Journal of
Sensor and Actuator Networks

Article

Distributed Ledger as a Service: A Web 3.0-Oriented Architecture

Francesco Chiti * **and Giorgio Gandini**

Department of Information Engineering, University of Florence, 50139 Florence, Italy; giorgio.gandini@unifi.it
* Correspondence: francesco.chiti@unifi.it

Abstract: This paper proposes a general and interoperable Web of Things (WoT)-oriented architecture to support a distributed storage application. In particular, the focus is on a distributed ledger service dedicated to machine-to-machine (M2M) transactions occurring in an intelligent ecosystem. For this purpose, the basic functional modules have been characterized and integrated into a comprehensive framework relying on an IOTA approach. Furthermore, a general protocol that is built upon an underlying publish-and-subscribe framework is proposed to support all the application phases. The proposed approach has been validated by a simulation campaign targeting the achievable latency and throughput and, further, by a qualitative analysis of high-level metrics, both pointing out several advantages in terms of interoperability, scalability, and mobility support, together with addressing some constraints affecting service availability and security.

Keywords: Web of Things; publish/subscribe architecture; MQTT; fog computing; distributed ledger as a service; IOTA

check for updates

Citation: Chiti, F.; Gandini, G. Distributed Ledger as a Service: A Web 3.0-Oriented Architecture. *J. Sens. Actuator Netw.* **2023**, *12*, 57. https://doi.org/10.3390/jsan12040057

Academic Editors: Mohamed Benbouzid, Leandros Maglaras and Mohamed Amine Ferrag

Received: 31 May 2023
Revised: 13 July 2023
Accepted: 18 July 2023
Published: 20 July 2023

1. Introduction

The term *Internet of Things* (IoT) refers to a network of autonomous interconnected devices, which can be classified into two categories: sensors that emit periodic or asynchronous data, and actuators that take action based on the output of sensors. The favorable reception of this paradigm is highlighted by the fact that the number of IoT devices has exceeded the world population since around 2010 [1]. This is largely due to the wide range of value-added services enabled by IoT technology, such as home automation, e-health, smart cities, industrial control, and vehicle automation, to name a few.

IoT networks typically consist of a large number of small, battery-powered devices, which often have limited resources in terms of memory, processing power, bandwidth, and energy. This can lead to two main issues, especially in Industry 4.0 environments, where the IoT as a Service (IoTaaS) paradigm is a key feature:

- The need for lightweight communication protocols designed for machine-to-machine (M2M) data traffic patterns. In the early years of IoT, many manufacturers developed their own Physical and Data-link Layer protocols, resulting in devices that were not interoperable with those produced by other manufacturers.
- The data generated by IoT devices are usually stored and processed by additional devices. The most common approach is to process data in a fog computing environment and to store them in a cloud-based database, a service typically provided by private companies. However, this creates a potential trust issue since these companies could potentially tamper with the registered data.

The development of heterogeneous and often incompatible technologies presented a significant obstacle to the provisioning of quality-oriented and ubiquitous IoT-based services. As a result, there has been a growing expectation for a common and widely adopted standard that could accommodate different existing technologies [2]. To address this issue, the World Wide Web Consortium (W3C) published the first architectural framework for the Web of Things (WoT) in 2017, with the goal of making all IoT platforms and application domains interoperable [3]. The information exchange occurring as a WoT domain is characterized by specific features, mainly dictated by device limitations in terms of energy supply, processing capacity, bandwidth, and intermittent connectivity. Moreover, communication patterns are opportunistic and event-triggered updates, integrating multiple data flows [4]. Accordingly, the WoT paradigm's main features are (i) the abstraction of devices into standard objects, (ii) the use of a common format of information representation, and (iii) the adoption of a communication protocol operating on top of the Transport Layer.

To address the last issue, the HyperText Transfer Protocol (HTTP) [5] was first evaluated. It represents a client–server protocol that is widely used to access Web pages and transfer data and information over the Internet based on a request/response model. The Constrained Application Protocol (CoAP) was further proposed to adapt HTTP format to constrained devices and to enable more advanced interaction patterns with respect to HTTP, e.g., support for multicast [6,7]. However, CoAP also operates according to a request/response model, which represents a limitation in IoT domains, where connectivity is often discontinuous and it is not straightforward to clearly assign client or server roles to devices.

The Advanced Message Queuing Protocol (AMQP) is a lightweight solution conceived for machine-to-machine (M2M) communications and specifically designed for reliability, security, provisioning, and interoperability purposes [8]. AMQP supports both request/response and publish/subscribe architectures [9]. According to this approach, information Producers or Consumers first create an *exchange*, whose name is broadcasted and allows them to discover each other. Subsequently, a Consumer creates a *queue* dedicated to that exchange, and the related messages successively received are matched to the queue via a process called *binding*.

On the other hand, the Streaming Text Orientated Messaging Protocol (STOMP) has been designed for asynchronous message passing between clients via mediating servers (usually called Brokers) [10]. STOMP's simplicity allows the on-demand creation of a client without the need for a specific Application Programming Interface (API). It can interoperate over many languages and platforms via a text-based wire format, with frames modeled on those of HTTP.

Finally, another more interesting candidate is represented by the Message Queue Telemetry Transport (MQTT) protocol designed for monitoring applications [11]. In contrast to previous protocols, it relies on the publish-and-subscribe paradigm, in which publishers (e.g., sensors) transmit data messages to a Broker, which in turn delivers such messages to interested entities, called subscribers [12]. This approach is extremely flexible but places complexity on the Broker. Furthermore, MQTT defines a lightweight header format and requires a small code footprint. In addition, a variant of this protocol, called MQTT for Sensor Networks (MQTT-SN), has been specifically designed to address typical IoT constraints by means of its optimized architecture, interfaces, and components [13].

In designing our proposal, we adopted the MQTT protocol because the inherent publish/subscribe model allows devices to remain in a low-power state until they are ready to send or receive relevant messages. In contrast, the request/response model requires devices to be continuously active in order to handle incoming requests. In addition, our system is supposed to operate across geographically separate networks, so its security features are of paramount importance. For this purpose, MQTT supports End-to-End (E2E) encryption and client authentication, making it a better choice than CoAP, which, despite being designed for IoT networks, is still a less mature technology in terms of security. In addition, we decided to not rely on STOMP since it was conceived for a wired medium rather than a wireless one, and it does not provide an effective E2E security mechanism by default. Finally, when compared with AMQP [14], MQTT is able to provide several benefits for extremely low power devices, since it achieves a more efficient use of communication links and requires fewer resources than AMQP, especially when data are sent in an asynchronous burst, and the loss rate is around 5–10%.

On the other hand, to tackle the aforementioned trust requirements, a potential solution for storage is to use Distributed Ledger Technology (DLT) [15–17], which leverages a peer-to-peer (P2P) network to create an immutable ledger of messages through cryptographic techniques. This ledger can be used to record economic, asset, or data transactions.

DLT approaches differ in the way that nodes register data and the consensus mechanism used to achieve a shared immutable view. In addition, the policies to access data (private vs. public) and network functions (permissioned vs. permissionless) also distinguish them. The public/permissionless solution is the most suitable for our system, as we want to minimize the need for a central authority controlling the network.

The most well-known kind of DLT is represented by the blockchain, which gained popularity with the publication of the Bitcoin whitepaper in 2008 by an author under the pseudonym Satoshi Nakamoto [18]. Currently, there are three recognized generations of blockchain, each expanding the original use cases and improving their functionalities. However, they all share some common elements:

- As shown in Figure 1, data are recorded in a *chain* of blocks, where each block contains the hash of the previous one. A block typically consists of a header with the necessary metadata and a series of messages to be registered.
- The consensus mechanism requires the payment of a fee before publishing a message. There are many different schemes, but usually, specialized nodes, called miners, generate blocks and append them to the chain. They select valid messages received through a gossiping protocol and collect a percentage of the fee as a reward for successful publication.

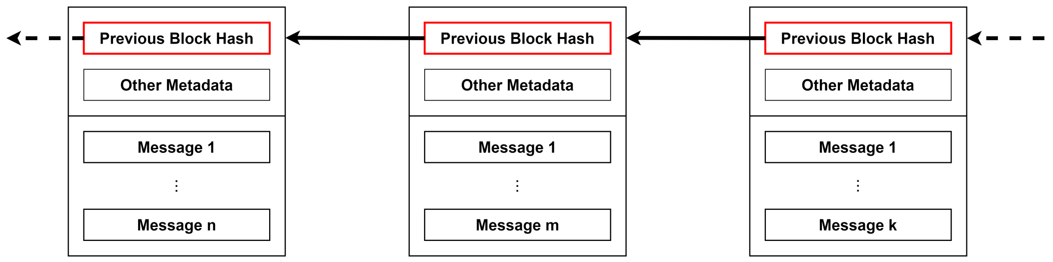

Figure 1. Example of a generic blockchain chunk.

Although recent publications proposed the use of public/permissionless blockchains to support IoT applications [19,20] and provided an overview of the evaluation of the achievable performance [21], their typical data structure, as well as their consensus mechanism, which is completely delegated to miners, makes them unsuitable for the purposes of this paper. The remarkable growth in the number of users [22] has resulted in miners accumulating transactions more quickly than they can process and publish into blocks. This can result in messages being left unselected since they are not received, leading users to resend their transactions after a certain time interval, thereby increasing the congestion (and blocking) probability. Moreover, IoT applications typically involve frequent transfers of small amounts of cryptocurrency or data, known as micro-transactions. A system that requires the payment of fees for each micro-transaction, even though they are small, would be economically impractical.

For these reasons, we have adopted a different approach in designing our solution. Without going into detail, an alternative could be represented by Hashgraph, but it is a patented option that can only be used by Hedera technology [23]. Instead, we chose to use IOTA [24], an open-source solution based on a Directed Acyclic Graph (DAG) that meets all of our requirements. More details on IOTA are further provided in Section 2.

The main contributions of this paper consist of the following aspects:

- The design of an original architectural framework capable of addressing the challenges associated with the DLTaaS paradigm for WoT domains. In detail, it involves the characterization of each module and the related communication interfaces. The framework also includes a data classification mechanism that is coupled with granular access policies, which makes it easier to retrieve data for applications while ensuring overall privacy.

- Technological mapping over widely available technologies, such as IOTA, MQTT, and MQTT-SN, by specifying the primitives and data format.
- The design of an integrated communication protocol.
- A performance evaluation conducted for different scenarios, pointing out the gains achievable in terms of overall latency and throughput.

After having briefly described the state of the art regarding solutions to interoperability and trust issues, this paper will cover the following key topics:

- Section 2: A high-level description of the selected software technologies for communication (MQTT and MQTT-SN) and storage (IOTA and Channels).
- Section 3: The definition of a modular and secure architecture that enables the system to be adapted to various application requirements, including mobility. This includes the design of a communication protocol for delivering data and signaling events.
- Section 4: A low-level numerical simulation of the most relevant metrics and a general performance analysis of the proposed framework in order point out the possible achievable advantages and address future developments.

2. Overview of Adopted Technologies

A viable approach to developing the proposed framework could have been based on the mere integration of IOTA technology and the Channels protocol, as they are specifically designed to support IoT networks. However, it was necessary to include the MQTT protocol for communication purposes. Both MQTT and Channels, as described in Sections 2.1.1 and 2.2.4, respectively, follow the publish/subscribe paradigm, but in different ways. MQTT employs optimization for message dissemination within the IoT ecosystem, while Channels involves the classification of encrypted data published on the IOTA distributed ledger, which can only be decrypted by specific users. Additionally, the complexity and resource-intensive nature of the IOTA software, along with the functionalities of Channels, make them unsuitable for deployment on devices with limited resources, which is often the case with IoT devices. In contrast, MQTT is implemented as lightweight software, making it well suited for installation on such devices, especially its MQTT-SN variant.

In this section, we provide a high-level description of the two selected technologies, focusing on their operating principles. The MQTT part also includes a description of the MQTT-SN variant, while the IOTA subsection focuses on the so-called Channels tool.

2.1. MQTT

MQTT is an Application Layer protocol developed by IBM in 1999. Since 2013, it has been standardized by the Organization for the Advancement of Structured Information Standards (OASIS) [11]. MQTT's latest version, 5.0, was released in 2019, and it provides an M2M communication model with three possible levels of Quality of Service (QoS) to achieve reliability through an acknowledgment and retransmission mechanism. This, combined with the benefits of the publish/subscribe model, previously introduced in Section 1, and the specific capabilities provided by the MQTT-SN protocol that are described below, makes MQTT the candidate communication protocol for the proposed framework.

2.1.1. Publish/Subscribe Model

The publish/subscribe model is based on the idea that certain nodes publish information, which is further received only by nodes interested in it. In MQTT, data are labeled with a Universal Resource Identifier (URI) string called the *topic*, which follows a tree structure (e.g., `root/parent1/leaf1`). Each character in the strings used is encoded in 8-bit Unicode Transformation Format (UTF-8). To create a tree-like structure, the "/" (U+002F) character is utilized. During the publish and subscribe phases, which will be briefly described in

Section 2.1.2, wildcards can be used to refer to multiple topics. The two special wildcard characters that can be used individually or simultaneously are as follows:

- Single-level wildcard "+" (U+002B): It matches a single level and can be used multiple times within a single usage. For example, the string `root/+/parent1/+` represents any topic starting with `root`, followed by any single-level string, then the level `parent1`, and ends with any single-level string.
- Multi-level wildcard "#" (U+0023): It matches any number of levels, including the parent and any subsequent child levels. For instance, the string `root/parent1/#` represents the topic `root/parent1` and all of its child topics.

A special node called the *Broker*, which can be a high-performance device or a hierarchical subnetwork, plays a crucial role in the network. Its address is known in advance to clients, and it is used to establish network connections using port 8883 for Transport Layer Security (TLS) or port 1883 for non-TLS. The Broker receives messages from clients and forwards them accordingly. Clients can assume the following roles with respect to a given topic:

- *Publisher*: generates messages, labels them with the topic, and sends them to the Broker.
- *Subscriber*: subscribes to the topic by communicating it to the Broker, from which it receives messages that belong to it.

The Broker builds a table, where each topic corresponds to a list of its publisher and subscribers, which can be dynamically updated due to (un)subscriptions. However, if not adequately protected, the Broker represents a single point of failure due to its communication centrality.

2.1.2. Pseudo-Primitives

The protocol operates using messages known as *Control Packets*, which have a standardized structure consisting of (i) a Fixed Header, (ii) an optional Variable Header, and (iii) an optional Payload. The Fixed Header, which is 2 bytes long, contains three fields:

- Type (4 bits): This field specifies the type of Control Packet being used.
- Flags (4 bits): The Flags field is a set of options that vary depending on the type of Control Packet.
- Remaining Length (8 bits): This field is an unsigned integer that indicates the length of the Control Packet beyond the Fixed Header, including the Variable Header and Payload, if present.

The Type field in the Fixed Header reserves the value 0, and the remaining $2^4 - 1 = 15$ values are used to identify different types of Control Packets, which can be categorized into four functional groups:

- Session: After establishing a network connection between a client and a Broker, the client sends a CONNECT (1) packet to the Broker to initiate an MQTT session. The client can indicate its willingness to use an authentication method, and in such cases, an exchange of AUTH (15) packets occurs between the client and the Broker. Once this phase is completed, the Broker sends a CONNACK (2) packet to the client, and the connection is established. To close a session, either the client or the Broker can send a DISCONNECT (14) packet.
- Publication: These functions involve sending data from the client to the Broker and subsequently disseminating it to subscribers. Depending on the Flags field in the Fixed Header, a publication can have three levels of Quality of Service (QoS). Level 0, or "At most once delivery", involves simply sending a PUBLISH (3) packet to the recipient. With level 1, or "At least once delivery", the sender retransmits the PUBLISH packet until it receives a PUBACK (4) packet within a specified time limit. Level 2, or "Exactly once delivery", limits the number of retransmissions by using the PUBREC (5), PUBREL (6), and PUBCOMP (7) packets between the sender and recipient.
- Subscription: To subscribe to one or more topics, a client sends a SUBSCRIBE (8) packet to the Broker. The subscription process is considered complete upon receiving a

SUBACK (9) packet. Likewise, to unsubscribe, there is an exchange of UNSUBSCRIBE (10) and UNSUBACK (11) packets.

- Activity check: The Broker can send a PINGREQ (12) packet to check the status of a client in case of inactivity. The client is expected to respond with a PINGRESP (13) packet. These packets are also used in the MQTT-SN variant to activate client devices in a low-power state.

For the sake of generality, this article is focused on describing the high-level behavior of the protocol using only the three pseudo-primitives mentioned in Table 1. An example illustrating these pseudo-primitives is provided in Figure 2.

Table 1. The three main operational pseudo-primitives in MQTT.

`publish(string topicURI, int QoS)`	This primitive is used by clients who wish to publish information. It is directed to the Broker and specifies the topic to which the information pertains, as well as the desired QoS level.
`notify(string topicURI, int QoS)`	This primitive is exclusively utilized by the Broker and is sent to subscribers of a particular topic to forward them a publish associated with it. It specifies the topic of the publish and the QoS level selected by the subscriber during subscription.
`subscribe(string topicURI, int QoS=NULL)`	This primitive is used by clients who wish to subscribe to a topic to receive notifies. It is directed to the Broker, and if QoS is not specified, the notifies are forwarded to the client with the same QoS level as the associated publish.

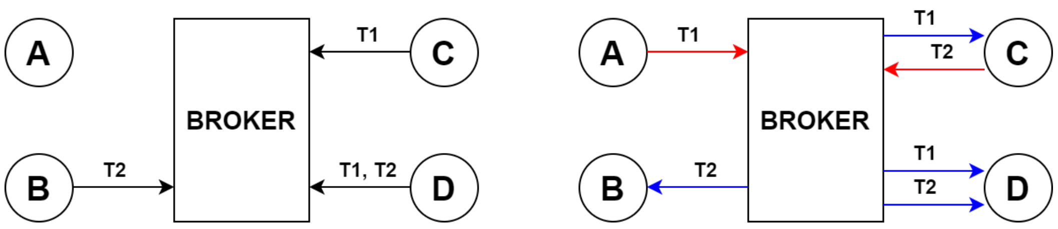

Figure 2. On the **left**, the subscriptions to topics T1 and T2. On the **right**, the publishes (in red) and their associated notifies (in blue).

2.1.3. MQTT-SN

In order for MQTT to work correctly, adhering to the three different QoS levels, it is necessary to rely on devices with IP-level connectivity, which is not feasible for all IoT devices. For this reason, in 2007, IBM introduced a new version of MQTT, known as MQTT-SN, which is analogous to and compatible with the original protocol. Its latest version, 1.2, dates back to 2011, and it is not standardized by OASIS. As shown in Figure 3, two novel network elements are introduced:

- *MQTT-SN Gateway*: This is crucial to the protocol's operation and can either be integrated into the Broker or be a standalone device. It serves as an interface between clients and the Broker, translating messages received from one protocol version to another.
- *MQTT-SN Forwarder*: This is useful whenever clients are not on the same Data-link Layer network as the Gateway. The packets in transit are not modified but only encapsulated in an IP packet if received by the client and decapsulated if the opposite occurs.

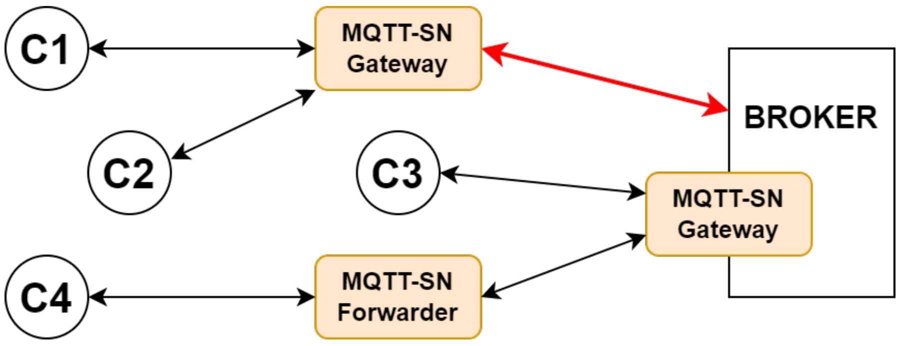

Figure 3. All possible MQTT-SN client configurations.

Clients can communicate by encapsulating messages directly into Data-link Layer frames (e.g., Bluetooth, ZigBee). More compact headers are enabled by various techniques, including translating the strings that define the topics into smaller, fixed-size IDs. When combined with (i) the client's capability to alternate between standby and active states and (ii) the exchanging of appropriate control messages with the MQTT-SN Gateway, this allows them to meet specific energy efficiency requirements.

2.2. IOTA

IOTA is an open-source project of the IOTA Foundation, founded in 2015, that defines a public/permissionless DLT differently from the previous approaches. The Foundation provides a wide variety of official platforms [24] for accessing documentation and enabling user–member interaction. There are three versions of IOTA, each one following this standardization process:

1. Guidelines for the protocol definition are published in a whitepaper.
2. Research specifications and software are defined almost independently on Devnet.
3. The protocol and its complete implementation are released for use on the Mainnet.

Currently, version 1.0 (Original) is considered outdated, version 1.5 (Chrysalis) is used on the Mainnet, and version 2.0 (Coordicide) is still in the design phase. Moreover, in order to access IOTA, a list of hard-coded entry nodes managed by the Foundation is provided. IOTA is also available for use in private/permissioned networks, where the entry node is managed by the administrator.

2.2.1. Tangle

The primary characteristic that distinguishes IOTA from blockchain technologies is its immutable data structure for recording messages. Instead of a chain of blocks, IOTA uses a DAG known as the *Tangle*, where vertices represent individual published messages and edges represent the multiple references they may have to each other. As shown in Figure 4, a message can fall into one of the following four categories:

- *Genesis* (in blue): a message published when the Tangle is created to distribute the system's cryptocurrency for the first time.
- *Referenced message* (in green): a message that has already been published and referenced by others, with its hash included in their headers.
- *Tip* (in red): messages that have been published but not yet referenced by others.
- *New* (in black): a message that has yet to be published and must reference at least two randomly chosen tips.

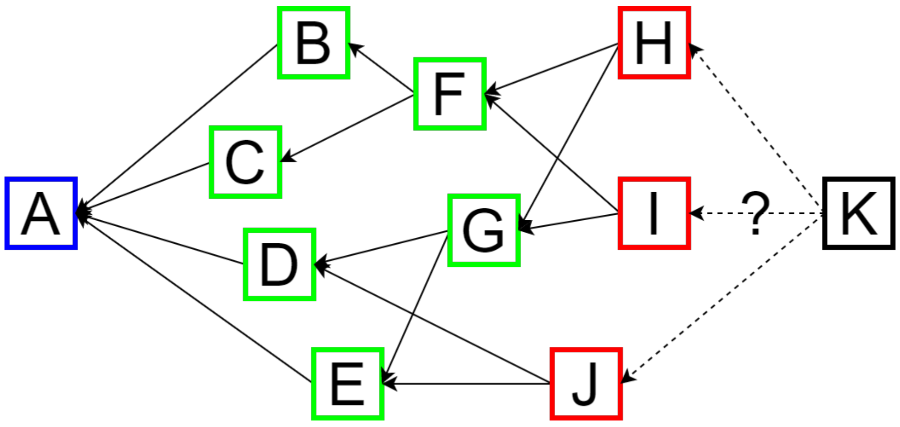

Figure 4. A simplified Tangle representation.

In a blockchain, there is a clear separation between the issuance and publication of a message: a node must wait for a miner to choose, validate, and pass its message through the mining algorithm. In IOTA, this separation does not exist, and all components of the consensus mechanism occur after publication. Therefore, each node can theoretically publish messages on the Tangle independently and at any time, thus avoiding the typical bottleneck of blockchains. The Tangle results in a more scalable data structure, and its performance improves as the number of available tips, and thus active users, increases.

2.2.2. Fee-Less Consensus and Coordinator

To achieve the benefits of scalability, that is, to maintain a large number of constantly available tips in the system, as previously mentioned, nodes are requested to frequently publish a large number of messages. The use of micro-transactions allows them to easily meet this goal, and for this reason, the IOTA Foundation has decided to adopt fee-less consensus algorithms.

In order to provide security to a highly traffic-loaded system without the filtering provided by miners, an effective, lightweight, and high-speed distributed consensus mechanism is required. However, since developing such an algorithm is extremely challenging, the Foundation has, in the meanwhile, adopted a centralized approach while waiting for a complete paradigm transition.

The 1.x versions of the protocol include the *Coordinator*, a special node controlled by the IOTA Foundation that handles various security aspects, including the consensus. The Coordinator enforces a unique view of the network state through the periodic publication of *milestones*, which are messages confirmed by default and which, in turn, confirm all messages that they directly or indirectly reference. However, this solution does not meet the requirements of the proposed system and will be abandoned in favor of decentralized security and consensus mechanisms in version 2.0, as described in the next subsection.

2.2.3. IOTA 2.0

To ensure a truly P2P, secure, and decentralized DLT, the IOTA Foundation has introduced a protocol stack for version 2.0 on top of the TCP/IP stack, as illustrated in Figure 5. The node functionalities are organized into three Layers and are supported by a cross-layer mechanism. This mechanism, known as *Sybil Protection*, aims to prevent malicious actors from gaining control of the system by creating multiple false identities. It is achieved by utilizing a limited resource called *Mana*, which is distributed to nodes through transactions generated by the Value Transfer module. Mana is allocated over time in an exponential manner based on the amount of currency transferred.

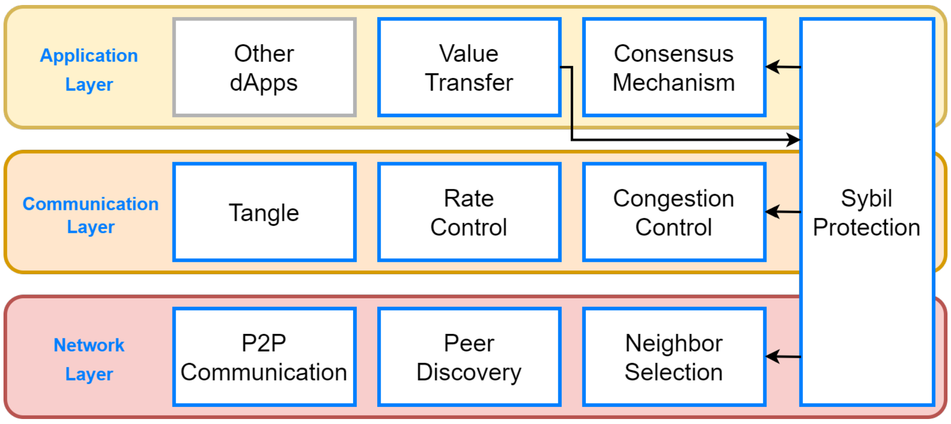

Figure 5. IOTA 2.0 protocol stack.

According to a bottom-up approach, the Layers are as follows:

- Network: This Layer is responsible for handling protocols related to P2P mechanisms, including peer discovery, neighbor selection, and message exchange using a gossip protocol.
- Communication: In this Layer, protocols are implemented to control the emission rate of messages, to prevent congestion, and to support Tangle-related mechanisms that ensure a shared, consistent, and immutable view of the registered data.
- Application: This encompasses mandatory applications for currency transfer and consensus. Additionally, nodes can host dApps that are installed by their operators. One example of such a dApp is Streams, which will be discussed in more detail in Section 2.2.4.

Without delving into specifics, Mana is used in three modules of the overall system, along with other mechanisms based on randomness: (i) the neighbor selection module employs it to prevent the eclipsing of a node by controlling the majority of its neighbors, (ii) the congestion control module uses it to mitigate Distributed Denial of Service (DDoS) attacks caused by message flooding, and, lastly, (iii) the consensus mechanism leverages it to safeguard against potential manipulations.

The IOTA Foundation outlines two possibilities for consensus making, both based on a distributed voting mechanism. The approach described in the research specifications [25] requires random nodes to exchange opinions until they reach a consensus, while the method used in the software prototype on Devnet [26] involves the publishing of messages that select tips belonging to a particular branch of the Tangle to express support in case of a conflict.

2.2.4. Channels

Streams is a framework developed by the IOTA Foundation that offers cryptographic tools for creating secure communication protocols based on the required Transport Layer technologies. The alpha version was released in 2020 [27], and since then, there have been no significant updates. Thanks to the tools offered by Streams, the Foundation was able to create a protocol called Channels. This protocol utilizes a publish/subscribe model and puts information on the Tangle. As of now, Channels is compatible with IOTA 1.5 [28], but it has not yet been released on the Mainnet. The protocol organizes messages into tree-like structures called *channels*, and clients can assume different roles:

- *Author*: responsible for creating the channel, defining its structure, and managing read and/or write access for each branch.
- *Subscriber*: any user who has access to at least one branch of a channel created by someone else; if they have write access, subscribers can also act as publishers for a branch.

When creating a branch, the author requests the authorized publishers to create a key pair and share the public key. Publishers can then use their private key to encrypt and sign their packets, and only those from authorized sources are accepted by the protocol.

When a client subscribes to a branch, the author provides it with a list of public keys to decrypt the packets. However, if a subscriber is removed from the branch, due to voluntary unsubscription or a ban, it can create a security risk. In this case, the author must create a new branch intended for the remaining authorized subscribers only while informing them of the changes.

Every branch consists of at least one chain of consecutive packets, with distinct publishers and subscribers, creating a finer level of detail when determining access. To establish the order of packets issued by each publisher, a sequencing branch is utilized, with one chain assigned to each publisher, including the author. Each of these chains contains references to all the packets issued by its publisher in any branch in a chronological order. Channels employs the message types specified in Table 2 in order to create this structure, as depicted in Figure 6.

Table 2. Types of packets used to build a tree-like structured channel.

`Announce`
Published by the channel's author upon its creation, this message signals the start of the channel and includes its identifier and other parameters.
`Keyload`
This message is published by the author of the channel, and it is used to create a new branch; for this purpose, it contains a list of its chains with their subscribers and publishers.
`SignedPacket`
Sent by channel publishers, these messages contain the actual data that have to be signed and may be encrypted.
`Sequence`
This message is added to the sequencing branch whenever a data packet is published. It contains a reference to the packet indicating its branch, its chain, and its position within it.

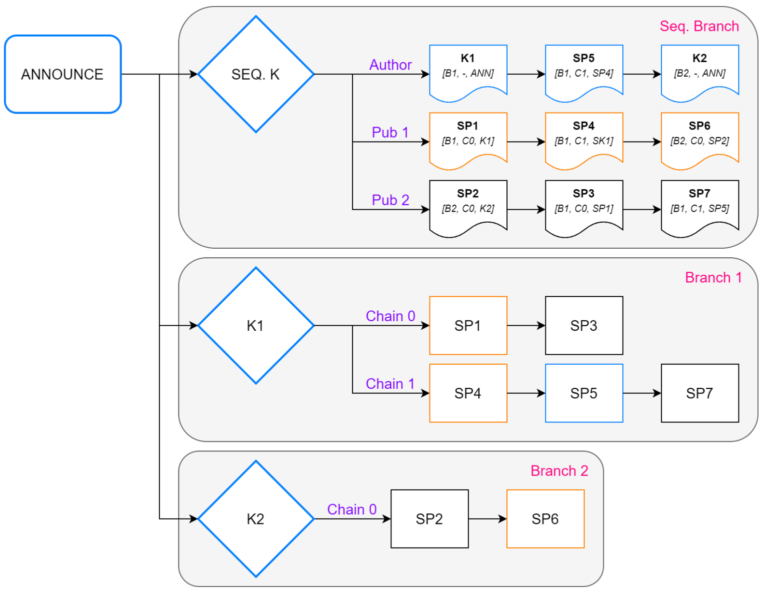

Figure 6. Example of a basic channel: blue represents the author's packets, while orange and black indicate the packets of two different publishers.

3. Proposed System Characterization

The proposed system aims at facilitating M2M communication patterns among all its constituent elements using the MQTT protocol. This enables the automation of the functionalities of a typical IoT ecosystem, comprising sensors and actuators, along with secure and decentralized storage via IOTA. By leveraging the Channels protocol, the system allows its administrator to organize messages into channels with finely grained access controls tailored to the specific services offered by the provided IoT-based application.

3.1. Architecture

The general architecture of the proposed system, as shown in Figure 7, supports the deployment of multiple geographically separated IoT sites, each comprising two functional areas:

- *IoT Area*: This area encompasses various IoT devices, such as sensors and actuators that generate and exchange data. It also includes a fog computing module to process data before publishing them to interested devices and on the Tangle.
- *DLT Area*: This area consists of a node running IOTA software, responsible for publishing data on the Tangle and processing it via Channels. Additionally, it implements a recovery mechanism in case that node experiences temporary unavailability.

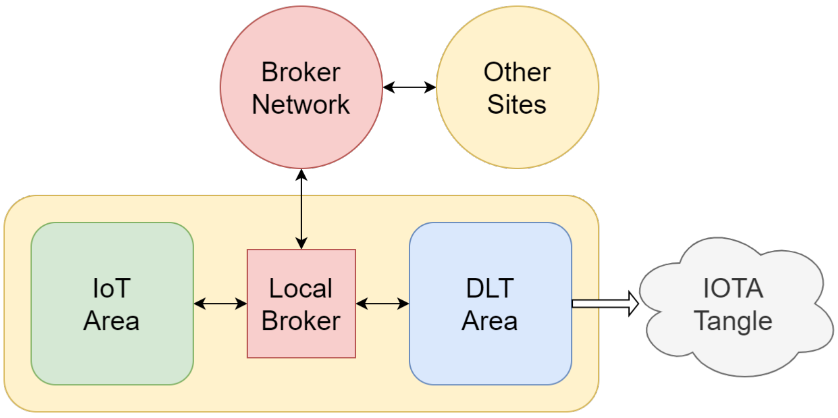

Figure 7. Modular structure of single-site system.

Every site is linked to the rest of the system via a hierarchical and distributed Broker network that effectively routes MQTT messages. This is accomplished by introducing a Local Broker in each site; the Local Broker is responsible for managing both the internal distribution of messages and communication with the transport network and, consequently, the entire system. For the sake of presentation simplicity, in the following subsections, the collection of Local Brokers and the Broker network are represented as a single unit.

3.1.1. IoT Area

Figure 8 gives insight into the IoT Area by depicting its constituent elements, including:

- *IoT devices*: Heterogeneous sensors that gather measurements and actuators that respond accordingly. These devices are typically battery-powered and use Data-link Layer protocols, making it necessary to use MQTT-SN.
- *MQTT-SN Gateway*: An essential component to enable IoT devices to communicate via MQTT-SN.
- *Fog Processing Unit* (Fog PU): Responsible for processing and organizing data from IoT devices in a manner that can be used by actuators at any site in the system. Additionally, it provides data to the DLT Area to be further published on the Tangle.

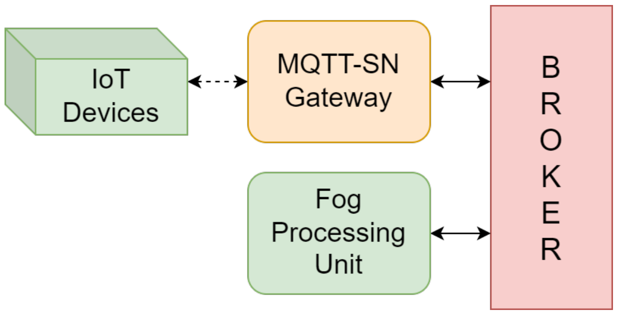

Figure 8. Constitutive elements of an IoT Area: the MQTT-based communication interface is shown with an arrow, while the MQTT-SN interface is shown with a dashed arrow.

3.1.2. DLT Area

In Figure 9, the elements of the DLT Area are depicted, including:

- *IOTA Node*: This element represents the unique interface with the IOTA P2P network, responsible for receiving processed data from the local site, encapsulating them in Channels packets, and publishing them as messages on the Tangle. It also employs MQTT to signal the IOTA Cache, ensuring its proper functioning.
- *IOTA Cache*: This element stores the local processed data until receiving a notification of successful publication from the IOTA Node. This function is particularly valuable in case of the momentary unavailability of the IOTA Node, where the IOTA Cache forwards all pending data to it as soon as it reconnects.

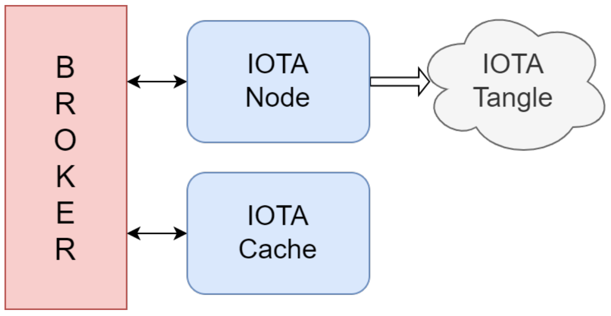

Figure 9. Constitutive elements of a DLT Area: the MQTT-based communication interface is shown with an arrow, while the IOTA interface is shown with a thick arrow.

3.2. Communication Protocol

Every IoT device is associated with the site where it is located, and it is uniquely identified within that site. By means of a `[site, ID]` pair, each device can publish data on a topic that follows the format:

$$\texttt{root/device/site/ID/type}$$

In addition, it is convenient to extend the subtopic `type` to differentiate between sensor measurements and actuator actions:

$$\texttt{type} = \begin{cases} \texttt{sense/dataType} \\ \texttt{act/dataType} \end{cases}$$

Without loss of generality, we further assume that a generic IoT device publishes its data on the topic:

$$\texttt{root/device/x} \qquad \text{where} \qquad \texttt{x} = \texttt{site/ID/type}$$

All data published by IoT devices are processed by the Fog PU located at the same site as the devices, which, in turn, republishes the data on the following topic for any interested device and for the DLT Area to access:

`root/fog/x`

After proper processing, data are received by both the IOTA Node and the IOTA Cache. The IOTA Node encapsulates the data into Channels packets, which are subsequently published on the Tangle. Meanwhile, the IOTA Cache temporarily stores the data in memory until it receives a notification from the IOTA Node on the following topic:

`root/node/site/tangle`

The notification carries a list of `[topic, timestamp]` pairs to identify data that are going to be deleted by the Cache, where the timestamp is set by the Fog PU in the previous step. This mechanism is in place to account for the possibility that the IOTA Node could be unreachable and, thus, not able to receive data from its own site. If this occurs, the IOTA Cache temporarily stores the data until the IOTA Node reconnects. Upon reconnection, the IOTA Node forwards a notification on the following topic:

`root/node/site/active`

Upon receiving the notification, the IOTA Cache sends all pending data, along with the corresponding `[topic, timestamp]` pair, through a limited number of messages to the IOTA Node on the following topic:

`root/cache/site/pending`

All possible topic formats are displayed in Table 3. Therefore, if we consider N_x as the number of subtopics, which we earlier named "`x`", utilized in a given network and N_s as the number of sites, the number of topics in the system N_T is equal to:

$$N_T = 2N_x + 3N_s$$

Table 3. All possible MQTT topic formats, their publishers, and their subscribers.

Topic	Publisher	Subscriber
`root/device/x`	IoT device	Fog PU
`root/fog/x`	Fog PU	IoT device(s), IOTA Node, IOTA Cache
`root/node/site/tangle`	IOTA Node	IOTA Cache
`root/node/site/active`	IOTA Node	IOTA Cache
`root/cache/site/pending`	IOTA Cache	IOTA Node

3.3. Multi-Tier Broker

In order to design a hierarchical Broker network, the `site` subtopic can be developed on multiple levels, as introduced in [29]. This involves connecting a Central Broker with a series of Aggregation Brokers that are linked to the Local Brokers deployed in each site. The required functions to enable this mechanism can be described as follows:

- The `site` subtopic is developed as:

$$\texttt{site} = \texttt{aggregation/local}$$

- If a Broker receives a subscription request for a topic related to a different site, it accepts the request and then acts as a client by sending the same subscription request to the Broker at the higher level.
- Each Broker saves the publish messages it receives from the lower level and, acting as a client, forwards them to the higher level's Broker.

The example shown in Figure 10 depicts the scenario where a device is interested in data from a site belonging to a different aggregation group. As explained in Section 3.2, devices subscribe to topics associated with the data processed by the Fog PU.

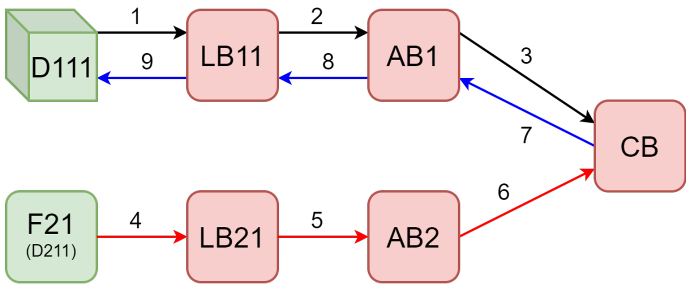

Figure 10. Example of MQTT message exchanges within three hierarchical levels of Brokers.

Assume that the topic of interest is denoted by:

$$T = \texttt{root/fog/agg2/loc1/ID1/type1}$$

The message exchange process can be summarized as follows:

1. Device D_{111} subscribes to topic T via Local Broker LB_{11}.
2. Local Broker LB_{11} subscribes to topic T via Aggregation Broker AB_1, since the site `agg2/loc1` does not match its own.
3. Aggregation Broker AB_1 subscribes to topic T via the Central Broker CB, since the group `agg2` does not match its own.
4. Fog PU F_{21} publishes the processed data of device D_{211} on topic T via Local Broker LB_{21}.
5. Local Broker LB_{21} forwards the publish message to Aggregation Broker AB_2 as a client.
6. Aggregation Broker AB_2 in turn forwards the publish message to the Central Broker CB as a client.
7. The Central Broker CB notifies Aggregation Broker AB_1 subscribed to T.
8. Aggregation Broker AB_1 forwards the notify message to Local Broker LB_{11} subscribed to T.
9. Local Broker LB_{11} forwards the notify message to device D_{111} subscribed to T.

3.4. Overall Communication Protocol Design

With the aim to describe the proposed overall communication protocol, we take into account a sufficiently general use case, where all the IoT devices are located at the same site `z1`, which is equipped with an MQTT-SN Gateway, a Fog PU, an IOTA Node, and an IOTA Cache, as defined in the system architecture. For this use case, we consider two devices employed to cover all the scenarios:

- *Sensor*: Identified by the pair `[z1, d1]`, it periodically publishes its measurements, which are of type `t1`. For clarity and brevity, we assume:

$$\texttt{s1} = \texttt{z1/d1/sense/t1}$$

- *Actuator*: Identified by the pair `[z1, d2]`, it acts asynchronously, and the output of its actions is of type `t2`. Again, we assume:

$$\texttt{a2} = \texttt{z1/d2/act/t2}$$

The number of topics in the system, as shown in Table 4, is $N_T = 2 \times 2 + 1 \times 3 = 7$. Finally, the sensor is not interested in the output provided by the actuator.

Table 4. MQTT topics considered in the investigated use case.

Topic	Publisher	Subscriber
`root/device/s1`	Sensor	Fog PU
`root/device/a2`	Actuator	Fog PU
`root/fog/s1`	Fog PU	Actuator, IOTA Node, IOTA Cache
`root/fog/a2`	Fog PU	IOTA Node, IOTA Cache
`root/node/z1/tangle`	IOTA Node	IOTA Cache
`root/node/z1/active`	IOTA Node	IOTA Cache
`root/cache/z1/pending`	IOTA Cache	IOTA Node

3.4.1. Sensed Data Publishing Phase

First of all, we present the message exchange occurring when the sensor publishes its measurements by means of the sequence diagram depicted in Figure 11.

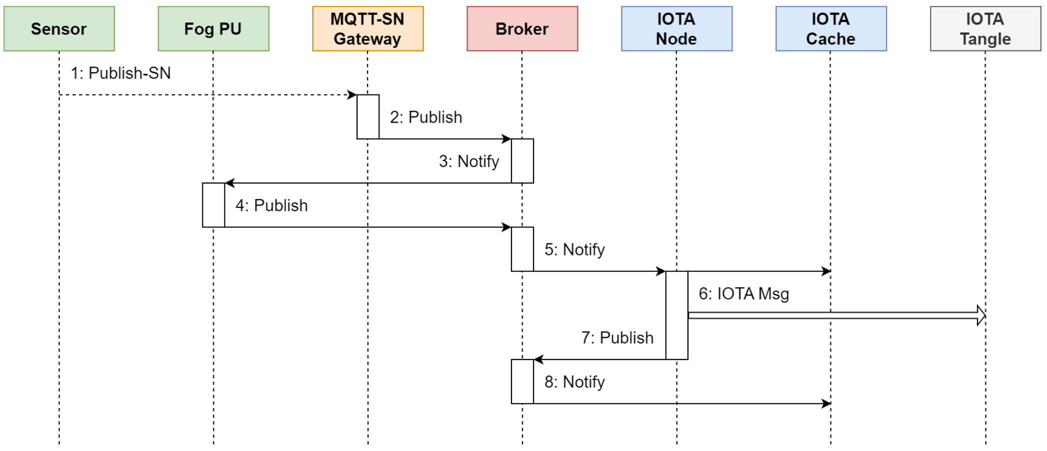

Figure 11. Message exchange occurring when a sensor is publishing (periodic) data.

The involved steps are as follows:

1. The sensor transmits the data to be published on `root/device/s1` to the MQTT-SN Gateway using its Data-link Layer communication protocol.
2. The MQTT-SN Gateway acts on behalf of the sensor and sends data to the Broker on `root/device/s1`.
3. The Broker forwards data to the Fog PU.
4. After processing data published by the sensor, the Fog PU sends the processed data to the Broker on `root/fog/s1`.
5. The Broker forwards the processed data to both the IOTA Node and the IOTA Cache.
6. The IOTA Node encapsulates the processed data in a Channels packet and publishes it on the Tangle.
7. The IOTA Node sends a notification of the publication on `root/node/z1/tangle`.
8. The Broker forwards the notification to the IOTA Cache, which in turn proceeds to delete the previously stored processed data.

3.4.2. Actuator Response Phase

The sequence diagram in Figure 12 depicts the messages exchanged when the actuator performs an action in response to the received sensed data.

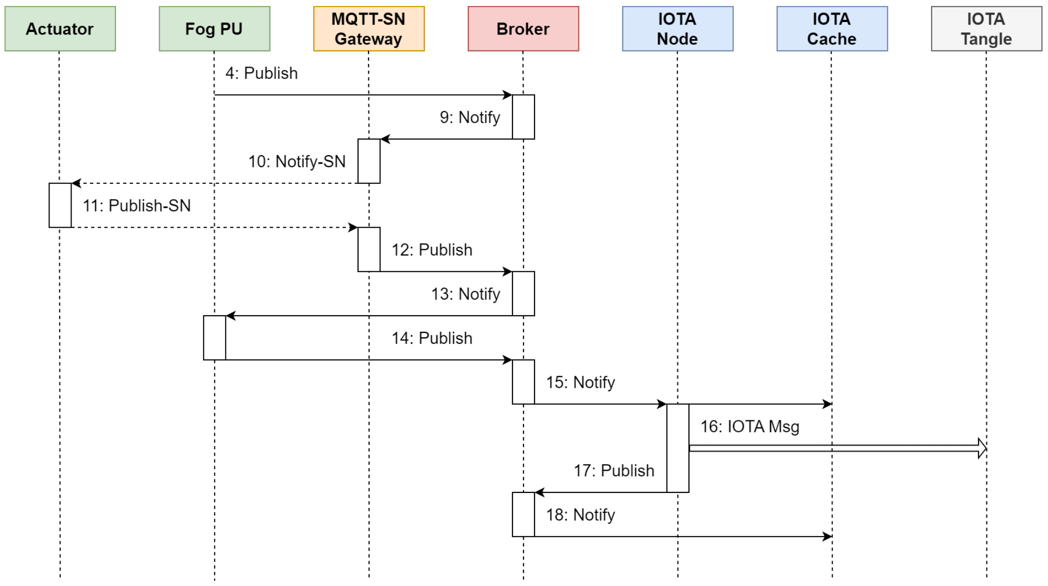

Figure 12. Message exchange when an actuator receives sensed data and responds accordingly.

The sequence of steps is the following:

4. As previously stated, the Fog PU forwards the processed sensor data to the Broker on `root/fog/s1`.
9. The same processed data carried by message 5 are also forwarded by the Broker to the actuator, but the MQTT-SN Gateway intercepts these data.
10. The MQTT-SN Gateway transmits the processed data to the actuator on `root/fog/s1` using its Data-link Layer communication protocol.

Messages 11–18 have the same structure as messages 1–8 seen earlier for the sensor, with the only difference being that the subtopic `s1` is replaced by `a2`. Once the actuator completes its task triggered by the last received measurements from the sensor, it publishes the output on the topic `root/device/a2`.

3.4.3. IOTA Node Unavailability Management Phase

Figure 13 depicts the sequence of messages exchanged when the IOTA Node is temporarily disconnected. Messages 4–8 for sensors and 14–18 for actuators are replaced. The subtopic `d0` is used to refer to either s1 or a2 in this scenario.

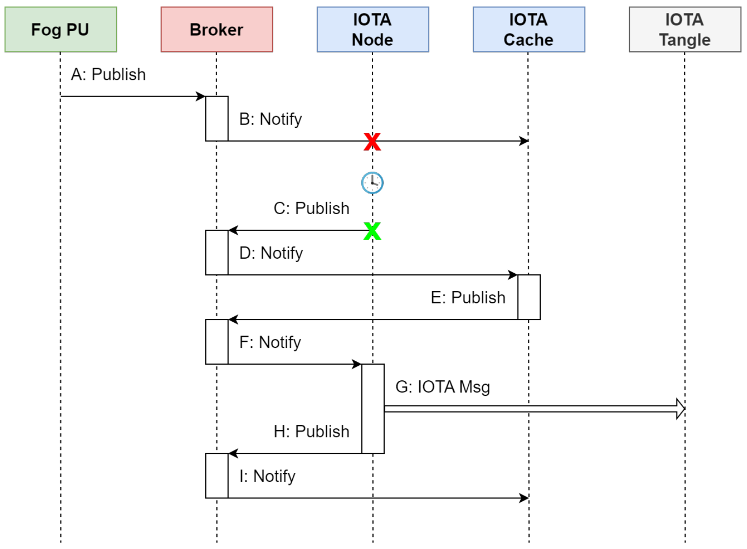

Figure 13. Message exchange to manage the temporary unavailability of the IOTA Node.

A. During the disconnection, the Fog PU sends multiple processed data to the Broker on the respective `root/fog/d0` topics.
B. The Broker forwards the processed data to both the IOTA Node and the IOTA Cache, but the former does not receive them due to the disconnection that occurred.
C. Once the IOTA Node reconnects, it sends a notification on `root/node/z1/active` to the Broker.
D. The Broker forwards the notification to the IOTA Cache.
E. The IOTA Cache sends the pending processed data on `root/cache/z1/pending` to the Broker.
F. The Broker forwards the pending processed data to the IOTA Node.
G. The IOTA Node encapsulates each entry in the pending processed data message into Channels packets, which are then published on the Tangle.
H. After successfully publishing the Channels packets, the IOTA Node sends a notification on `root/node/z1/tangle`.
I. The Broker forwards the notification to the IOTA Cache, which in turn deletes the processed data associated with the Channels packets.

4. Performance Evaluation

In the following, a performance analysis of the proposed framework is performed in terms of the most relevant metrics in order point out the possible achievable advantages. Among the features presented and discussed in [21], we limited our investigation to the most relevant ones, i.e., latency and throughput, while additional high-level aspects like scalability, security, availability, interoperability, and mobility support are qualitatively evaluated.

It is worth noticing that, actually, a high-fidelity system emulation is not completely possible, since the centralized consensus and security mechanisms in IOTA 1.5 are still used in practice. According to the IOTA Foundation, the release of version 2.0 seems to be imminent, although a specific timeframe has not yet been announced [30]. Another concern is that Streams, and thus Channels, is still in its alpha version, and to date, there have not been relevant updates on its development status.

4.1. System-Level Simulation

In order to model and simulate the network elements involved in the proposed system, we selected OMNeT++ version 6.0.1 [31] as the simulation framework. Indeed, OMNeT++ is a C++-based discrete-event simulator that uses the process-interaction approach. An OMNeT++ model consists of modules communicating by message passing. Modules can be arbitrarily nested. The model topology is specified by a topology description language that supports the separation of the interface and functionality and facilitates model reuse. Moreover, the use of the tracing/debugging capabilities does not require additional code [32]. Moreover, this open-source platform offers a comprehensive collection of built-in libraries that define several network protocols. Additionally, it provides great programming flexibility through the use of the C++ language, allowing for the introduction of novel messages, interfaces, and protocols.

Neither the MQTT protocol nor the operation of the IOTA network is included in the native libraries of the software. Therefore, we needed to develop and integrate these components. Given the framework's agnosticism toward lower-level technologies and protocols, our emphasis was on high-level characteristics. The behavior of MQTT was defined using the pseudo-primitives outlined in Section 2.1.2, while the IOTA network involved was simulated by utilizing the TangleSim [33] library available on GitHub.

4.1.1. Scenario Characterization

Considering the modularity of the proposed system, we performed a single-site analysis while neglecting the hierarchical Broker network, as it is reasonable to assume that it represents a critical aspect for the overall performance.

To ensure generality and address the worst-case scenario, the specific nature of IoT devices (whether sensors, actuators, or both) is not relevant. Therefore, each device was simulated as a periodic traffic source, referred to as an *IoT source*.

Each packet emitted by an IoT source is empty and consists of an MQTT pseudo-header. The pseudo-header includes the following information: (i) an integer variable specifying the MQTT pseudo-primitive (refer to Table 1), with `notify` and `publish` being the only ones used by the Broker and all other devices, respectively, (ii) a URI string defining the MQTT topic, and (iii) the publication time at the IoT source represented by a `time_t` variable.

This implies that the messages sent by the IoT sources in the simulated scenario do not adhere to the MQTT-SN protocol. Consequently, the MQTT-SN Gateway serves the purpose of receiving these messages and retransmitting them with a certain delay, simulating the processing time that would typically take place. However, this does not have a significant impact on the simulated scenario, which primarily focuses on the impact of the Broker.

Furthermore, since the evaluation of parameters such as latency and throughput in steady-state operation conditions does not depend on IOTA Cache, we omitted this module. Lastly, all MQTT subscriptions were defined before the simulation campaign. As a result, the simulated network functional blocks and the path of each packet correspond to the configuration depicted in Figure 14.

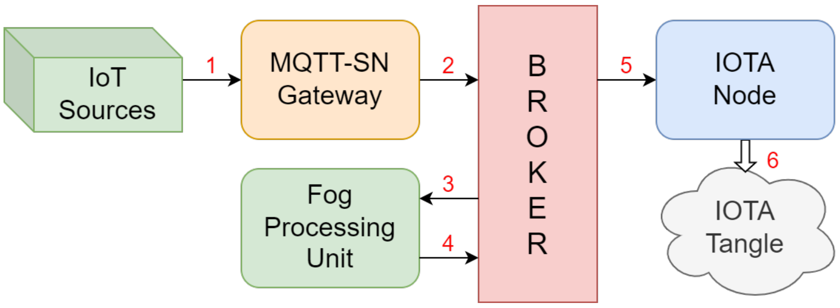

Figure 14. Functional blocks and packet path in the simulated network.

In order to improve the performance, as explained in the following, we implemented the Local Broker as a set of load-balanced devices with a weighted fair queueing (WFQ) scheduling discipline.

For the sake of generality, we did not refer to specific lower-level networking technologies and protocols, but we modeled this aspect by introducing a set of parameters and specific reasonable values ranges, as detailed in Table 5.

Firstly, two rounds of simulations were performed, each lasting 7200 s. In the first round, the IoT source publishing period was assumed to be equal to 1 s, while, in the second one, this value was reduced to 100 ms to focus on the worst-case scenario.

Table 5. Parameters adopted in simulation campaign.

Name	Value(s)	Meaning
`channelDelay`	0.5 ms	Time necessary to transmit a message between a couple of nodes over a link: it includes the channel delay and the delay introduced by a specific Data-link Layer technology.
`iotaNode.powTime`	100 ms	Time required by the IOTA Node to perform the lightweight Proof of Work (PoW).
`* .elaborationDelay`	2 ms	Time required by each node to process a message and perform the necessary actions upon its reception.
`numSources`	$[1, 2, \ldots, 100]$	Number of IoT sources.
`numBrokers`	$[1, 2, 4]$	Number of load-balanced Brokers.
`brokers[*].serviceTime`	[5 ms, 6 ms, 7 ms]	Message service time for each Broker FIFO queue.

4.1.2. End-to-End Publishing Latency and Throughput

The results presented in this subsection were obtained by conducting multiple simulations, each with (i) a different number of IoT sources, (ii) a different number of load-balanced Brokers, and (iii) different service times of their queues, as pointed out in the last three rows of Table 5. The outcomes were analyzed as a function of the number of IoT sources.

First, we discuss the E2E Average Publishing Latency, which is the average time elapsed from the message transmission by an IoT source to its publishing on the Tangle by the IOTA Node.

In Figure 15, the latency performance, plotted on a logarithmic scale, is investigated for a publishing period equal to 1 s. It is evident that the initial linear growth is followed by a sudden queue saturation, which causes the system to become unstable. For example, if only one Broker is utilized with service times of 6 ms and 7 ms, the latency experiences an exponential increase, rapidly approaching infinity. This congestion-related drawback can be mitigated by possibly adopting shorter service times and/or an increased number of Brokers. However, it is important to note that, in practice, the message service time cannot be varied, while the converse is true for the number of Brokers and their balancing scheme, as they are usually virtualized processes. It is worth noticing that this approach is also able to enhance the system scalability.

We focused on the worst-case scenario by further considering a publishing period of 0.1 s, as shown in Figure 16. In this case, messages enter the Broker(s) queues at a higher frequency, leading to earlier congestion. Therefore, when addressing a real system design, it is essential to consider the overall publishing rate and to proportionally and dynamically adjust the number of Brokers and their load-balancing scheme.

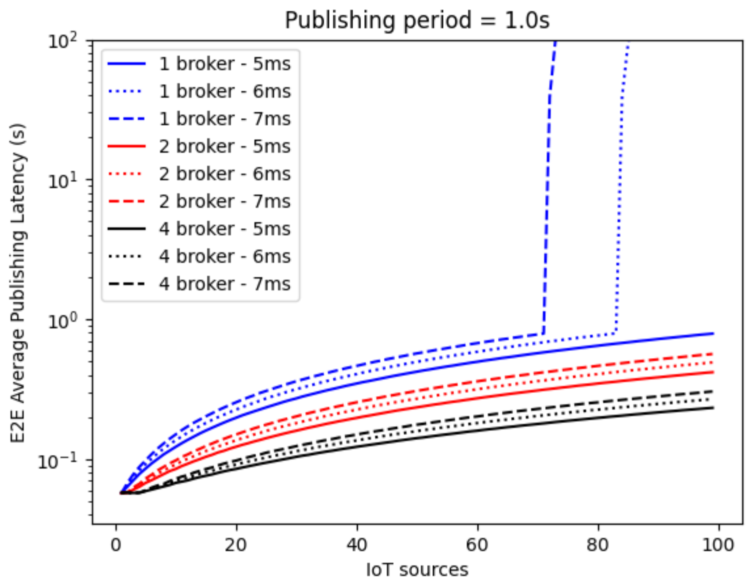

Figure 15. E2E Average Publishing Latency with a publishing period of 1 s.

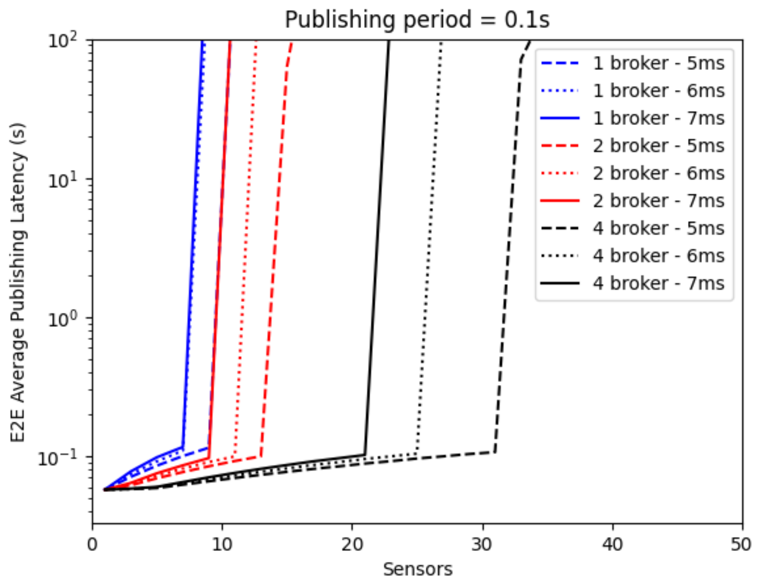

Figure 16. E2E Average Publishing Latency with a publishing period of 0.1 s.

To conclude the performance analysis, the On Tangle Throughput, which represents the average number of messages eventually published by the IOTA Node per second, is depicted in Figure 17 for a publishing period of 1 s. It can be pointed out that, in the absence of bottlenecks caused by Brokers, the throughput directly corresponds to the number of messages generated per second by all IoT sources. Moreover, a couple of curves (in blue) correspond to the worst-case situation, i.e., one Broker with a higher service time (as already identified in Figure 15), where the throughput decreases due to the congestion effect. Once again, the load-balanced solution provides a remarkable advantage in terms of achievable performance.

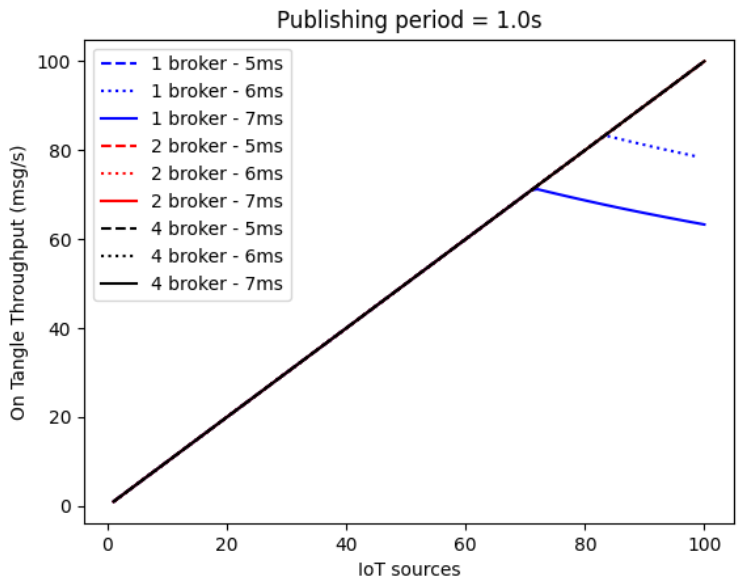

Figure 17. On Tangle Throughput with a publishing period of 1 s.

As a final consideration, it is important to note that the Fog PU and the IOTA Node can also potentially become bottlenecks if they are unable to handle the traffic generated by the IoT sources. Hence, a load-balancing architecture can be applied to these network elements. However, the message arrival rate and, consequently, the processing rate for these devices are typically lower than those for the Brokers. As a result, their design requirements are expected to be less demanding, and for this reason, we adopted a conservative approach in the proposed system.

4.2. Qualitative Evaluation

In the following subsections, aspects like scalability, security, availability, interoperability, and mobility support are qualitatively evaluated in order point out the possible achievable advantages and future developments.

4.2.1. Scalability

The purpose of the fog computing paradigm is to simplify the central network components and shift the complexity to peripheral ones. When adopted in a modular system like the proposed one, where a hierarchical network of Brokers represents the central component, it offers significant scalability advantages. Additionally, each site independently manages data storage within the IOTA Tangle, eliminating the traffic flows that would arise with a centralized database.

Moreover, we use the public Mainnet because of its higher efficiency and reliability, particularly in light of the growing number of network users and activity, in contrast to a private solution.

4.2.2. Security

IOTA, like most DLTs, depends on cryptography-based security mechanisms to ensure the integrity and confidentiality of its data. In version 2.0, these mechanisms will be integrated into a module of the protocol stack called *Sybil Protection*, which uses a reputation-based mechanism implemented through a tokenized resource that is difficult to obtain and stack. This resource regulates the selection of neighboring nodes in the P2P network, as well as the congestion control and consensus mechanisms, preventing attacks such as eclipsing, flooding, or consensus manipulation.

The data in the proposed system are publicly stored on the Mainnet, making them traceable, but their confidentiality is maintained through the access control provided by the encryption of the Channels protocol.

The system's security can be further enhanced by taking measures to secure the connections relying on the IP Layer functions in addition to the mechanisms provided by the Data-link Layer communication protocols. In addition, it is worth noticing that MQTT supports the secure version of both TCP, called Transport Layer Security (TLS), and UDP, called Datagram Transport Layer Security (DTLS).

Another countermeasure that could be taken is to use firewalls and Intrusion Detection and Protection Systems (IDPSs) to secure the connections of individual sites to the outer networks occurring through the Local Broker and the IOTA Node.

4.2.3. Availability

The proposed framework adopts IOTA technology to ensure that data are always retrievable once confirmed on the Tangle, which in turn is securely and immutably stored in almost all participating nodes belonging to the P2P network. For this purpose, the use of the public Mainnet ensures that confirmations are relatively quick, making the data readily available.

The only exceptions to this would be (i) in the case of Distributed Denial of Service (DDoS) attacks, which are nearly impossible to achieve on the public Mainnet due to its large number of nodes, or (ii) in the case of eclipsing attacks that aim to control all the P2P neighbors of an IOTA Node, with the intent of blocking all of its message publications.

4.2.4. Interoperability

Generally speaking, WoT-oriented communication protocols are specifically designed to bridge the gap between IoT platforms and application domains that are fundamentally incompatible. By selecting MQTT and MQTT-SN, all devices within the proposed system can independently contribute to the network functions and provided services, regardless of their connectivity and location. This approach also enables the connection of IoT and DLT functional areas, effectively linking their respective domains.

4.2.5. Mobility Support

There are several additional use cases related to the domain of mobile IoT, where the wireless interface could involve 5G cellular technology. In this case, the devices would publish and receive MQTT-SN messages via gNodeBs, which connect them to other elements at a site level through an MQTT-SN Gateway. In addition, the networking, processing, and storage resources available in a zone can be virtualized and properly managed with Software-Defined Networking (SDN) and Network Function Virtualization (NFV) approaches [34]; for instance, in order to ensure the necessary speed and latency requirements, a dedicated network slice must be allocated to the offered service [35,36].

5. Conclusions

This paper deals with the design of an intelligent IoT environment by focusing on the context awareness achieved by a distributed event storage system. For this purpose, the primary challenges facing autonomous IoT networks, which include the necessity for lightweight communication technologies for M2M traffic and the absence of trust in third-party companies providing cloud-based data storage services, are first outlined.

In addition, a brief overview of the current approaches proposed in the WoT paradigm is provided, while blockchain-based alternatives are dismissed due to the limitations pointed out. Based on these considerations, we adopted MQTT and MQTT-SN protocols for communication and IOTA as an immutable distributed ledger.

We further characterized a novel integrated framework, describing its architectural components, interfaces, and communication protocol. Additionally, we highlighted a wide range of potential use cases where the utilization of IOTA Channels enables the on-demand development of applications by providing appropriately categorized data generated by the IoT domain itself.

Finally, we provide both (i) a numerical simulation and (ii) a qualitative-by-design performance analysis of the proposed framework in terms of the most relevant metrics, which are latency, throughput, scalability, security, availability, interoperability, and mobility support, in order point out the achievable advantages and future developments.

Though the performance achieved by the proposed approach pointed out its capability of supporting a DLTaaS distributed platform, an open issue is still represented by the management and control of the Broker subnetworks, so it could be beneficial in a future investigation to focus on the joint application of NFV and SDN approaches in order to properly address this aspect.

Possible further developments of the proposed approach could be focused on (i) a system redesign in light of the next expected release of the IOTA standard in order to overcome the current limitations or (ii) the investigation of 5G/6G technologies' impact by means of the proper management of virtualized communications/storage/computing resources.

Author Contributions: Conceptualization, F.C. and G.G.; methodology, F.C. and G.G.; software, G.G.; validation, F.C. and G.G.; formal analysis, F.C. and G.G.; investigation, F.C. and G.G.; resources, F.C.; writing—original draft preparation, G.G.; writing—review and editing, F.C. and G.G.; supervision, F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Project FABuLOuS—Florence Area Blockchain Laboratory Observatory Studies, Bando "Ricercatori a Firenze", Fondazione Cassa di Risparmio di Firenze (2021).

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to express their gratitude to Alessio Chen for his support and fruitful discussion.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Evans, D. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*; CISCO White paper; CISCO: San Jose, CA, USA, 2011.
2. Higginbotham, S.; Pesce, M. Internet of Everything: Macro & Micro. *IEEE Spectr.* **2021**, *58*, 22–23.
3. Web of Things (WoT) Architecture 1.1. Available online: https://www.w3.org/TR/wot-architecture (accessed on 17 May 2023).
4. Heuer, J.; Hund, J.; Pfaff, O. Toward the Web of Things: Applying Web Technologies to the Physical World. *Computer* **2015**, *48*, 34–42. [CrossRef]
5. HTTP Documentation. Available online: https://httpwg.org/specs (accessed on 17 May 2023).
6. The Constrained Application Protocol (CoAP). Available online: https://datatracker.ietf.org/doc/html/rfc7252 (accessed on 17 May 2023).
7. Bormann, C.; Castellani, A.P.; Shelby, Z. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Comput.* **2012**, *16*, 62–67. [CrossRef]
8. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017; pp. 1–7.
9. Vinoski, S. Advanced Message Queuing Protocol. *IEEE Internet Comput.* **2006**, *10*, 87–89. [CrossRef]
10. The Simple Text Oriented Messaging Protocol. Available online: http://stomp.github.io/stomp-specification-1.2.html (accessed on 8 July 2023).
11. MQTT Specification. Available online: https://mqtt.org/mqtt-specification (accessed on 17 May 2023).
12. Gomez, C.; Arcia-Moret, A.; Crowcroft, J. TCP in the Internet of Things: From Ostracism to Prominence. *IEEE Internet Comput.* **2018**, *22*, 29–41. [CrossRef]
13. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
14. Uy, N.Q.; Nam, V.H. A comparison of AMQP and MQTT protocols for Internet of Things. In Proceedings of the 6th NAFOSTED Conference on Information and Computer Science (NICS), Hanoi, Vietnam, 12–13 December 2019; pp. 292–297.
15. Sadawi, A.A.; Hassan, M.S.; Ndiaye, M. A Survey on the Integration of Blockchain With IoT to Enhance Performance and Eliminate Challenges. *IEEE Access* **2021**, *9*, 54478–54497. [CrossRef]
16. Shammar, E.A.; Zahary, A.T.; Al-Shargabi, A.A. A Survey of IoT and Blockchain Integration: Security Perspective. *IEEE Access* **2021**, *9*, 156114–156150. [CrossRef]

17. Ullah, Z.; Raza, B.; Shah, H.; Khan, S.; Waheed, A. Towards Blockchain-Based Secure Storage and Trusted Data Sharing Scheme for IoT Environment. *IEEE Access* **2022**, *9*, 36978–36994. [CrossRef]
18. Nakamoto, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*; Bitcoin Whitepaper; United States Sentencing Commission: Washington, DC, USA, 2008.
19. Alrubei, S.; Ball, E.; Rigelsford, J. Securing IoT-Blockchain Applications through Honesty-Based Distributed Proof of Authority Consensus Algorithm. In Proceedings of the 2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), Online Event, 17 June 2021.
20. Bonadio, A.; Chiti, F.; Fantacci, R.; Vespri, V. An integrated framework for blockchain inspired fog communications and computing in internet of vehicles. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *11*, 755–762. [CrossRef]
21. Ferrag, M.A.; Shu, L. The Performance Evaluation of Blockchain-Based Security and Privacy Systems for the Internet of Things: A Tutorial. *IEEE Internet Things J.* **2021**, *8*, 17236–17260. [CrossRef]
22. Number of Blockchain Wallet Users 2022/2023: Breakdowns, Timelines, and Predictions. Available online: https://www.financesonline.com/number-of-blockchain-wallet-users (accessed on 17 May 2023).
23. Hello Future | Hedera. Available online: https://hedera.com (accessed on 17 May 2023).
24. Home | IOTA. Available online: https://www.iota.org (accessed on 17 May 2023).
25. IOTA 2.0 Research Specifications. Available online: https://wiki.iota.org/IOTA-2.0-Research-Specifications/Preface (accessed on 17 May 2023).
26. GoShimmer Protocol Specification. Available online: https://wiki.iota.org/goshimmer/protocol_specification/components/overview (accessed on 17 May 2023).
27. Final Alpha Release for IOTA Streams. Available online: https://blog.iota.org/final-alpha-release-for-iota-streams-5a4cfeca506c (accessed on 17 May 2023).
28. Streams Software Documentation. Available online: https://wiki.iota.org/streams/welcome (accessed on 17 May 2023).
29. Kurdi, H.; Thayananthan, V. A Multi-Tier MQTT Architecture with Multiple Brokers Based on Fog Computing for Securing Industrial IoT. *Appl. Sci.* **2022**, *12*, 7173. [CrossRef]
30. IOTA 2022: Year of the Ecosystem. Available online: https://blog.iota.org/2022-a-year-in-review-a-year-in-preview (accessed on 17 May 2023).
31. OMNeT++ Discrete Event Simulator. Available online: https://omnetpp.org (accessed on 17 May 2023).
32. Varga, A. Using the OMNeT++ discrete event simulation system in education. *IEEE Trans. Educ.* **1999**, *42*, 11. [CrossRef]
33. TangleSim. Available online: https://github.com/richardg93/TangleSim (accessed on 17 May 2023).
34. Yousaf, F.Z.; Bredel, M.; Schaller, S.; Schneider, F. NFV and SDN - Key Technology Enablers for 5G Networks. *IEEE J. Sel. Areas Commun.* **2017**, *35*, 2468–2478. [CrossRef]
35. Rost, P.; Mannweiler, C.; Michalopoulos, D.S.; Sartori, C.; Sciancalepore, V.; Sastry, N.; Holland, O.; Tayade, S.; Han, B.; Bega, D.; et al. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Commun. Mag.* **2017**, *55*, 72–79. [CrossRef]
36. Ordonez-Lucena, J.; Ameigeiras, P.; Lopez, D.; Ramos-Munoz, J.J.; Lorca, J.; Folgueira, J. Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges. *IEEE Commun. Mag.* **2017**, *55*, 80–87. [CrossRef]