodes Site Supervision Node (Charlie1), Construction Company Node (Charlie2), Project Owner Node (Charlie3) to the Backend GeodataHub Node (Bob).

5. **GeodataHub Node Reassembly**:

- The **GeodataHub Node** (Bob)
  - receives all Event Data Transactions 3Tx via parallel Channels in Lightning Transaction Format.

6. **Verification and Logging**:
   - the Receiving Backend Node service verifies all signatures.
   - On parallel routing setup verifiability rules such as "2 out of 3 copies must arrive and match", enabling robust validation of the incoming data packets.
   - Stores the verified data and create logging entries.
7. **GeodataHub**
   - Sends the Sensor Event Data to the Import-Module of the GeodataHubServer.

2.6 Hardware Prototype

The **LN Data Transmission System** can be implemented using compact hardware units such as single-board computers.

Figure 7 depicts the hardware used to set up a prototype of the described system in the frame of the CISSAN project.

Each node of the prototype is based on an **Odroid N2+ minicomputer** equipped with Amlogic S922X processor with four 2.4 GHz Cortex-A73 cores and two 2 GHz Cortex-A53 cores and Mali-G52 Graphics Processing Unit (GPU). It has **4 Gigabytes (GB) of Random Access Memory (RAM)** and micro secure digital (microSD) card slot and embedded Multi-Media Card (eMMC) connector. The board features the most popular communication interfaces.

Shown in Figure 7 is a Raspberry Pi 4 Model B (Linux/Raspbian environment) used to log the data of two sensors, a temperature sensor and an inclination sensor, sign the data with an Infineon security chip, and transmit it to the Sensor Node (Alice).

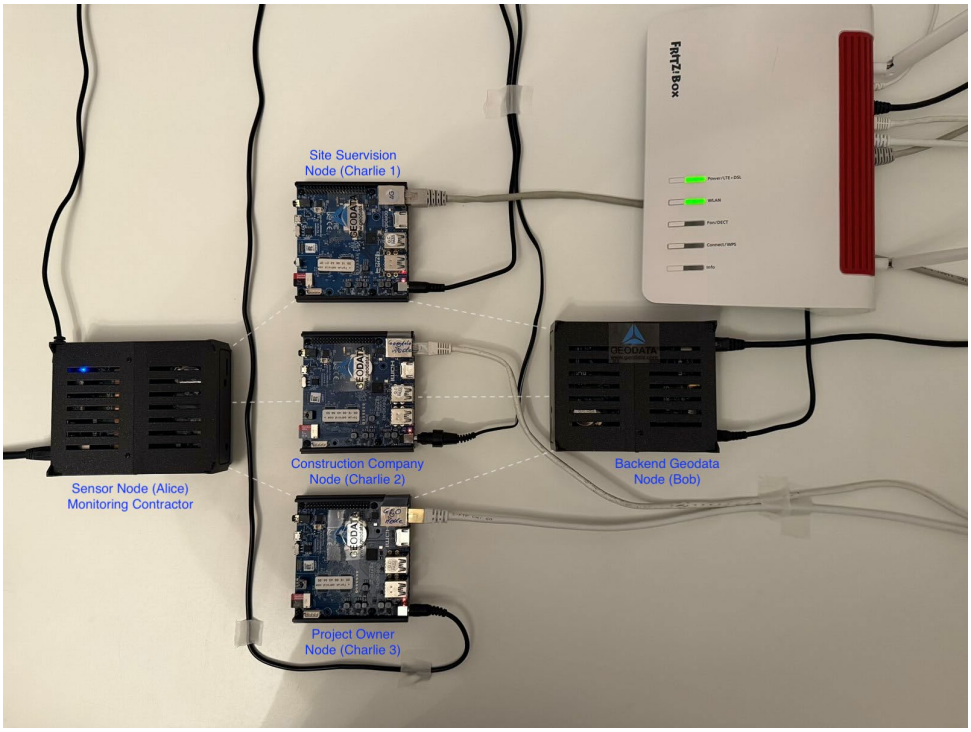

Figure 7. LN-Data-Transmission-System, hardware prototype based on single board computers

2.7 Software Prototype

This section presents a comprehensive overview of the software architecture, task distribution, programming languages, platforms, and selected software packages implemented on the nodes of a measurement and data transmission system based on the Lightning Network. Designed for robust, verifiable, and efficient sensor data flows, esp. in construction and infrastructure monitoring scenarios, the system leverages both custom-developed software modules and established LN standard software (Lightning Nodes), with multiple Linux-based embedded devices and cloud platforms forming the computational backbone. The software prototype has been developed specifically for implementing the demonstrator (see Section 9).

2.7.1 Node Types and Their Software Implementations

The prototype consists of seven specialized nodes, each with distinct tasks, roles, and technical implementations (see Figure 8).

2.7.1.1 Sensor and Geodata Logger

Role & Tasks:

- Reads measurements from connected sensors (e.g., temperature, inclination sensors).
- Generates a digital signature for each measurement event, ensuring authenticity and integrity of the data.
- Transmits the signed measurement data to the Geodata-Sensor Lightning Node (Alice) for further processing and relay within the Lightning Network.

Software Platform & Language:

- Custom-developed application using .NET (C#) running on a Raspberry Pi 4 Model B (Linux/Raspbian environment).

Key Implementation Details:

- Sensor interfaces realized via GPIO, I2C as available on the Raspberry Pi.
- Cryptographic functions implemented in C# using .NET cryptography libraries (e.g., System.Security.Cryptography and Infineon Custom libraries
- Communication with the Tier 0 Lightning Node via Google Remote Procedure Call (gRPC), encapsulating the signed data payloads.
- gRPC communication uses Google.Protobuf, Grpc.Net.Client and Grpc.Core
- Robust error-handling, logging, and retry logic to guarantee measurement delivery even under network disruptions.

2.7.1.2 Tier 0 Lightning Node

Instances:

- Geodata-Sensor [Alice]

Role & Tasks:

- Operates as Lightning Node, enabling secure and fast microtransactions.
- Acts as a single gateway to the LN.
- Relays payloads to 1st Tier Lightning Nodes.
- Participation in LN channels for decentralized, tamper-evident measurement data relay and multi-party approval/notarization schemes (joint data transmission), dispute resolution.

Software Platform & Language:

- Runs a standard LN implementation Node (Linux, ARM architecture).

Key Implementation Details:

- Lightning Node implementation: Lightning Network Daemon (LND) *polarlightning/lnd:0.18.5*.
- Preconfigured routing to respective 1st Tier nodes.
- gRPC endpoints provided for client and inter-node communication.
- Runs on an ODROID N2 hardware devices.

2.7.1.3 Tier 1 Lightning Nodes

Instances:

- Project Owner (PO) [R1]
- Construction Company (CO) [R2]
- Supervision (SU) [R3]

Role & Tasks:

- Operates as Lightning Nodes, enabling secure and fast microtransactions
- Relays payloads to 2nd Tier Lightning Nodes
- Participation in Lightning Network channels for decentralized, tamper-evident measurement data relay and multi-party approval/notarization schemes (joint data transmission), dispute resolution

Software Platform & Language:

- Identical to Tier 0 Lightning Node

Key Implementation Details:

- Identical to Tier 0 Lightning Node

2.7.1.4 Tier 2 Lightning Nodes

Instances:

- Backend PO [Bob1]
- Backend CO [Bob2]
- Backend SU [Bob3]

Role & Tasks:

- Operates as Lightning Nodes, enabling secure and fast microtransactions
- Serve as relaying targets/observers and receive payloads from 1st Tier Lightning Nodes
- Participation in LN channels for decentralized, tamper-evident measurement data relay and multi-party approval/notarization schemes (joint data transmission), dispute resolution Software Platform & Language:

Software Platform & Language:

- Identical to 1st Tier Lightning Nodes

Key Implementation Details:

- Identical to 1st Tier Lightning Nodes

2.7.1.5 Backend-Gateway

Role & Tasks:

- Queries Lightning Nodes for new measurement data in regular intervals or on demand.
- Checks the completeness of received records and verifies digital signatures (if present) for authenticity.
- Forwards validated data to the GeodataHub Application Programming Interface (API) for aggregation, visualization, and further analysis.

Software Platform & Language:

- Custom software developed in .NET/C#, deployed as a cloud service (Azure).

Key Implementation Details:

- Implements gRPC client interfaces for Lightning Node communication and Representational State Transfer (REST) client interfaces for GeodataHub API integration.
- gRPC communication uses Google.Protobuf, Grpc.Net.Client and Grpc.Tools
- Digital signature verification realized with .NET cryptography libraries.

2.7.1.6 GeodataHub

Role & Tasks:

- Central platform for managing and visualizing all measurement data from field nodes.
- Provides user interfaces for project configuration, data exploration and visualisation, reporting, and analytics.
- Maintains long-term storage, data integrity, and comprehensive access control.

Software Platform & Language:

- Frontend developed in Angular/TypeScript as Single Page Web App
- Backend developed in .NET/C# (REST API and business logic) running as Azure Web App
- Databases: PostgreSQL (relational data), Neo4J (graph and relationship data) and Blob Storage on Azure Cloud infrastructure.

Key Implementation Details:

- Implements advanced data models for measurement types, provenance, and time-series analytics.
- Supports integration with external systems via standard APIs.
- Role-based access and authentication leveraging OAuth/OpenID.

2.7.2 Software Package Summary

- Lightning Node Software: LND (polarlightning/lnd:0.18.5, open-source, widely deployed for Lightning Network operations).
- Containerization & Orchestration: Docker, Docker Compose.
- Programming Languages: .NET/C# (custom modules), Python and Bash (auxiliary scripts).
- Databases: PostgreSQL (structured data), Neo4J (graph data).
- Cloud Platform: Azure Cloud (for GeodataHub backend, databases, and Gateway deployment).

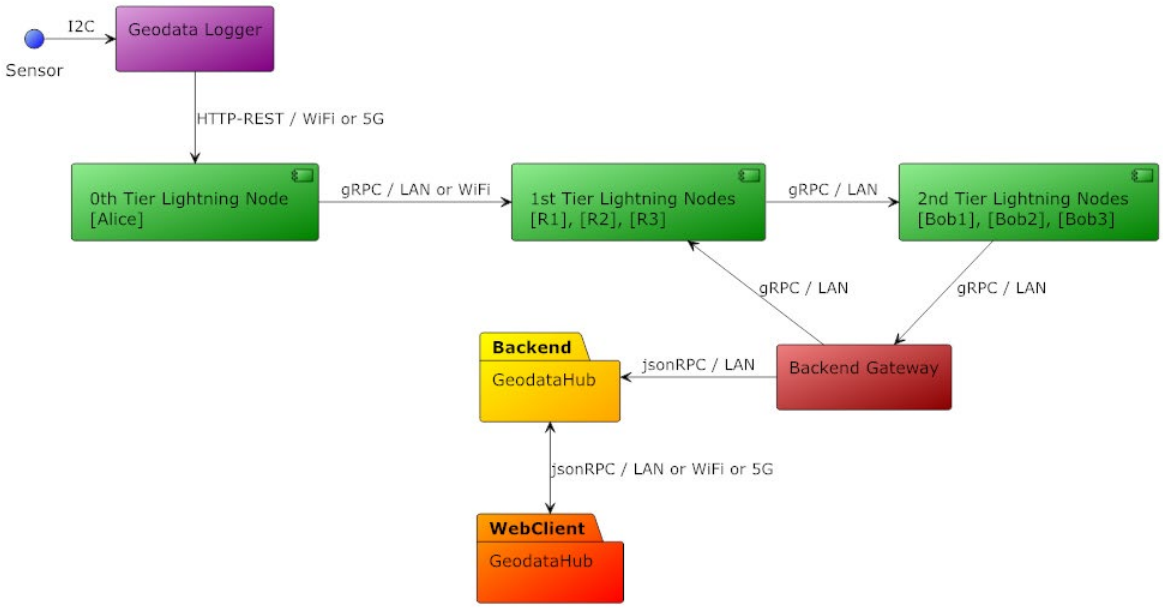

Figure 8. LN-Data-Transmission-System, software prototype architecture and used communication protocols, data formats and networks

2.8 Demonstrator

To showcase the developed system. a demonstrator consisting of 7 Lightning nodes has been developed to show the system in operation and simulate attack scenarios.

For the demonstrator, all the ODROID Lightning nodes are replaced by docker containers running the same software images: LND polarlightning/lnd:0.18.5

- Project Owner Nodes R1, Bob1: tier1_project_owner, tier2_geodata_po
- Construction Company Nodes R2, Bob2: tier1_company, tier2_geodata_co
- Supervision Nodes R3, Bob3: tier1_supervision, tier2_geodata_su

Figure 9 to Figure 13 depict 5 different system states of the demonstrator ranging from normal operation to operation while being attacked.

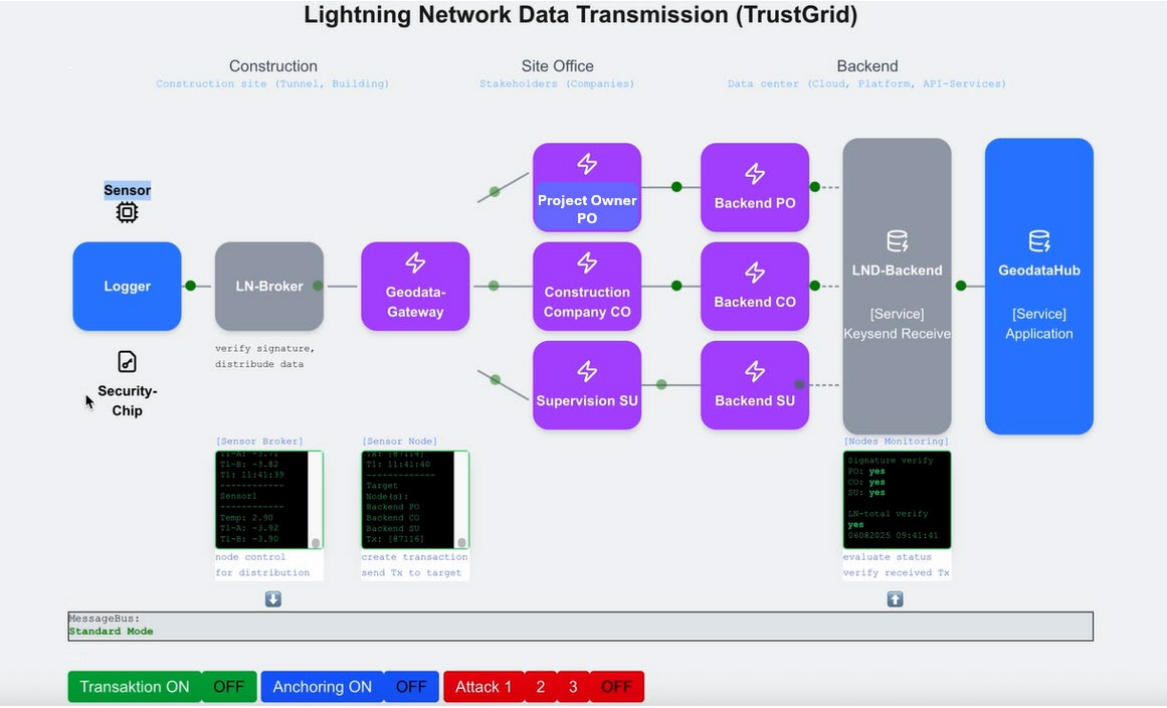

Figure 9. System demonstrator in state 1: normal operation

State 1 (see Figure 9): Normal operation, the sensor data is acquired and signed correctly, the signatures are successfully verified already at the Monitoring Contractor Node (LN-Broker) and again at the Backend Node (LND-Backend), all transaction records/data packets (shown as dots) are transmitted over the Lightning Network nodes to their final destination, the cloud platform GeodataHub.

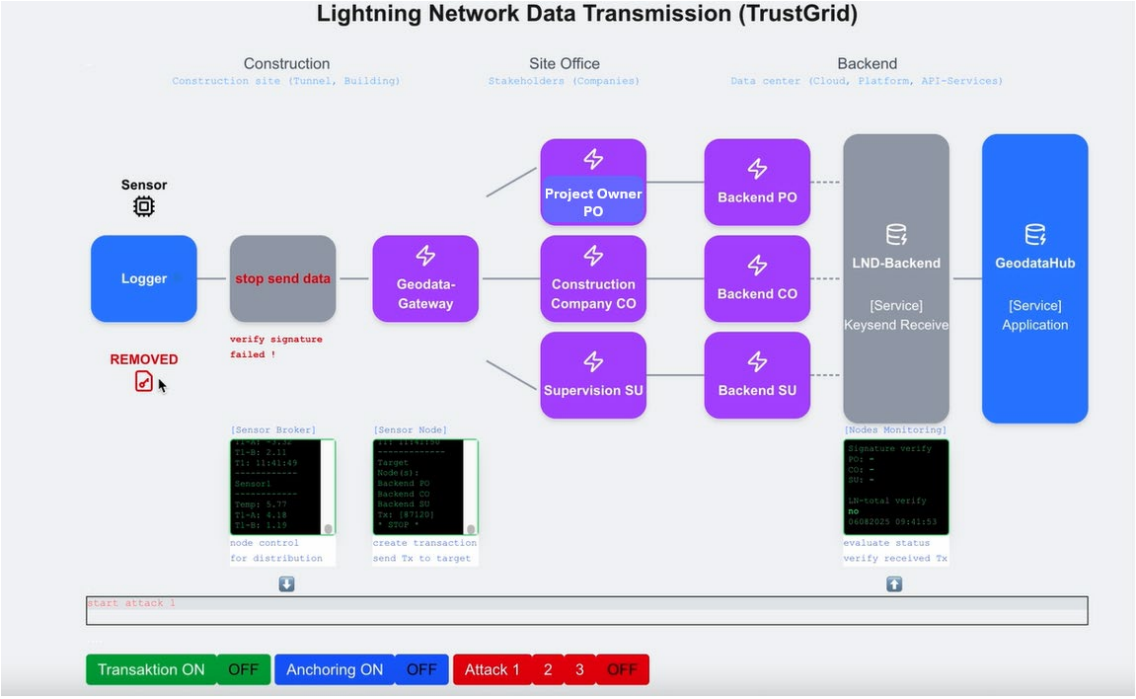

Figure 10. System demonstrator in state 2: security chip attack

State 2 (see Figure 10): Verifying the signature already fails at the Monitoring Contractor Node (Alice) because the logger's security chip has been removed, no data is transmitted any further from the Gateway.

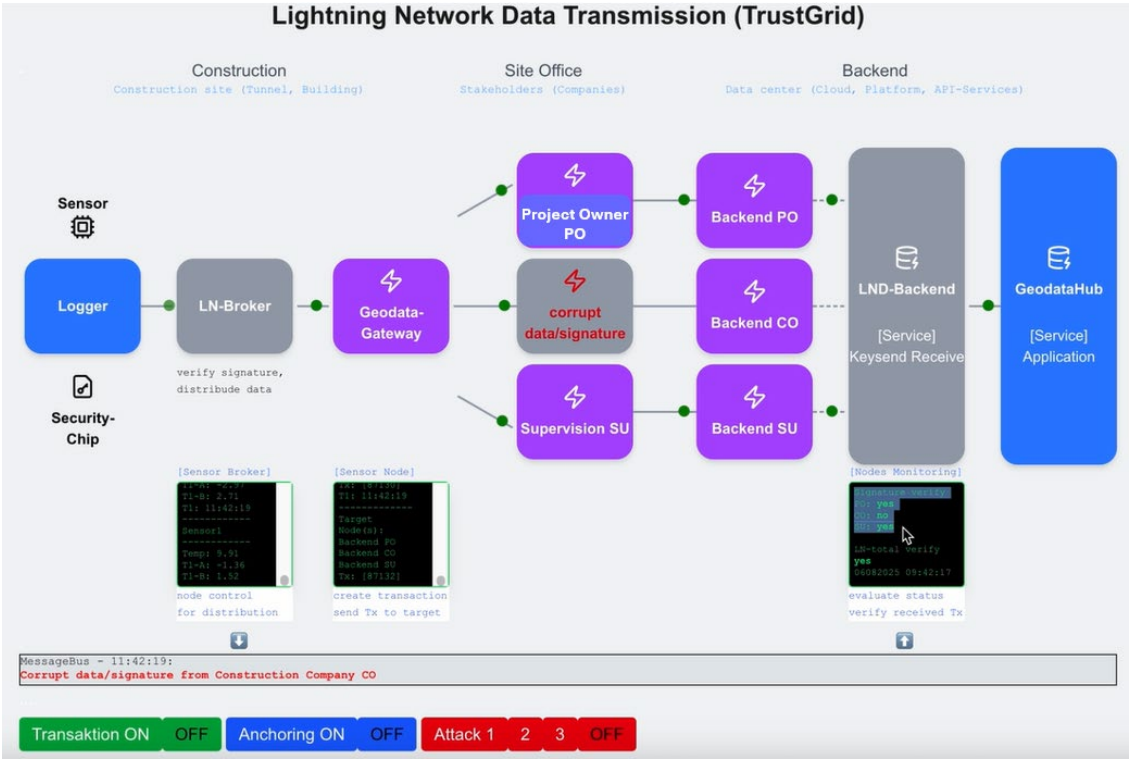

Figure 11. System demonstrator in state 3: Node attack, data tampering

State 3 (see Figure 11): The Construction Company Node is attacked, the data transmitted via this node is tampered, the signature is not verified by the Backend node. However, as the data of the two other routing nodes still arrive, match and their signatures are verified (="2 out of 3 rule"), their data is accepted by the Backend Node and passed to the cloud platform GeodataHub.

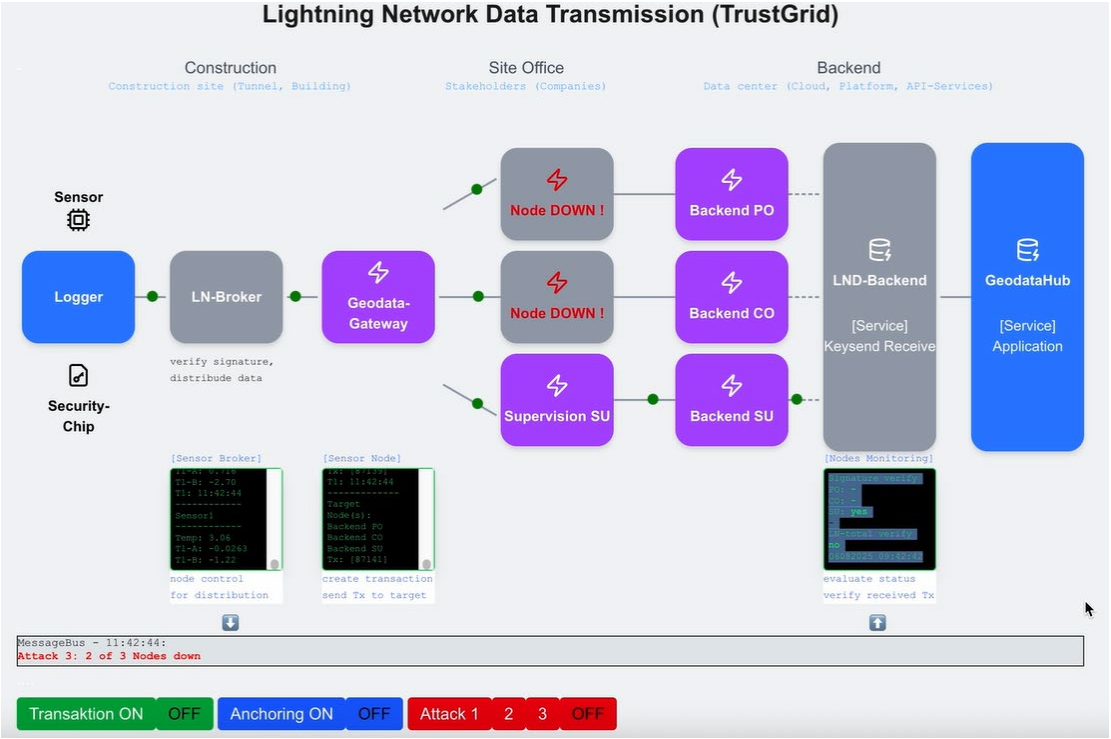

Figure 12. System demonstrator in state 4: Two nodes attacked/shot down

State 4 (see Figure 12): Both the Construction Company Node and the Project Owner Node are attacked and shut down. Although data from the remaining Supervision Node still arrives and its signature is verified, the data is not accepted by the Backend Node (because of the ”2 out of 3 rule”) and not passed to the cloud platform GeodataHub.

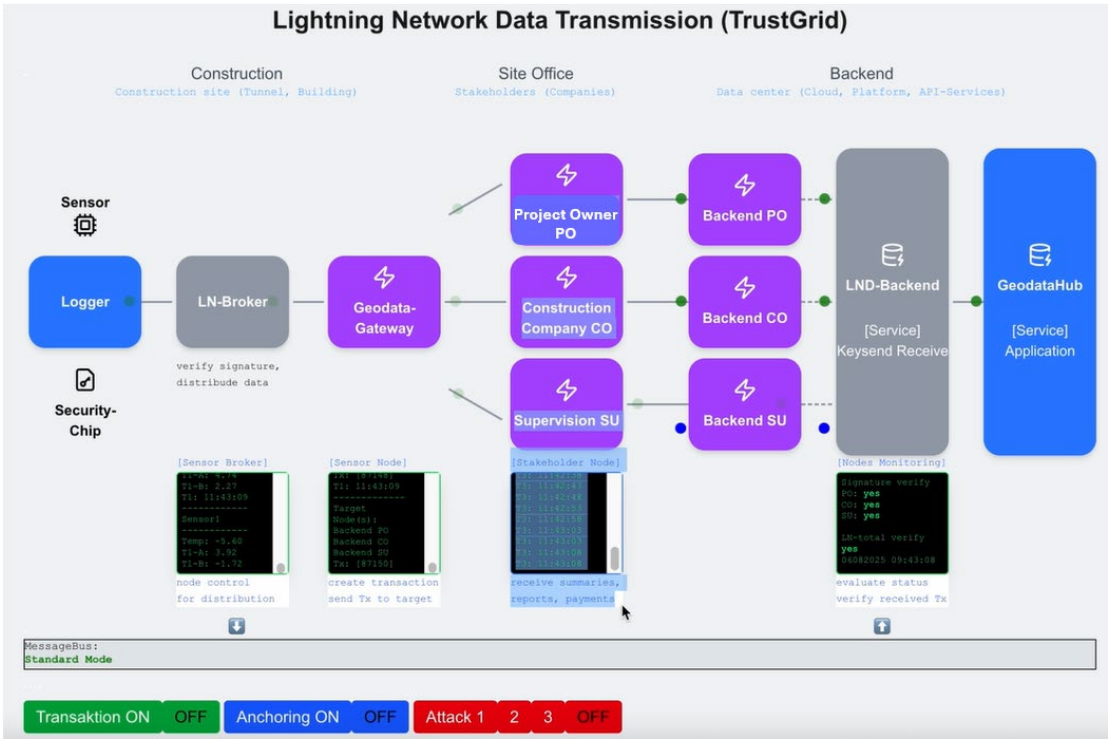

Figure 13. System demonstrator in state 5: blockchain anchoring

State 5 (see Figure 13): The LN channel from/to the Supervision Node is closed (e.g., every week). All data transferred within this week from/to the node is hashed and transferred to/permanently anchored to the Bitcoin Blockchain (shown as blue dots), this process is an automatic and inherent functionality of the Lightning Network.

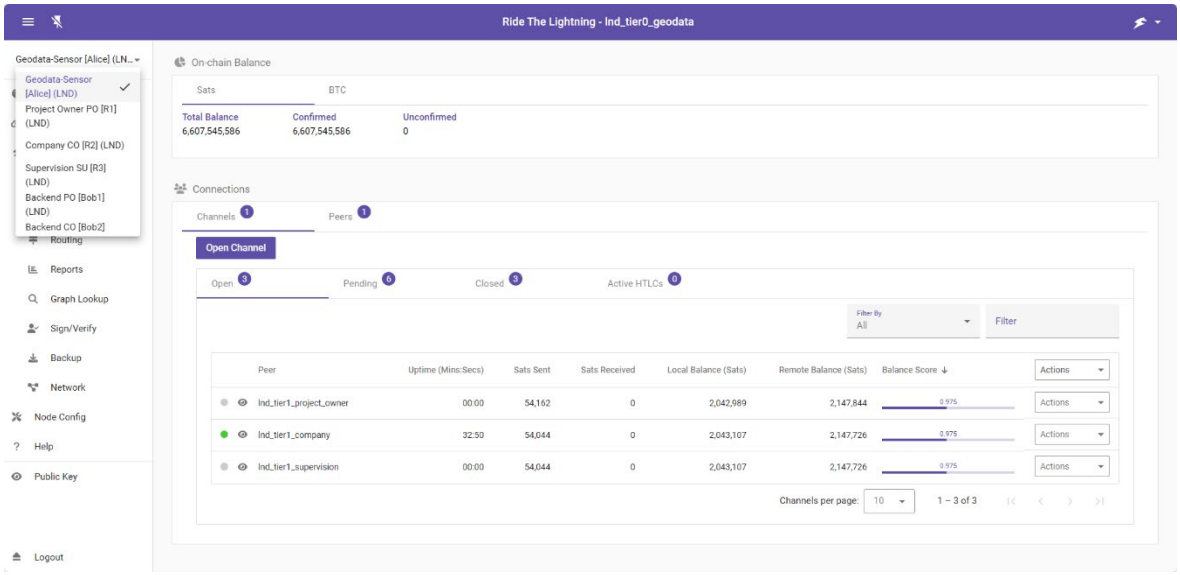

Figure 14. Lightning Network configuration details

Figure 14 shows the configured nodes and channels in the Lightning Network management software “Ride the Lightning”. The LN node “Geodata-Sensor (Alice)” (= “Geodata Gateway” in the figures above) is configured to forward the sensor data to three further nodes, the Project Owner, the Construction Company and the Site Supervision.

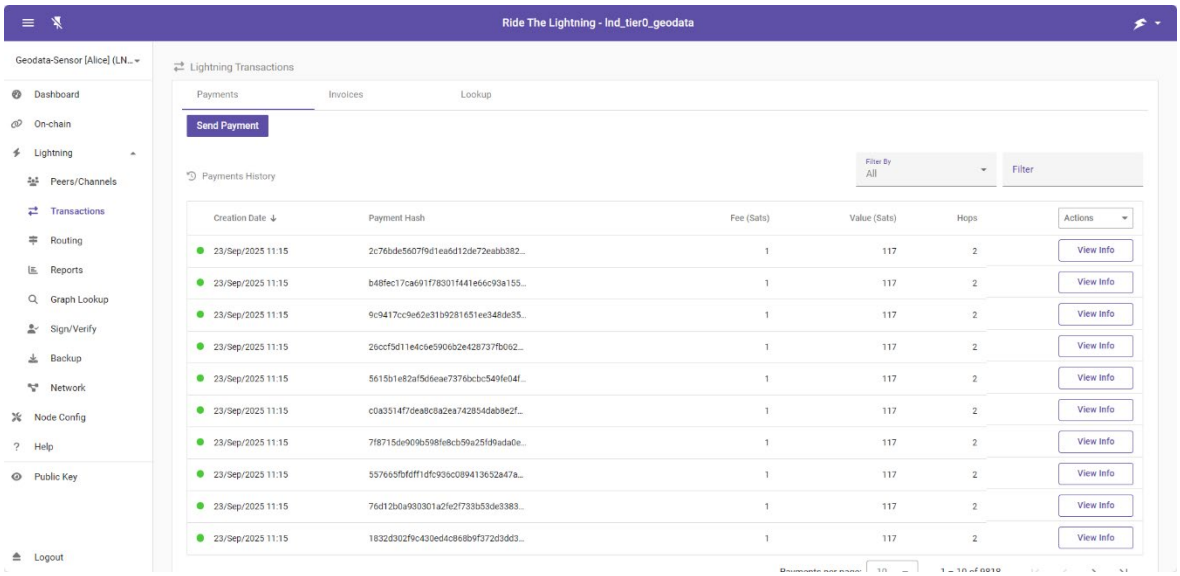

Figure 15. LN Transactions of the LN node “Alice”

Figure 15 shows the transactions of the LN node “Geodata-Sensor (Alice)” (= “Geodata Gateway” in the figures above) in the Lightning Network management software “Ride the Lightning”.

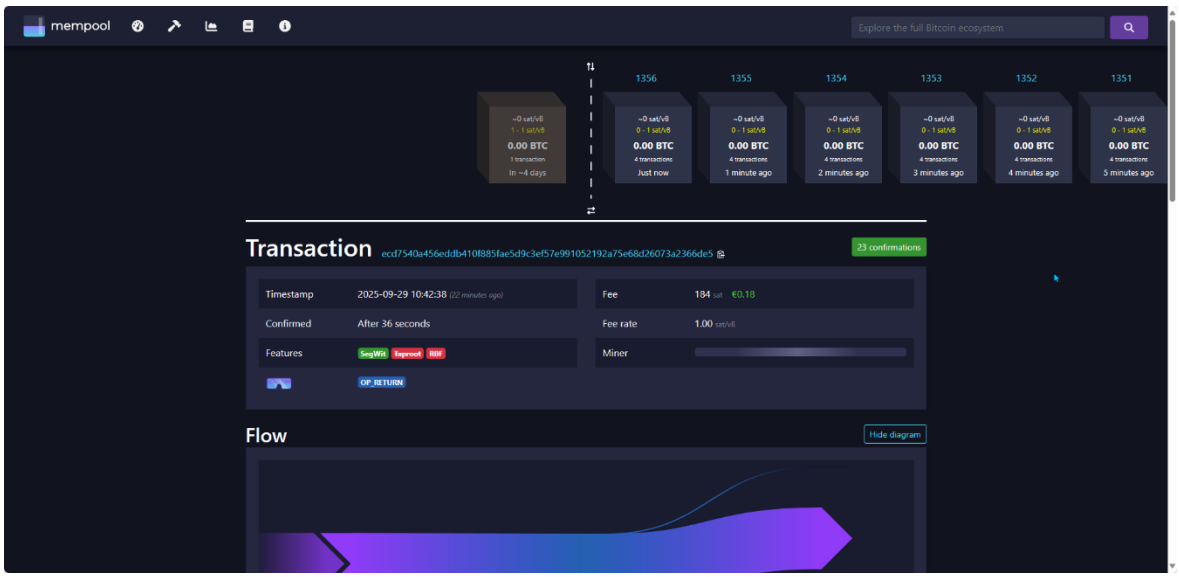

Figure 16. Blockchain anchoring in the Regtest Blockchain, transaction view

Figure 16 shows the blocks (built and under mining) and an LN transaction in the webtool “mempool” for the used Regression test (Regtest) Blockchain which is a local, private test network for developers that offers complete control over the blockchain state. Developers can adjust parameters such as block time and difficulty, mine blocks immediately, and test transactions without using real cryptocurrencies.

3. Eventchain Blockchain Logging System

The Councilbox Eventchain blockchain network logging system is designed to provide security and auditability for IoT ecosystems through a dual-blockchain architecture. Its primary purpose is to create an immutable, verifiable record of IoT events with integrated anomaly detection, ensuring both data integrity and real-time threat identification. By combining Byzantine Fault Tolerant (BFT) consensus with machine learning algorithms, the system establishes a robust framework for IoT data management that scales horizontally while maintaining cryptographic security guarantees.

This network logging system addresses critical challenges in IoT deployments including device authentication, trust score management and anomaly detection.

3.1 System Objective

The primary objectives of the Eventchain network logging system within IoT ecosystems focus on ensuring data immutability, Byzantine fault tolerance, integrated anomaly detection, and efficient device state management. Here is a detailed examination of these objectives:

Data Immutability and Verifiability:

- Dual-Blockchain Architecture: The system implements a two-tier blockchain structure with local hub blockchains synchronized to a master canonical chain. This ensures that once data is logged and consensus is achieved, it cannot be altered retroactively. Immutability is crucial for compliance auditing and forensic analysis in security incidents.
- Byzantine Fault Tolerance: Implementing a BFT consensus protocol with 51% quorum ensures that the system continues to operate correctly even when up to 49% of nodes are compromised or offline.
- Cryptographic Verification: Every transaction is signed using Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve, providing cryptographic proof of origin at the node level and ensuring non-repudiation for all logged events.

Enhanced Security Through Machine Learning:

- Integrated Anomaly Detection: The system embeds anomaly detection directly into the blockchain consensus mechanism, ensuring that every node independently validates not just the cryptographic integrity but also logging the behavioral normalcy of events.
- Multi-Layer Detection: Two complementary detection methods (statistical analysis, temporal frequency analysis) operate simultaneously at three sensitivity levels (1%, 5%, 10% contamination), providing comprehensive threat coverage.
- Deterministic Machine Learning (ML) Models: All nodes use identical model parameters ensuring that anomaly detection results are reproducible and verifiable across the network.

Device Lifecycle Management:

- Blockchain Key-Value Store (BKVS): The system implements a distributed key-value store on top of the blockchain, enabling efficient storage and retrieval of arbitrary data such as device states, configurations, and trust scores without requiring full blockchain traversal.
- Metadata API Abstraction: A RESTful API layer, designed to efficiently log device lifecycle events into the blockchain and retrieve them through simple queries by levering BKVS. It provides endpoints for device registration, trust score updates, anomaly detection, and blacklisting operations, enabling both immutable recording and fast retrieval of device state information.

Real-time Operations and Compliance:

- 20-Second Block Time: The system generates new blocks every 20 seconds, providing near real-time confirmation of events while balancing throughput with consensus overhead.
- Merkle Tree Optimization: Transactions are organized in Merkle trees, enabling efficient proof generation for individual events without exposing entire blocks. This tree-based architecture is critical for scalability, allowing parallel transaction processing and validation across multiple trees within a single block.
- Bitcoin Anchoring: Periodic anchoring to the Bitcoin blockchain provides an additional layer of immutability and establishes indisputable timestamps.

Integration and Interoperability:

- Multi-Protocol Support: The system accepts events via REST API and MQTT gateway. The MQTT gateway acts as a proxy to the REST API, translating MQTT messages into API calls.
- Zero-Trust Architecture: RabbitMQ message brokers implement queue isolation ensuring that compromised hubs cannot interfere with other participants' operations.

3.2 System Architecture

The Eventchain system implements a multi-tier architecture designed to balance decentralization with operational efficiency. Each component plays a critical role in ensuring data integrity, security, and scalability. The system architecture consists of the following integrated components:

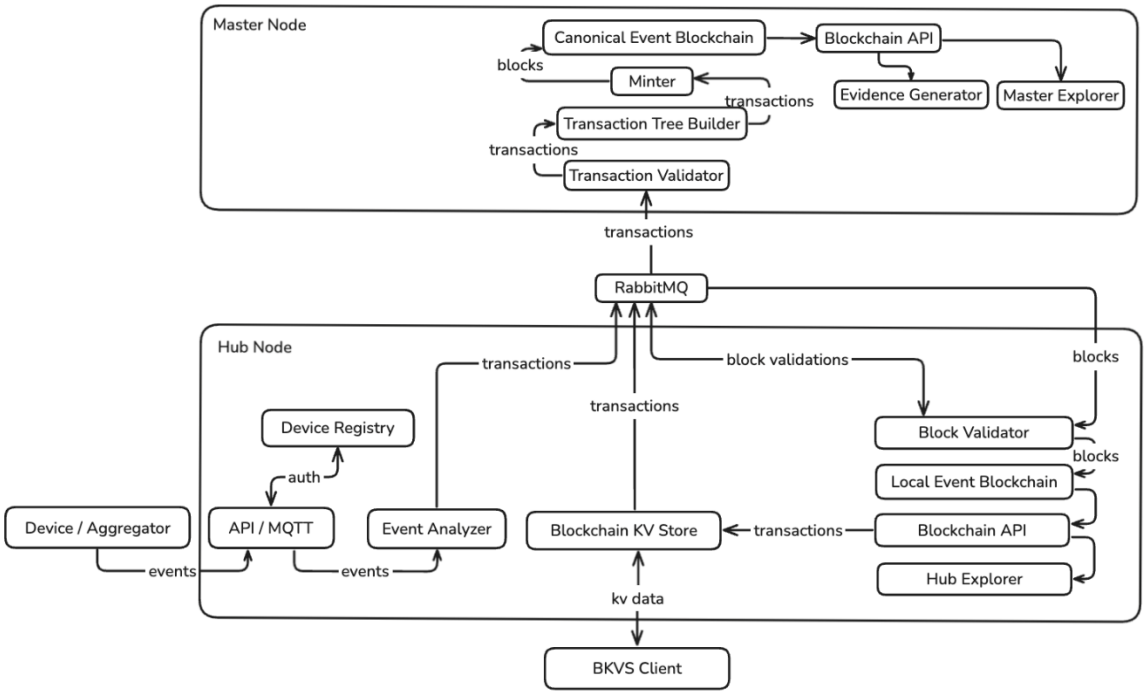

Figure 17. Eventchain System Architecture Overview

3.2.1 Master Node

The Master Node serves as the authoritative coordinator for the Eventchain network through seven coordinated services: the consensus minter for block generation, transaction bridge for aggregation, validation consumer for vote collection, block publisher for distribution, replay handler for synchronization support, and training instructor for ML model coordination. It provides REST API services and a blockchain explorer interface, implementing several critical subsystems:

Consensus Minter Service:

The minter service operates on a deterministic 20-second cycle, aggregating transactions from all hub nodes into new blocks. It implements a tree-building algorithm that organizes transactions into Merkle trees, optimizing both storage efficiency and proof generation speed. Each block is signed with the master's ECDSA private key using the secp256k1 curve, establishing cryptographic authenticity.

Transaction Processing Pipeline:

- Collection Phase: Transactions arrive via RabbitMQ from hub nodes, each pre-signed by the originating hub
- Transaction Bridge Service: Acts as the gateway between hub transactions and the master's pool:
  - Validates hub signatures using ECDSA verification
  - Checks transaction format and required fields (type, data, timestamp)
  - Ensures timestamps are within acceptable drift (±5 minutes)
  - Bridges validated transactions from individual hub queues to the master's transaction pool
- Tree Building: Validated transactions are organized into balanced Merkle trees with Secure Hash Algorithm (SHA) 3-256 hashing
- Block Assembly: Trees are combined with metadata (timestamp, previous hash, etc.) to form a complete block
- Broadcast: New blocks are distributed via fanout exchange to all participating hubs

3.2.2 Hub Nodes

Hub nodes represent organizational boundaries in the Eventchain network, operating through several specialized services: blockchain validator for consensus participation, distributed trainer for ML model updates, validation sender for vote submission, anomaly handler for detection processing, transaction processor for event handling, device registry interface and MQTT gateway bridge. Each enterprise operates one or more hubs to manage their IoT devices. This federated model provides autonomy while maintaining global consensus participation. Each hub maintains an independent device registry to handle event submissions through it.

Event Processing Pipeline:

1. Ingestion: Events arrive via REST API or MQTT with device tokens
2. Authentication: Tokens are validated against the local registry
3. Enrichment: Hub metadata and timestamps are added
4. Anomaly Detection: ML models evaluate the event
5. Transaction Creation: Event is serialized with anomaly scores
6. Signing: Hub's ECDSA private key signs the transaction
7. Propagation: Signed transaction sent to master via RabbitMQ

Block Validation Responsibilities:

When receiving new blocks from the master, each hub:

- Verifies the master's signature using stored public key
- Validates Merkle tree construction and root calculation
- Re-computes anomaly scores for all transactions using local models

- Compares computed scores with declared values (must match within tolerance)
- Sends signed validation vote if all checks pass
- Persists block locally upon seeing 51% validation quorum

3.2.3 Device Registry

The Device Registry provides authentication and authorization for event submission through a REST API, operating independently from the Metadata API's state management functions. This architectural separation ensures that device authentication remains distinct from trust scoring and lifecycle tracking:

API Endpoints:

- POST /devices - Create new device with unique ID and token (admin only)
- POST /devices/{device_uid}/token - Reset/rotate device token (admin or device)
- POST /login - Validate device_uid and device_token pair
- DELETE /devices/{device_uid} - Revoke device access (admin only)

Registration Workflow:

1. Device initialization with unique identifier generation
2. Token creation using cryptographically secure random generation (64-char hex)
3. Token hashing before storage in Device Registry
4. Token delivery
5. Device metadata sent to Metadata API
6. Metadata API records registration event in blockchain with all device information

3.2.4 RabbitMQ Message Broker Architecture

The system implements a dual-broker architecture providing security isolation between external cluster communication and internal master processing:

RabbitMQ Public Broker:

- Role: Bidirectional communication between master and hub nodes
- Hub to Master Communication:
  - eventchain-txs (direct exchange): Hubs send signed transactions to master
  - eventchain-validations (direct exchange): Hubs send validation votes for consensus
  - eventchain-replay-requests (direct exchange): Hubs request blockchain synchronization
- Master to Hub Communication:
  - eventchain-blocks (fanout exchange): Master broadcasts new blocks to all hubs
- Access Control: Master has admin privileges; each hub has restricted credentials with queue-specific read/write permissions enforcing zero-trust isolation

RabbitMQ Master Broker:

- Role: Internal master processing pipeline for transaction handling

- Architecture: Isolated Docker network inaccessible to hub nodes
- Processing Flow:
  - Transaction Bridge: Receives transactions from RabbitMQ Public Broker, forwards to internal-txs exchange
  - Tree Builder Service: Consumes from internal-txs, organizes transactions into Merkle trees, publishes to internal-trees exchange (supports multiple replicas for horizontal scaling)
  - Consensus Minter: Consumes completed trees from internal-trees, creates blocks with 20-second interval
- Security: Complete network isolation prevents hub nodes from accessing master's internal processing
- Scalability: Tree builder can run multiple instances (2+ replicas) for high-throughput transaction processing

Queue Architecture:

- Persistent queues ensure message durability across system restarts
- Message retention enables hub synchronization after downtime
- Zero-trust fanout: Each hub accesses only its isolated queue (blocks.hub.{hub_id}) with cryptographically enforced RabbitMQ permissions

3.2.5 Blockchain Explorer

Each node (master or hub) can deploy an identical web-based blockchain explorer providing visibility into the distributed ledger.

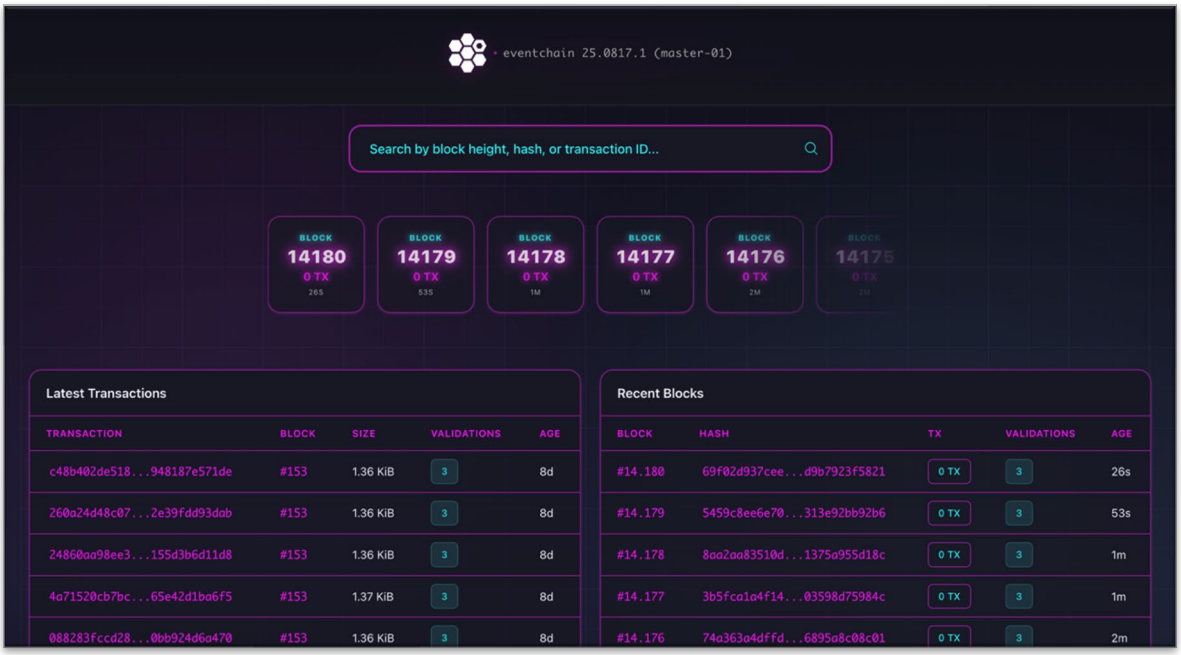

Figure 18. Eventchain Blockchain Explorer (main view)

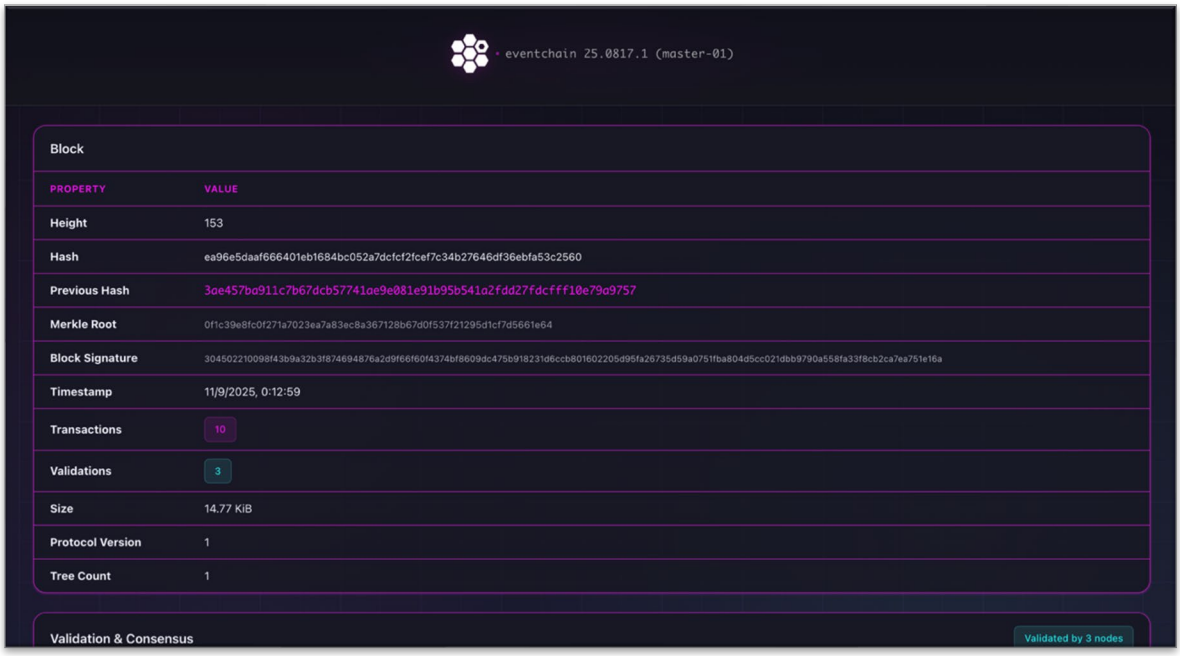

Figure 19. Eventchain Blockchain Explorer (block view)

Explorer Features:

- Real-time block display with automatic refresh
- Transaction search by hash
- Block search by height or hash
- Anomaly detection results visualization
- Display of validation votes per block

Blockchain API:

Each node exposes a blockchain API that the explorer and other systems use for data retrieval:

- GET /api/v1/blocks: Retrieve blockchain blocks with pagination
- GET /api/v1/blocks/{height_or_hash}: Get specific block details
- GET /api/v1/transactions: Query transactions with filters
- GET /api/v1/transactions/{tx_hash}: Retrieve individual transaction details
- GET /api/v1/proof/{tx_hash}: Generate cryptographic proof for transaction verification
- GET /api/v1/stats: Get blockchain statistics and consensus metrics

The blockchain API provides read-only access to the immutable ledger. The proof endpoint returns cryptographic evidence that can be independently verified without trusting the Eventchain network.

3.2.6 Blockchain Key-Value Store (BKVS)

The BKVS provides efficient state management on top of the immutable blockchain:

Architecture:

- Primary Storage: Key-value pairs stored in blockchain transactions

- Indexing Layer: SQLite database maintaining current state view
- Replication: Automatic state synchronization across all nodes

Operations:

- PUT: State updates create special BKVS transactions signed with the writer's private key
- GET: Direct lookup of values by key within the namespace derived from writer's public key
- DELETE: Logical deletion with tombstone markers

Namespace Isolation:

- Each writer has an isolated namespace computed from their public key
- Namespace = SHA3-256(public_key)
- Writers can only modify data within their own namespace
- Readers can query any namespace if they know the public key

Authentication works in the same way as with the Hub's Events API (via device_token).

3.2.7 Metadata API Integration Layer

The Metadata API, served exclusively by the master node, provides one of the main integration points for the CISSAN architecture, abstracting blockchain complexity behind familiar REST semantics.

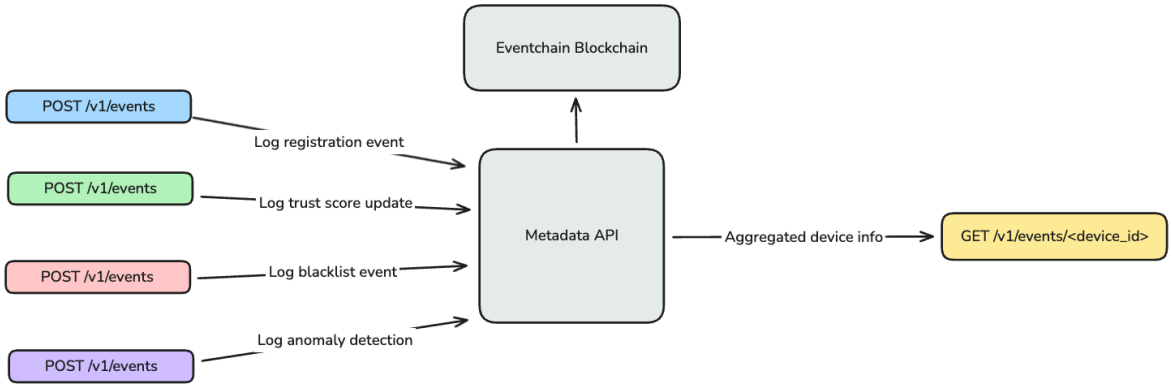

Figure 20. Metadata API Architecture

Authentication (JWT-based):

POST /v1/token

```
{
    "username": "cissan_client_id",
    "password": "cissan_client_secret"
}
```

Response:

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIs...",
```

```
    "token_type": "bearer",
    "expires_in": 3600
}
```

Core Endpoints:

POST /v1/events - Submit device meta events

Supported event types:

- registered: New device onboarding
- updated_trust_score: Trust metric changes
- blacklisted: Device isolation
- detected_anomaly: Security incident

GET /v1/events/{device_id} - Retrieve device latest events

Query parameters:

- events: Comma-separated event types to retrieve

3.3 Byzantine Fault Tolerant Consensus

The Eventchain system implements a Practical Byzantine Fault Tolerance (PBFT) variant optimized for IoT environments:

3.3.1 BFT Protocol Implementation

The consensus protocol operates in distinct phases ensuring agreement despite Byzantine failures:

- Phase 1: Block Proposal (Master)
    - Create block from transaction pool
    - Sign with master private key
    - Broadcast via RabbitMQ fanout
    - Hub-N Receives
- Phase 2: Independent Validation (Each Hub)
    - Verify signatures
    - Check hash chain
    - Validate Merkle trees
    - Recalculate anomalies
- Phase 3: Voting (If Valid)
    - Sign validation vote
- Phase 4: Quorum Detection (All Nodes)
    - Count validations >= 51%?
        - Persist block locally, otherwise, wait

Protocol Characteristics:

- No view changes: Simplified protocol without leader election
- No timeout: System waits indefinitely for quorum
- Deterministic: Same input always produces same result
- Non-blocking reads: Queries don't interfere with consensus

3.3.2 51% Quorum Mechanism

The system requires a simple majority (51%) of configured nodes to validate each block:

Quorum Calculation:

```
quorum_threshold = ceil(total_nodes * 0.51)
```

Examples:

- 3 nodes: requires 2 validations
- 5 nodes: requires 3 validations
- 10 nodes: requires 6 validations
- 100 nodes: requires 51 validations

Validation Requirements:

Each validating node must confirm:

1. Master signature is valid (ECDSA verification)
2. Block hash matches computed hash (SHA3-256)
3. Previous block hash exists in local chain
4. Merkle roots are correctly calculated
5. Timestamp is within acceptable drift (5 minutes)
6. Anomaly scores match local computation

Validation Vote Format and Flow:

When a hub approves a block, it sends a structured validation vote containing:

- block_height: The height of the block being validated
- validator_id: The hub's unique identifier from eventchain.yaml
- signature: ECDSA signature of the block hash using the hub's private key
- timestamp: When the validation was performed

The validation consumer service on the master node:

1. Collects validation votes from the RabbitMQ validation exchange
2. Verifies each vote's signature against the hub's public key
3. Maintains a counter of valid votes per block height
4. When votes reach the quorum threshold (51%), marks the block as confirmed
5. Broadcasts validation results via fanout exchange for transparency

Node Registry Configuration:

The system maintains a shared configuration file (eventchain.yaml) containing the public keys and network parameters for all participating nodes. This configuration file serves several critical functions:

- Public Key Registry: Contains ECDSA public keys for the master node and all hub nodes, enabling cryptographic verification of signatures during consensus
- Quorum Definition: Specifies the quorum percentage (51%) used for consensus calculations
- Trust Anchoring: Establishes the cryptographic identities that are trusted to participate in consensus

This shared configuration ensures that all nodes use the same public keys when validating signatures, maintaining consensus integrity. When new hub nodes join the network, they are added to this configuration file and distributed to all existing participants.

3.3.3 Block Validation Process

The validation process ensures both cryptographic integrity and semantic correctness:

Cryptographic Validation Steps:

Each validating node performs the following checks:

1. Signature Verification: Validates the master's ECDSA signature using the public key from the node registry
2. Hash Verification: Recomputes the block hash by combining previous hash, Merkle root, timestamp, and tree data, then compares with the declared hash
3. Chain Continuity: Verifies that the previous block hash exists in the local blockchain
4. Merkle Root Validation: Recalculates Merkle roots from transactions and validates against declared values

Anomaly Validation Process:

For each transaction in the block, the validator:

1. Extracts Event Data: Retrieves the original IoT event from the transaction
2. Recalculates Anomaly Scores: Uses local ML models to compute
3. Compares Results: Checks that locally calculated scores match the declared scores
4. Validates All Contamination Levels: Ensures consistency across contamination thresholds

The block is only accepted if all cryptographic validations pass and anomaly scores match.

3.4 Data Handling

3.4.1 Input Data and Event Processing

The Eventchain system processes IoT events through multiple ingestion pathways, each optimized for different device capabilities and network conditions:

Event Schema:

Events follow a standardized JSON structure containing:

- Event type classification (sensor_reading, status_update, alert)
- Unique device identifier

- Unix timestamp in milliseconds
- Namespace for logical grouping
- Key-value pairs for event data (temperature, humidity, etc.)
- Optional metadata (firmware version, battery level, signal strength)

Processing Pipeline:

1. Ingestion Layer:
   - REST API: JSON validation against schema
   - MQTT: events proxied to the API

2. Validation Layer:
   - Device authentication via token lookup
   - Timestamp verification
   - Schema compliance check

3. Analysis Layer:
   - Anomaly detection and labelling

3.4.2 Transaction Formation

Events are transformed into blockchain transactions with comprehensive metadata:

Transaction Structure:

```
{
    "hash": "0x7f9fade1c0d3b82c...",
    "type": "iot_event",
    "hub_id": "hub_01",
    "signature": "0x3045022100...",
    "timestamp": 1703001234567,
    "data": {
        // Original event data
    },
    "anomaly_metadata": {
        "scores": {
            "0.01": {
                "statistical": 0,
                "temporal": 0
            },
            "0.05": {
                "statistical": 0,
                "temporal": 0
            },
            "0.1": {
```

```
                "statistical": 1,
                "temporal": 0
            }
        },
        "model_version": "1.2510.0"
    }
}
```


Signing Process:

1. Serialize transaction to canonical JSON
2. Hash with SHA3-256
3. Sign hash with hub's ECDSA private key
4. Attach signature to transaction
5. Verify signature locally before transmission

3.4.3 Blockchain Persistence

The system implements a storage strategy optimizing for both write performance and query efficiency:

Block Storage Structure:

- Height index: Sequential block numbers
- Hash index: SHA3-256 block hashes
- Timestamp index: Millisecond precision
- Transaction index: Nested tree structure
- Validation index: Hub votes tracking

Cryptographic Proof Generation:

The system implements a specialized proof format for transaction verification using LoR encoding:

- Position Encoding: Each hash in the proof path is prefixed with a position indicator:
  - 'l': Hash is on the left side of the concatenation
  - 'r': Hash is on the right side of the concatenation
  - 'o': Only node (no sibling) requiring self-hashing

- Compound Proof Structure: Transaction proofs consist of two components:
  - Tree Proof: Path from transaction to Merkle tree root within a block
  - Block Proof: Path from tree root to block hash including metadata
  - Format: tree_proof:block_proof concatenated with colon separator

- Proof Verification Process:
  1. Parse position indicators and hashes from proof string
  2. Reconstruct hash path using SHA3-256 with level-based salting
  3. Validate final hash matches the declared root

4. Return verification result with validated transaction hash

This proof format enables compact representation while maintaining cryptographic security, allowing any party to independently verify transaction inclusion without accessing the full blockchain.

Transaction Content Deletion:

The master node administrator can trigger the content deletion of specific transactions through administrative scripts. When a transaction is censored:

- The transaction hash is added to a censored transactions file on the master node
- The censorship instruction is broadcast to all hub nodes via RabbitMQ
- Hub nodes delete the censored transaction content upon receiving the instruction
- Censored transaction hashes are preserved in the file to maintain the list
- During model training, censored transactions are explicitly excluded from the training dataset
- This ensures deterministic model training across all nodes by excluding the same transactions
- The Merkle tree structure remains intact as transaction hashes are retained

After a block has been validated by the quorum and consensus is achieved, nodes can erase transaction content while keeping the transaction hashes since validated blocks with quorum consensus cannot be rewritten backwards. Thus, emptied transactions will continue to be considered consensus valid.

This exceptional mechanism allows for removal of problematic data while maintaining blockchain integrity and ensuring all nodes train models on identical datasets.

3.4.4 Bitcoin Anchoring Process

The Master Node periodically anchors blockchain state to the Bitcoin blockchain, providing external timestamp proof and additional immutability:

- Epoch formation: every N blocks, collect block hashes and build a Merkle tree
- SHA3-256 (Epoch Merkle root) = 0xabc123...
- Create Bitcoin Transaction

  OP_RETURN 0xabc123... (epoch Merkle root)
- Broadcast to Bitcoin Network
- Wait 3 confirmations
- Store Bitcoin TXID associated to the epoch Merkle root

Anchoring Schedule:

- Epoch-based anchoring every N blocks
- Bitcoin fee: Dynamic based on mempool within a range

Verification Process:

To verify an anchor:

- Fetches the corresponding Bitcoin transaction
- Extracts the OP_RETURN data from the Bitcoin transaction

- Compares the Eventchain epoch Merkle root with the anchored hash in Bitcoin

3.5 Anomaly Detection System

3.5.1 Multi-Method Detection Architecture

The Eventchain system implements a multi-method anomaly detection system that operates deterministically across all nodes, ensuring anomaly scores calculated by different nodes remain identical and can be verified through blockchain consensus. The architecture employs two complementary detection layers working in parallel: statistical analysis methods, and temporal frequency analysis. Each layer examines IoT events from a different perspective, providing comprehensive anomaly coverage.

When an IoT event enters the system, it undergoes feature extraction and simultaneous processing by all detection layers. The statistical layer applies five mathematical methods (IQR, Z-Score, MAD, percentile, and skewness). The temporal layer analyzes event timing patterns. Each method produces binary scores (0 for normal, 1 for anomalous) stored as transaction metadata. During consensus validation, all hub nodes independently recalculate and verify these scores.

3.5.2 Statistical Analysis Methods

The statistical layer implements five complementary methods with dynamic thresholds that adjust based on the contamination parameter. Interquartile Range (IQR) flags events falling outside the middle 50% of data distribution, robust against extreme outliers. Z-Score measures standard deviations from the historical mean, effective for detecting significant behavior deviations. Median Absolute Deviation (MAD) provides outlier-resistant alternative to standard deviation. Percentile-based detection flags events outside historical percentile boundaries. Skewness analysis detects distributional changes indicating systematic behavior shifts.

The statistical layer employs consensus voting when multiple methods detect anomalies simultaneously. Rather than requiring unanimous agreement, the voting threshold dynamically adjusts based on contamination parameter and active methods, reducing false positives while maintaining sensitivity.

3.5.3 Temporal Frequency Analysis

Temporal frequency analysis examines event timing patterns, detecting anomalies in when events occur rather than their content. The detector maintains historical timestamps for each namespace, calculating intervals between events and comparing against statistical distributions of historical intervals. Normal device behavior exhibits consistent timing patterns that deviations can indicate compromise or malfunction.

The method uses Z-score analysis with dynamic thresholds scaled by contamination parameter. It effectively detects abnormally short intervals, unexpected silence periods, and subtle spacing deviations.

3.5.4 Consensus-Integrated Validation

Anomaly detection integrates directly into blockchain consensus, ensuring detection results are cryptographically verified by all participants. When the master creates blocks containing transactions with anomaly scores, all hub nodes independently recalculate scores using local models and verify they match declared values. Score mismatches cause block rejection, preventing consensus without 51% validation quorum.

3.5.5 Distributed ML Training Synchronization

The master coordinates training across hubs using RabbitMQ fanout messaging. Training instructions broadcast block ranges (e.g., blocks 1000-2000), contamination levels (0.01, 0.05, 0.1), fixed random seed, and estimator count. All hubs independently train models on local blockchain

data with explicit censored transaction exclusion. Deterministic parameters guarantee identical models across honest nodes, critical for consensus validation. Training occurs upon a trigger by the master node.

3.6 Implementation Details

3.6.1 Software Components

The Eventchain system leverages a modern technology stack optimized for blockchain operations:

Core Technologies:

Programming Language and Runtime:

- Python 3
- Docker for containerization
- AsyncIO for concurrent operations

Web Framework Stack:

- FastAPI for high-performance REST APIs
- uWSGI application server for production deployments
- Co-routine-based networking

Cryptographic Components (Important Distinction):

- ECDSA with secp256k1 curve for digital signatures
- SHA-256 specifically for ECDSA signature generation/verification
- SHA3-256 for hashing (blocks, transactions, Merkle trees)

Message Queue:

- RabbitMQ with Advanced Message Queuing Protocol (AMQP) protocol
- Dual-broker architecture (Public and internal for the Master Node)
- Connection pooling with heartbeat monitoring

Storage Layer:

- SQLite with Write-Ahead Logging (WAL) mode for concurrent reads
- SQLite for blockchain persistence
- SQLite for BKVS implementation
- File-based model storage for anomaly detection

Anomaly Detection:

- Model serialization for distributed consensus

3.6.2 Hardware Requirements

The system is designed to operate on the following hardware configurations:

Master Node Requirements:

Minimum Configuration:

- CPU: 4 cores @ 2.4 GHz (x86_64)
- RAM: 8 GB DDR4
- Storage: 256 GB Solid State Drive (SSD)
- Network: 100 Mbps Ethernet
- Operation System (OS): Ubuntu 24.04 LTS

Recommended Configuration:

- CPU: 8 cores @ 3.0 GHz (x86_64)
- RAM: 16 GB DDR4 ECC
- Storage: 1 TB SSD
- Network: 1 Gbps Ethernet
- OS: Ubuntu 24.04 LTS

Hub Node Requirements:

Minimum Configuration:

- CPU: 2 cores @ 2.0 GHz
- RAM: 4 GB DDR4
- Storage: 256 GB SSD
- Network: 100 Mbps Ethernet
- OS: Ubuntu 24.04 LTS

Recommended Configuration:

- CPU: 4 cores @ 2.4 GHz
- RAM: 8 GB DDR4
- Storage: 1 TB SSD
- Network: 1 Gbps Ethernet
- OS: Ubuntu 24.04 LTS

3.6.3 Docker Deployment

The whole system is containerized for consistent deployment across environments with docker compose files for the master and the hub nodes.

4. Conclusion

This report presents two complementary blockchain-based logging systems developed by GeoData and Councilbox in the CISSAN project, each addressing different dimensions of IoT data security and integrity. Geodata's system combines the speed and scalability of the Lightning Network with the robustness of the Bitcoin blockchain to ensure consensus on the sequence and validity of IoT events, generating an indisputable and tamper-evident record. Councilbox's Eventchain system uses a dual-blockchain architecture with Byzantine Fault Tolerant consensus based on cryptographically signed validation votes, integrated machine-learning-driven anomaly detection using deterministic models, and a key-value store with namespace isolation for device state management.

The Lightning Network-based measurement and data transmission system described here demonstrates a modular, scalable, and secure architecture. By combining robust open-source Lightning Node implementations with custom-developed .NET/C# applications, the system provides reliable sensor integration, cryptographic verification, and distributed data access for all stakeholders in construction and infrastructure monitoring. Docker-based deployment strategies facilitate rapid demonstration and scaling, while cloud-based aggregation and analysis ensure that data is not only collected but made actionable for decision-makers and auditors alike.

In detail, the unique features of the system are:

- the use of security chips signing sensor data already at creation / as close as possible to the sensor,
- the use of the Lightning Network for the redundant transfer of sensor data and signatures in form of transactions over multiple nodes and channels from the field to the cloud platform,
- the verification of signatures at several points along the route,
- the comparison of data coming in from different channels and routes, and their rejection in case of inequality and
- the anchoring/immutable logging of sensor data in the Bitcoin blockchain

For the tunnel construction industry, the developed system clearly is a complete novelty and innovation providing an ultimate data security level.

Councilbox's Eventchain Logging System demonstrates that sophisticated IoT security can be achieved through software-defined architectures without specialized hardware requirements. The integration of machine learning directly into the blockchain consensus mechanism represents a significant innovation, enabling real-time anomaly detection that is cryptographically verifiable and agreed upon by all network participants. The Byzantine Fault Tolerant consensus with 51% quorum ensures system resilience against both failures and attacks, while the Blockchain Key-Value Store with Metadata API abstraction provides seamless integration with existing IoT management platforms.

Key achievements of the Eventchain system include:

- Technical Innovation: The successful integration of multi-method anomaly detection directly into blockchain consensus represents a novel approach to securing IoT data, ensuring that all nodes can independently verify anomaly scores with identical results, maintaining the trustless nature of blockchain while adding intelligent threat analysis capabilities. The validation vote system, where each hub sends cryptographically signed votes verified by the master's validation consumer service, ensures Byzantine fault tolerance with mathematical certainty.
- Scalability Demonstration: The horizontal scaling architecture ensures that the system can grow with organizational needs without fundamental redesign. The tree-based transaction architecture enables parallel processing of thousands of transactions per block, with each tree processed independently before being combined into the final block structure. Each hub operates independently while participating in global consensus through structured validation votes, allowing enterprises to scale their IoT deployments incrementally.
- Practical Implementation: The comprehensive Metadata API with JWT authentication demonstrates that blockchain systems can integrate seamlessly with existing enterprise architectures. The abstraction of blockchain complexity behind familiar REST interfaces reduces adoption barriers and accelerates deployment timelines.