technologies

Article

An Enhanced Lightweight Security Gateway Protocol for the Edge Layer

Md Masum Reza and Jairo Gutierrez *

Department of Computer and Software Engineering, Auckland University of Technology, Auckland 1010, New Zealand; masum.du.mis@gmail.com
* Correspondence: jairo.gutierrez@aut.ac.nz

Abstract: With the rapid expansion of the Internet of Things (IoT), the necessity for lightweight communication is also increasing due to the constrained capabilities of IoT devices. This paper presents the design of a novel lightweight protocol called the Enhanced Lightweight Security Gateway Protocol (ELSGP) based on a distributed computation model of the IoT layer. This model introduces a new type of node called a sub-server to assist edge layer servers and IoT devices with computational tasks and act as a primary gateway for dependent IoT nodes. This paper then introduces six features of ELSGP with developed algorithms that include access token distribution and validation, authentication and dynamic interoperability, attribute-based access control, traffic filtering, secure tunneling, and dynamic load distribution and balancing. Considering the variability of system requirements, ELSGP also outlines how to adopt a system-defined policy framework. For fault resiliency, this paper also presents fault mitigation mechanisms, especially Trust and Priority Impact Relation for Byzantine, Cascading, and Transient faults. A simulation study was carried out to validate the protocol's performance. Based on the findings from the performance evaluation, further analysis of the protocol and future research directions are outlined.

Keywords: IoT security; edge server; sub-server; gateway functions; lightweight protocol; traffic filtering; policy framework

Citation: Reza, M.M.; Gutierrez, J. An Enhanced Lightweight Security Gateway Protocol for the Edge Layer. *Technologies* **2023**, *11*, 140. https://doi.org/10.3390/technologies11050140

Academic Editor: Kyoung-Don (KD) Kang

Received: 28 August 2023
Revised: 27 September 2023
Accepted: 9 October 2023
Published: 12 October 2023

1. Introduction

The number of IoT devices is increasing significantly. Along with this rapid proliferation, the number of vulnerabilities and the chances of security breaches are also rising [1]. A great deal of research has been conducted to secure IoT-based networks, but new vulnerabilities have been found, making it challenging to secure the entire IoT environment [2]. Industrial IoT (IIoT) is one of the promising domains where sensors and actuators perform sophisticated tasks for automation and job efficiency [3]. Here, latency and security are both curtailed for effective IoT communication. It is estimated that over 25% of cyberattacks have been conducted through the IIoT domain [4]. With the development of embedded systems and intelligent technology, devices have been deployed, but they still have constraints in terms of memory, computing power, and energy [5–8]. Lightweight protocols aim to fulfill system and security requirements through considering resource constraints. Minimizing CPU usage, consuming low power (especially for battery-powered devices), implementing overhead light, and reducing memory (RAM) usage are the essential requirements [9]. However, implementing security for these embedded devices has become challenging due to the resource constraints associated with security operations, including cryptographic operations, authentication, and critical management, thus increasing resource usage [5]. This research article proposes an enhanced lightweight security gateway protocol (ELSGP) for the edge layer, aiming to increase system proficiency considering security and latency aspects.

To ensure low latency and QoS for IIoT communication, it is essential to install the edge server near the edge devices [10]. Due to their wide acceptability and low latency communication characteristics, many researchers are interested in deploying edge servers [11]. Edge computing may fulfill the low latency requirement, but security is still a big concern for IoT communication due to the heterogeneity of devices and the physically unprotected environment [1]. Numerous security threats, such as injecting false or malicious data, can cause disastrous outcomes in the entire communication infrastructure [12]. Therefore, a lightweight threat protection mechanism is crucial for the edge layer. A limited number of research works have considered IIoT edge servers for device management, enhanced security, and lightweight communication schemes to fulfill low latency requirements. Therefore, it is predicted that a developed protocol based on these requirements can significantly enhance the proficiency of IIoT edge layer functions.

Based on the above, in this paper, we propose an enhanced lightweight security getaway protocol to enhance edge layer latency and security performance. In this research work, we also define a new type of node called a sub-server. The main objective of the sub-server is to assist the edge devices for computing. The proposed sub-server is not meant to replace the edge servers; instead, its purpose is to assist in distributing the computation power that can be used for security and individual operations. To evaluate the designed protocol and the proposed model, we mainly focused on a highly populated IIoT environment where heterogeneous devices accomplish various operational tasks.

2. Background and Related Work

Industrial IoT (IIoT) is a subset of IoT [4] and it is expanding fast to reduce operational costs, make the industrial processes agile and seamless, and enhance controlling and monitoring [13]. IIoT ecosystems boost industrial productivity but at the risk of increasing the attack surface, which causes security vulnerabilities [13]. To secure the IIoT ecosystem from anonymous threats and attacks, researchers have developed various specialized security models.

2.1. IoT Edge Architecture

The architecture in the IoT edge layer is equipped with numerous devices and protocols [11]. The IoT edge layer architecture contains the following:

IoT devices: IoT devices work as the end nodes; these, depending on their function, work as sensors, actuators, repeaters, integrators, filters, data transmitters, or intelligent devices [1,12].

Gateway: A border gateway is required to communicate between the edge IoT network and another external network [14]. To secure communication between an external network and the internal edge IoT network, the border gateway applies various traffic filtering rules [15]. To reduce the traffic volume in the gateway, numerous reduced traffic algorithms can also be applied. In addition, a gateway routes the traffic from source to destination [16].

Edge servers: The edge server functions as the central hub of an IoT edge network [11] in many applications where time-sensitive communication is required. To fulfill those requirements, the edge server must be installed at the edge of the network. IoT devices transfer data to the network server, and the server processes the received data on behalf of the devices as the devices do not have adequate computing power [10]. The devices can communicate with each other, but that communication process may experience security breaches [2]. The primary working principle of the server is to aggregate and process the data from the IoT devices and transfer the processed data to the gateway to transfer it to an external server (or cloud server) [14]. Unlike traditional servers, these edge servers should be handy, easy to maintain, and cost-effective [10].

Sub-servers: The edge server assists in computing and processing the functions of IoT edge devices. To make the process of assisting the end devices more proficient, we have proposed a new term for a device called a sub-server. The sub-server is like an edge server but is not a substitute for it. The main objective of the sub-server is to distribute

the computation power to perform the functions of the edge devices more efficiently and strengthen the security operations. In terms of the proposed model, the sub-server will be installed near the edge devices to reduce the physical and operational distance between the edge server and end device.

Significance of the Sub-Server device:

In a highly dense operational environment, such as those used with IIoT configurations where edge servers are connected to numerous end devices and work as end device gateways, the sub-server can be installed to achieve the following operational benefits:

Enhanced security: Stronger security operations can be implemented on the edge.

Ensuring high availability: The failure of an edge server may cause an outage or break communication with (or within) the associated end devices. Sub-servers can function as a backup gateway with defined base operations until the recovery of the corresponding edge server.

Load sharing: One of the purposes of this measure is to reduce the operational tasks of the associated edge server.

Distributional operations: Supports co-functional activities, including microservices, adopting SDN (Software-Defined Network) functions, protocol translation, and validating external requests.

Segmentation of the end devices: Supporting segmentation both physically and logically based on end-used protocols or technologies.

Implementing policies: Adopting custom-defined policies and implementing them accordingly.

Cost efficiency: As edge servers are costly, the implementation of sub-servers may reduce the number of operational edge servers (based on the system requirements), which may reduce implementation costs.

2.2. Related Work

To identify cyberattacks, some researchers have opted to design an intelligent architectural paradigm [4]; in the aforementioned work, a low-power IIoT edge gateway was considered for the architectural paradigm. A testbed result was presented where the gateway node used a shallow footprint as the authors presented the gateway device as a constrained device. In addition, an algorithm was developed for randomized cyberattack mitigation for edge IoT devices.

A secured edge computing mechanism has been proposed to facilitate Microservices (μs) for heterogeneous devices in industrial domains [15]. The experimental result of this research study shows that the communication delay was significantly reduced by adopting the μs in the security gateway. However, in this proposed edge computing mechanism, the low computing attribute of the edge devices was not considered to apply the μs on the security gateway. The architecture was designed based on several independent server modules rather than a single server module.

Ref. [17] presented a multi-key-based mutual authentication system where a secure vault was used to collect keys. The secret keys secure the communication between IoT devices and servers. In the IoT system architecture, a cloud-based IoT server is considered to communicate with the end devices over the WAN network. Instead of a single key-based authentication mechanism, a set of keys are used to authenticate the communication between the IoT server and the devices. Here, the end devices' latency and CPU usage constraints are not considered.

The secure mutual authentication protocol (SMAP) uses three techniques to secure the authentication process: pseudo-random number generator, hash functions, and timestamps [18]. The advantage of this protocol is that it does not store the master secret key and does not repeat the session keys, which is more secure than the traditional authentication process.

Ref. [19] presented a data aggregation protocol for a WSN to verify and validate sensed data. In this protocol, data are encrypted, segmented, and subsequently signed with a

homomorphic MAC tag before forwarding. A comparison of overhead size packet drop rate, energy consumption, and transmission delay among different protocols—the access control and authentication (SDAACA) and Efficient Integrity-Preserving DA Protocol (EIPDAP)—was carried out, and the proposed protocol showed enhanced performance.

Ref. [20] proposed a lightweight cryptographic protocol for IoT-based applications, presenting two initiatives: integrating certificateless signatures and bilinear pairing crypto primitives to accomplish security operations on constrained devices. The proposed protocol was performed in a testbed platform that was developed using Raspberry PI 3 Model B. The results of the performance evaluation of this cryptographic protocol show that the total computational cost and time are improved.

Ref. [9] proposed a lightweight protocol for a serverless system to collect data from Mobile Data Collectors (MDCs) and transfer it to a trusted third party. In their study, time-sensitive communication was not considered. In the proposed protocol, a lightweight cryptographic algorithm was used to keep the data and hash functions safe. Automated Validation of Internet Security Protocols and Applications (AVISPA) and ProVerif tools were used to verify the protocol, and it was shown that the protocol can fulfill the security requirements.

To secure the communication between a server and a device, Advanced Encryption Standard Constrained Queuing Telemetry Transport Protocol (AESCQTT) was developed [21]. This protocol is designed to block IoT network and application-level vulnerabilities. Constrained Application Protocol (CoAP) functions (over TLS) were applied by the above-mentioned authors to minimize the communication overhead. A table comparing the performances of CoAP, Data Distribution Service (DDS), and AESCQTT was presented, wherein five parameters (throughput, energy consumption, packet delivery ratio, security, and end-to-end delay) were considered for every parameter, and enhanced performance was recorded for AESCQTT.

Ref. [22] proposed an authentication and key agreement (AKA) scheme to address the security risks and computation costs of the Internet of Drones (IoD). This lightweight scheme secures a one-way hash function and runs bitwise XOR operations to authenticate both user and drone. This proposed scheme was compared with two other schemes [23,24] in terms of bandwidth (communication cost) and computation cost. The results of the performance evaluation show that the AKA scheme has the lowest costs among these three schemes.

Ref. [19] proposed an aggregation tree-based protocol to aggregate data fragments by using data validation and integrity verification. This protocol was designed for wireless sensor networks for the aggregation, verification, and synchronization of fragmented data blocks by using a homomorphic MAC tag. This newly developed Data Validation and Integrity Verification for Trust-based Data Aggregation Protocol (DVIVTDAP) demonstrates better performance in comparison with the Efficient Integrity and Preserving Data Aggregation Protocol (EIPDAP) [25] and the Secured Data Aggregation using the Access Control and Authentication (SDAACA) protocol [26]. Here, the performance of DVIVTDAP was evaluated mainly based on overhead size, detection delays, and energy consumption.

Ref. [27] studied the performance of a developed IoT testbed environment using CoAP and Datagram Transport Layer Security (DTLS) on the Contiki operating system. The comparison between CoAP and CoAP-DTLS showed the experimental results in terms of energy and latency, and the CoAP-DTLS protocol incurs higher CPU usage, memory usage, and latency than the non-encrypted CoAP protocol. Ordinarily, an encrypted platform requires higher resource usage than an unencrypted platform. Here, the authors showed comparative relational data, but the experimental testbed data was not significantly improved to a great extent.

There exists several recent research papers wherein it is identified that either security functions are compromised to reduce the operational tasks of the end devices [28,29] or the constraint of the end devices is not considered enough to adopt the security

functions [15,17,18,30]. To a limited extent, responsiveness during time-sensitive communication has been considered [4,9,31,32].

Considering all of the above requirements, limited research [19,21,22] has been conducted on the adoption of flexible custom-defined policies according to system requirements. Adopting improved services [19,20,31] may produce additional traffic, which may not be relevant to the functionalities of the end devices. This shows the necessity of adopting improved traffic filtering techniques to prevent unwanted resource utilization.

Any considered technique compromises one or many of the following to some extent: security, power, latency, or traffic aggregation, and filtering. There should be a solution that can more closely balance security operations and minimal operational power with improved latency performance. By considering all these metrics, we propose a distributed computational architecture with an enhanced security protocol.

3. Design of an Enhanced Lightweight Security Gateway Protocol for the Edge Layers

This lightweight security gateway protocol aims to enhance security operation by ensuring operational latency preferences. In this section, in order to demonstrate the proposed ELSGP protocol, we will explore the challenges of the existing edge server-based edge layer architecture. Subsequently, we will present how the proposed IoT edge layer architectural model mitigates the explored challenges. Finally, following the presentation of the model, the ELSGP protocol will be presented.

3.1. Challenges of the Existing Edge Server-Based System

In many research studies and previous IoT computational models, edge layer servers are proposed to assist constrained IoT devices. However, for this research paper, we investigated edge server operations and discovered several challenges for edge-server based IoT edge layer communication systems. These challenges include the following:

1. Edge server failure and its impact: In a highly populated IIoT system, an edge server computes, stores, and controls data for many edge devices. The impact of the failure of one server may cause a significant outage or it may increase the load on the other servers.

2. Handling diversified devices: In an industrial scenario, many heterogeneous devices are controlled from or through the edge server. Therefore, the servers need to be designed and configured according to the requirements of the system operations, which is challenging if there is a requirement to maintain the diversified compatibilities.

3. Supporting multi-communication technologies: Several communication technologies need to be incorporated into the server system, which makes the server configuration complex. Wi-Fi, Zigbee, Bluetooth, 6LoWPAN, 5G, and so on are the different kinds of communication technologies that interconnect the edge devices to the system. Maintaining the various technologies from the same server is truly complex and challenging.

3.2. The Proposed Model

The proposed model has been developed to make the edge server-based edge layer architecture more distributed in terms of computation, security, and system operational tasks. Figure 1 shows the proposed edge layer architecture. The end nodes are considered as the IoT devices, which include edge IoT devices, operational sensors, and actuators.

In this architectural model, we have introduced a new network entity called a sub-server, which is a server-like device that is not meant to replace an edge server but is designed to enhance the capabilities of the constrained IoT devices. Implementing a sub-server can help to achieve the following objectives:

To support lightweight communication and advanced encryption at the same time in the edge layer.

To fulfill the latency requirement of sensitive traffic.

To reduce operational tasks at the edge server.

To achieve logical and physical isolation among the group of IoT devices to reduce attack surfaces.

To enable a more distributed computational system.

To provide two-layer validation, i.e., primary and secondary validation (IoT device to sub-server and sub-server to edge server). When the generated data travel from one logical zone to another zone or (to or from) the edge server, this two-stage validation could help to maintain privacy and isolate the devices of one logical group from other groups of the same system.

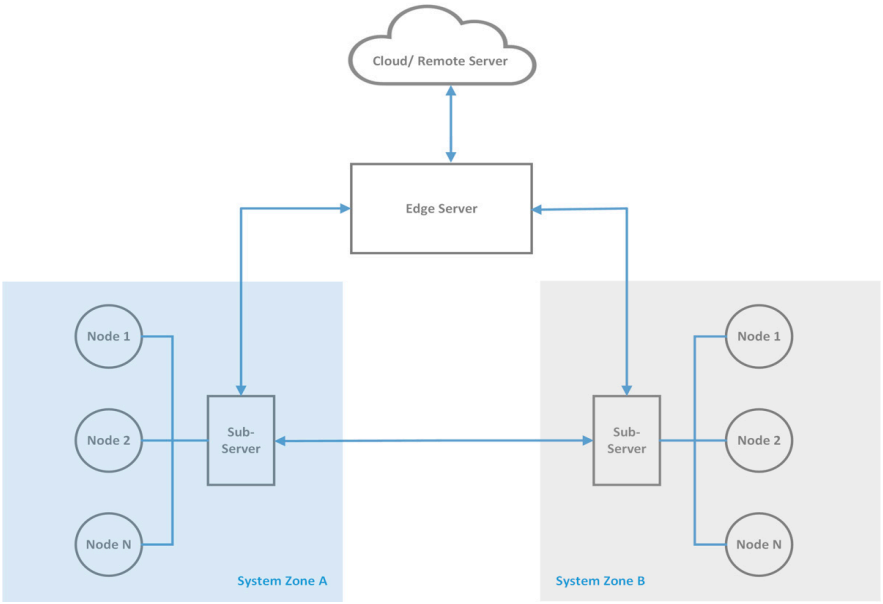

Figure 1. The Architecture of the IoT edge layer.

The end nodes will have client principles, and the Sub-servers will have both server and client principles. Participating sensors in the network can act like a typical sensor network wherein the communication channels can be established in a wired or wireless medium. However, according to this designed architecture, the formation of a wired/wireless sensor network(s) should follow the developed principles, which are elaborated in Section 3.2.1.

Based on a system requirement, one or many edge servers can act as the coordinating server(s). Both sub-server and edge servers should have gateway features, and these features are elaborated in Sections 3.2.3 and 3.2.4. As is the case with the typical IoT architecture, the edge servers are connected to a cloud-based server or a remote server.

3.2.1. Concept of Zones and Groups

In the proposed model, every IoT device is directly connected to a sub-server through a wired or wireless medium. A zone is determined as a physical group of IoT devices that are directly connected and administrated through or from a sub-server. Every end device within a zone should contain the same Zonal ID. Here, the zonal ID is denoted as Z_{ID}. According to the proposed model, every Z_{ID} of a particular network should be configured identically to the others. The Z_{ID} can be determined by the system admin or through a predetermined group policy.

The key differences between the zone and group concepts can be summarized as follows: A zone is the physical segmentation of all operational nodes, whereas a group is all about the logical segmentation of the devices. Grouping can be modified by system control commands, whereas a zone is more resistant to change; a zone can only be changed when a device is physically moved to another zone.

In an IoT system, an end device may need to transfer or receive data from other end devices to execute an operation. Virtual grouping between end devices may require fulfilling the security and independent communication requirements of a system. A group of end devices from the same or different zones can exchange data without repeated

authentication processes, and it is expected to reduce the data transmission latency and computation resource usage. In an IoT system, numerous heterogeneous devices perform operational tasks together. Depending on the system requirements, these devices have various attributes, which may create a complex operational environment. On the other hand, among these devices, many homogeneities or operational dependencies exist. To synchronize and secure system operations with the designed model, we have introduced the idea of grouping.

According to the proposed protocol, a logical grouping (as shown in Figure 2) categorizes end devices based on attributes and predesigned group policy to isolate the group devices from other groups and secure intra-group communication. The devices in a local zone can be the members of a group, but the devices of a group may not be members of a local zone. Group members of a particular group are modifiable through the group policy or under the local or remote administration. In contrast, A-Zone members cannot be modified within the policy framework or local/remote administration unless the physical changes happen in the zone. However, The Z_{ID} can be determined via the system admin or through a predetermined group policy.

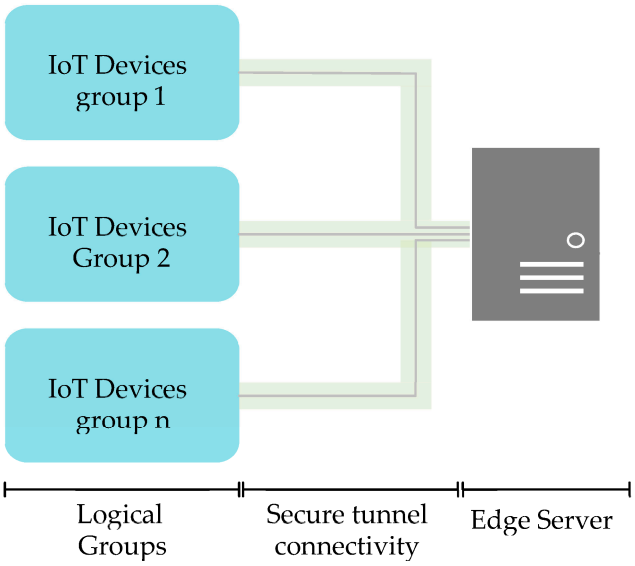

Figure 2. Edge layer logical groups of end devices.

3.2.2. Implementation of Microservices

In a complex IIoT environment, heterogeneous devices need to be operated and maintaining the quality of service (QoS) of the system is challenging. In various contexts, conventional edge computing cannot fulfill the QoS [15]. Therefore, researchers are currently focusing on implementing μs in IoT systems due to their multitudinous properties, including scalability, modularity, resiliency, and compatibility in heterogeneous devices [15].

In the proposed model, multiple operations need to be conducted at different levels of the system, which could make the system more complex. μs can be implemented to resolve this problem and simplify the operational tasks. From protocol translation services to the different kinds of gateway operations in the edge layer, μs can be implemented to make the system more simplistic, resilient, and constrained resource-compatible.

To rescue the operational and computational tasks in a cloud–edge-based hybrid environment, numerous applications need to be processed at the edge to fulfill the latency requirement, and monolithic services can be replaced with μs [33]. The μs will be deployed in a sub-server and edge server according to the predetermined policy framework.

However, maintaining the dependency relationship within the μs is complex in a large IIoT system, as a large amount of data needs to be transferred among the adjacent μs [33]. In a μs-based system, every service request and response should have a unique identifier,

i.e., a correlation ID [34]. Managing and coordinating these unique request processing identifiers requires additional computational resources.

In this distributed computation-based architecture, to make the μs more simplistic, the μs-based applications are processed into two phases. In phase 1, only the relational services will be deployed and operated at the sub-server. The relational services may vary from one sub-server to another sub-server. This way, the transition of redundant data and the processing of redundant operations can be stopped. In phase 2, only the adjacent μs need to be deployed. The adjacent μs also may vary from one edge server to another edge server.

3.2.3. Gateway Functions at Sub-Server (Phase 1)

The end devices are constrained in nature. These devices have limited computational power, operate on light-power batteries, and have less memory in use. The main purpose of the sub-server is to assist the end devices with processing, memorizing, and optimizing power consumption. Many lightweight gateway protocols have been designed to establish communication with the other internal or external devices of a system. These protocols have different characteristics and implementation methods to establish communication among the devices. Many access requests come from the external network of the existing system, though this is not applicable to the in-use lightweight protocols. For example, if a web command passes to the IoT system, it usually passes with HTTP/HTTPS protocols. However, the HTTP/HTTPS is not a lightweight protocol; thus, it may need to be converted into a lightweight protocol like COAP or MQTT [35]. A protocol translation may come into operation to convert the conventional protocols to the lightweight protocols or lightweight protocols to the conventional protocols.

Transmitting external requests: Transmitting, executing, and deploying external requests from a different zone, group, or network requires passing through the various predetermined security points of a particular system to ensure the security requirements. In this circumstance, implementing lightweight compatibility with end devices becomes more challenging. Security operations such as authentication, authorization, data request validation, and encryption require additional computation. Sub-servers control downstream communication (with the end IoT devices), as they can act as a parent device. As a parent device, a sub-server should act like the head-point that responds to the upstream data request, encrypts the transmitted data, encapsulates it, supports the implementation of VPN tunneling, and validates the requests.

Validating requests: In an unprotected environment, IoT devices can become the entry point for an attacker to gain access, which may disrupt the whole system. Therefore, validating the upstream data can be an effective solution to reduce this vulnerability. This type of validation includes analyzing data packet patterns, validating access tokens, handling unknown access and unknown data pattern, and validating unique identifiers like Z_{ID}, Dynamic D_{ID}, and the correlation ID for μs (if μs have been adopted).

Gateway Load balancing: In Figure 1, we present our proposed IoT edge layer architecture, in which sub-servers are interconnected as well as connected with the edge server. To deploy load balancing, the execution command operated by the local/remote server or the edge server (depending on the system architecture and deployment policy), the sub-servers will interact with the load balancing control command and share the deployed tasks among them. The sub-servers also exchange their load status with the parent edge server and within the predetermined sub-server group members. This way, the entire system is deployed based on a distributed processing architecture.

Adopting microservices: In heterogeneous IoT systems, researchers have shown more interest in adopting μs instead of a monolithic architecture. However, implementing and managing the μs in a large and complex environment requires some additional processing, which may increase the use of processing resources at a particular stage. To avoid the additional processing requirements from the end devices, we introduced the concept of a sub-server. The sub-server manages every unique μs request and response on behalf of the end devices, exempting the end devices from additional data processing or computational

resource usage. Converting the existing system solution to a smart solution is always challenging. The existing independent system solutions or monolithic legacy system solutions can still be transformable with the μs [36].

Adopting Software-defined gateway functions: A software-defined gateway (SDG) is designed to focus on industrial interconnections to converge with heterogeneous devices [30]. We adopted the software-defined IoT gateway to automate and dynamically manage the edge network for flexibility and easy programmability [37].

3.2.4. Gateway Functions at Edge Server (Phase 2)

In this architecture, the gateway has some smart functions, including protocol translation between different used protocols, executing μs, executing policy framework, filtering, and processing generated data at the edge before sending it to the remote server or cloud, executing an external operational command, and managing the logical groups of the internal devices. Regarding our proposed model, the gateway functions at the edge server are presented below.

Transmitting external requests: The proposed model is a gateway-centric local network wherein an edge server is the ultimate edge gateway device for transmitting external requests. In this gateway-centric local network, the dependent devices act as the client. The sub-server itself provides a client-supporting functionality as well as a parenting functionality to the end devices. To execute the gateway functions of an edge gateway to the end devices, the corresponding sub-server passes the gateway functions to the end devices. This property of a sub-server can be denoted as Transparent mode; for the edge gateway, it becomes Server mode, and for the end devices, it is Client mode. In summary, in our proposed protocol, these three tire devices have three different modes: Server, Transparent, and Client. The edge gateway and the end devices operate only on the Server and Client mode, respectively, but the sub-server can perform in two modes: Client or Transparent.

Managing the encryption decision: In an IoT environment, it is very challenging to implement a robust encryption method through all IoT communication channels. To implement strong security, strong encryption needs to be implemented. The implementation of encryption depends on the security requirement. The decision to execute encryption from lightweight to strong [38] depends on the security requirement of the policy framework.

Protocol translation: In an IoT network, Different protocols are used to manage, control, and execute the command and establish a connection between external and internal networks. In large and complex heterogeneous IIoT networks, various independent protocols are implemented. Achieving synchronization and interoperability within this large variety of protocols are challenging [39].

Managing logical groups: A logical group enables the devices to logically isolate one group of devices from another. Managing and interconnecting devices within a logical group ensures the implementation of the group policy and other relevant policies. By implementing grouping techniques, devices (including sub-servers) from different zones can be compiled into a group, enhancing security, manageability, and interoperability and enforcing the group policy among the group members. According to our proposed model, an edge server locally assigns coordinates and manages the associated end device groups. An edge server also maintains a correlation with the external devices corresponding to other edge servers to form a group. These external device groups can be called extended groups. The operational and security features of both types of groups (immediate and extended) can be defined through a related group policy in the policy framework.

Dynamic load distribution and balancing: The motivation for introducing the sub-server is to assist the edge server with operational and computational tasks. Dynamic load distribution can be used to reduce the workload from the edge server. As the hub of the local IoT system of the proposed model, an edge server executes commands regarding dynamic load distribution and load balancing depending on the availability of the end nodes, traffic transmission channel preferences, system preferences, and deployment policy.

Filtering the generated data: To avoid overwhelming the whole IoT system, the filtering functions filter the routine or unnecessary data and send the critical and infrequent data away for future analysis and development purposes. Traffic filtering techniques are implemented to optimize the network traffic to comply with the available network bandwidth. The filtering techniques used for the proposed protocol were developed via categorizing, prioritizing, and processing the generated traffic based on traffic attributes and the generating device.

Adopting Microservices: At the edge server, the μs are coordinated and deployed to the associated sub-servers based on the operations that need to be performed. Every microservice deployment and operation can be identified with a unique correlation ID that correlates with a processing request. In this IoT service model, the functions of the μs include (but are not limited to) authentication, protocol translation, managing logical groups, software-defined gateway functions, ABAC operations, deploying filtering decisions, managing, and deploying core policy framework.

Adopting SDN-enabled gateway functions: In the context of evaluating IoT networks and network management features, SDNs have become a promising form of technology for enhancing the programmability, resiliency, and security of an IoT network [37]. SDN-enabled gateway functions are adopted to execute advanced gateway functions. The control commands of remote or cloud servers are managed at the edge server and sub-server for execution at the edge network. In the proposed model, the developed policy framework defines the SDN-enabled gateway functions.

3.2.5. ELSGP Operational Flow and Policy Framework

Connectivity, privacy, security, adoption with open standards, device and data heterogeneity, and interoperability are the key concerns for the evaluation of IoT domains [40]. We have developed a prototype of a policy framework based on the above-mentioned key concerns of IoT domains. We focused on the IIoT domain for the development of the policy framework. To demonstrate and execute the policy framework functions in the proposed model, we developed algorithms with the main functions of the designed ELSGP.

In Figure 3, we present a flowchart of the used ELSGP operational streams. The ELSGP operations followed the developed policy framework. The flow of the policy framework execution is illustrated, along with several identical operations, which include the following: developing ABAC operations; defining protocol translation requirements and operational flow; transmitting traffic filtering decisions; authentication requirements; secure tunneling requirements; the deployment process of the software-defined gateway functions; defining and developing gateway operations (if software-defined gateway functions are not deployed); developing group, zonal, and network policies (depends on the executing model; here, we have considered our proposed model); and designing and developing μs.

The policy framework deployment depends on the nature of the device. Our developed model contains three types of devices at the edge layer: end devices (end IoT devices like sensors and actuators), a sub-server, and edge servers.

Dynamic policy generation, adjustment, and enforcement are currently being researched for the deployment of various software-defined functionalities [41]. To alleviate the critical challenges via the use of the designed ELSGP protocol, a policy is dynamically deployed on each node and communication channel based on the predefined modular functions, functional IDs, and device IDs.

3.3. Design of ELSGP

The ELSGP was designed with six basic functions: attribute-based access control, authentication, dynamic load balancing and distribution, dynamic interoperability function, traffic filtering, and secure Tunneling. Algorithms were developed based on the functions of the proposed ELSGP.

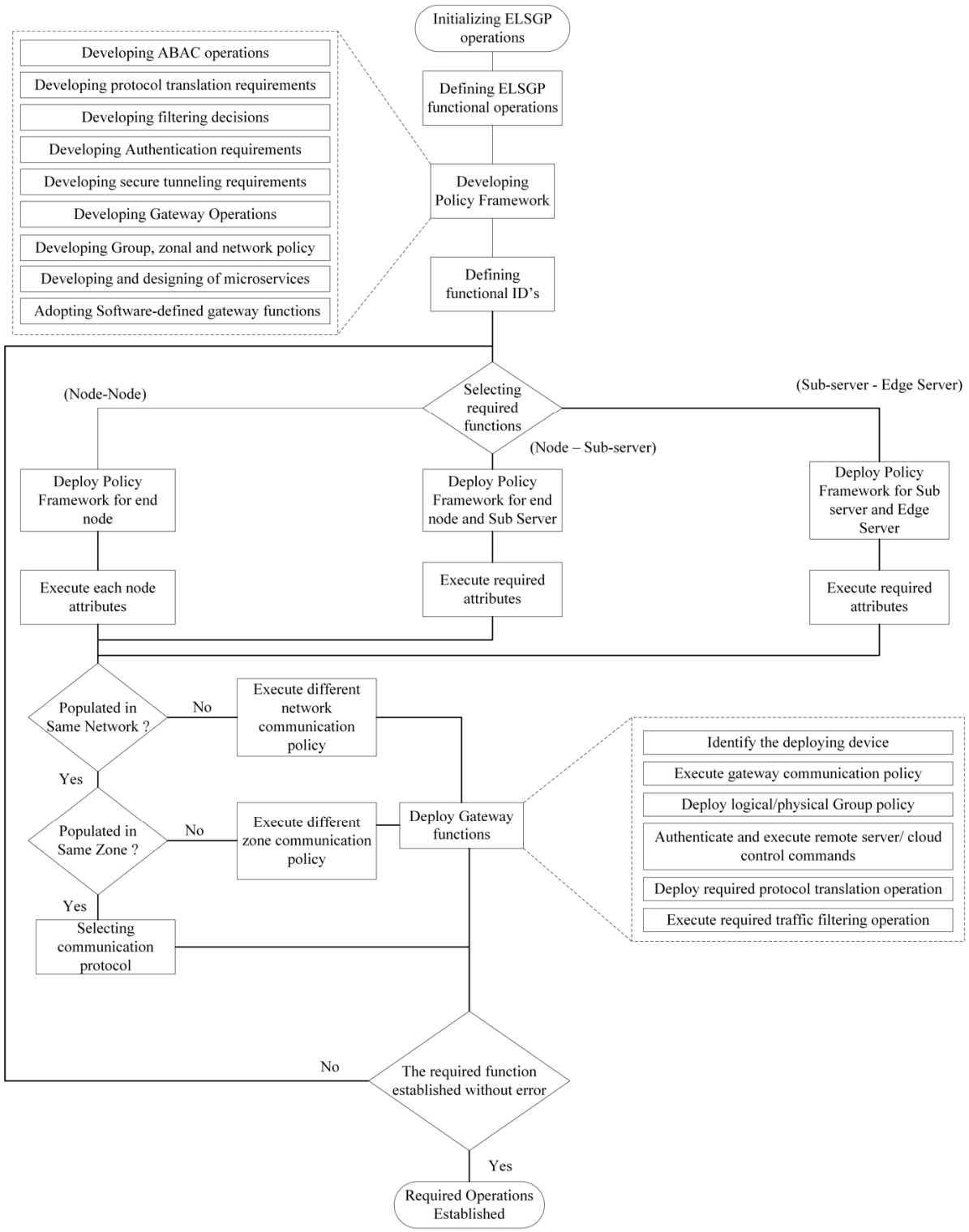

Figure 3. Operational flow chart of ELSGP and policy framework.

3.3.1. Authentication

In the edge server gateway, several security operations need to be processed and validated at the edge layer [42]. Edge-based authentication will increase data communication security among the nodes [18]. Untrusted request management is one of the security operations that needs to be conducted before processing the generated data from an end node [42].

The proposed ELSGP authentication process is designed with three parameters: *MAC*, D_{ID}, and an access token. In this protocol, to enhance security and access control, every operating device in the IoT system network will have multiple D_{ID}, which a sub-server will manage. An edge server will manage the access token. The D_{ID} will be allocated dynamically depending on the system security requirement. A single D_{ID} will expire depending on the defined operating time to authenticate one or more operational request(s). In this scenario, the D_{ID} can be allocated dynamically, and the dynamically deployed device ID is denoted as DD_{ID}. An AND operation between *MAC* and D_{ID}/DD_{ID} will authenticate the node requests and the node-generated data. IoT device communication WITHIN a logical group or a system zone will be validated via *MAC*, D_{ID}/DD_{ID}, and the access token (determined depending on system security level and lightweight weight performance requirement).

The authentication process is facilitated via the use of two algorithms. The first algorithm is for access token distribution and validation, and the second algorithm calls the described functions in the access token distribution and validation algorithm to the authentication algorithm. The authentication process cannot be independently conducted, but it depends on the participating protocol(s) to finalize the authentication process (Algorithms 1 and 2).

Algorithm 1 Access Token Distribution and Validation

1. $F(AT_D)$//Defining Access Token Distribution function
2. $F(AT_V)$//Defining Access Token Validation function
3. D_{ID}, Z_{ID}, N_{ID}//Defining Device ID, Zone ID, and Network ID
4. $F(DD_{ID})$//Defining function for Dynamically defined device ID
5. *Get_Source ()*//Function for Identifying source operating device
6. *Get_Destination ()*//Function for Identifying destination operating device
7. **Initialize** $F(DD_{ID})$//Operation at Sub-server
8. **Step A: Get** pattern
9. **Define** DD_{ID} Bitstream format
10. **Step B: Set** DD_{ID}
11. **Generate** DD_{ID}
12. **Allocate** DD_{ID}
13. **Step C: Reset** DD_{ID}
14. **Define** Reset Period//the reset function triggering interval
15. **Repeat** {**Generate** DD_{ID}
16. **Allocate** DD_{ID}}
17. **Initialize** $F(AT_D)$//Operation at edge server
18. **Step A**: *Get_Source ()*
19. **Get** D_{ID}/F(DD_{ID})
20. **Get** Z_{ID}, N_{ID} and MAC
21. **Step B:** *Get_Destination ()*
22. **Get** D_{ID}/F(DD_{ID})
23. **Get** Z_{ID}, N_{ID} and MAC
24. **Step C: Generate** AT//Access token (AT) generating operation
25. **Re-Generate** AT
26. **Step D: Distribute** AT//(AT) distributing operation
27. **Sync** regenerated AT
28. **Initialize** $F(AT_V)$//Operation at edge server
29. **Step A: Verify** AT Timestamp
30. **Step B: Verify** AT Bitstream
31. **Step C: Validate** AT
32. **If**
33. *Step A* **AND** *Step B* = True
34. **Else**
35. validation unsuccessful

It is essential to establish a trust relationship between an end node and a server to validate the generated traffic from a trusted device. Our developed access token validation and distribution technique can be an effective solution to establish a strong trust relationship between the server and the end IoT nodes. The end devices are allocated with the device ID. Depending on the system preference, the allocation can be either statically user-defined or dynamically system-defined. For dynamic device ID allocation, a sub-server operation, the function $F(DD_{ID})$ is used. The second function, $F(AT_D)$, is for access token distribution, an edge server operation. The third and the final function of the algorithm is to validate the distributing access token. The two main steps of this function are to verify the timestamp of the received and generated access token and verify the bitstream.

The bitstream of an access token is predefined but should not be identifiable by an end device or an operating sub-server. The access token bitstream can be verifiable at the edge server, which is expected to enhance the trustworthiness between the transmitting devices without additional computing requirements at the end device. In Algorithm 2, function *F1_Auth* is initialized for establishing the authentication and protocol translation functions. *F1_Auth* executes the access token distribution, validation, and dynamic device ID allocation functions, as presented in Algorithm 2.

This authentication process is only for authenticating end devices and the generated traffic from the nodes (not the user). User authentication will be performed with the used protocol.

Algorithm 2 Authentication and Dynamic Interoperability functions

```
F1_Auth () // Defining Authentication function
F2_IO () // Defining interoperability (IO) function
F(P) // Defining protocol translation function
deployment_id // unique deployment id as μs identity
P// operating protocol
    Initialize F1_Auth
        Initialize F(AT_D) // For both server and client
        Initialize F(AT_V)
      Validate D_ID, Z_ID, N_ID // For both server and client
        Initialize F(DD_ID) // If the system authentication requirement allowed
                    If Check result is TRUE
                        Authorize device
                        Authorize generated Traffic
                        Check IO requirement
                        If check returns True
                            Initialize F2_IO ()
                            Get system configuration information
                                        Get deployment_id
                      Request μs // from cloud or remote server
                      Get μS// operation at edge server, getting the requested μs
                      Deploy μS //Deploy μs operation at Sub-server
                        Initialize F(P)
            If F(P) = Success
            Get P //The operating communication protocol(s) P
              Get entities of the operating protocol
              Authorize protocol translation service
                    Else stop translation function to find protocol entities
                Else stop IO function
              Else Reject Authentication request
```

3.3.2. Dynamic Interoperability Function

In a real IIoT environment, many communication protocols and operational standards are outlined, which makes interoperability and scalability more challenging [43]. Recent

research has uncovered several solutions to the interoperability problem, including protocol proxy, protocol translation, and middleware or MiddleBridge [39,44,45].

Example Scenario

An IIoT system can be operated via the use of various protocols, such as CoAP, MQTT, REST, AMQP, HTTP, HTTPS, WebSocket, and so on. In this sample scenario, we assume an IIoT system is operated with A and B protocols. Protocol A complies with advanced encryption, where the computational requirement is much higher than the maximum operational requirement of the end devices. Therefore, a much lighter protocol, protocol B, is used for the end devices. Various portions of the edge layer of the system are operated with some legacy solution. An edge server converts the transmitted traffic between protocols A and B.

The existing problems of the IIoT system include the fact that the edge layer is vulnerable to low-security exposure, requires massive operational tasks for translating the protocol, is hugely complex in terms of system management and maintenance (not dynamic), and makes it very hard to adopt safer and new technology due to the legacy solutions. The system requirement is to mitigate these existing problems.

Existing solutions: Middleware or MiddleBridge is hardware that works as a protocol converter. However, the middleware solution is mainly adaptable with network-layer protocols but not with the application layer protocols, and this is one of the limitations of middleware [39]. Using a protocol proxy is another solution for the interrogatability problem, but the protocol proxy solution is not efficient enough in an IIoT environment wherein the number of participants is high [39,44]. Protocol translation can mitigate the limitations of the middleware and protocol proxy techniques. Protocol translation is required where two or more protocols (from network layer protocol to application layer protocols) are used to establish communication. This translating mechanism bridges two different protocols that can convert the proprietary aspects of the entities or the data standard of a protocol to a different protocol [45]. Executing a large operation every time at the edge layer and securing seamless interconnection and configuration compatibility for the heterogeneous end devices can be challenging for the protocol translation.

The developed interoperability functions: The interoperability function should be secure, dynamic, and compatible with various end device configurations (including legacy solutions) while using the least amount of operational tasks possible. The interoperability operation can also securely convey remote control commands to the edge layer. The developed interoperability gateway function supports strong encryption up to the edge layer distribution point (sub-server) and complies with the enhanced lightweight security protocols when an end device communicates with a remote/cloud server (or with another end device of a different network). The implementation of these security features also depends on the adopted policy framework or the system security level and the lightweight performance requirements.

A master server (according to the proposed model, the master server is an edge server) will execute the configurations of the interoperability function. Depending on the system configuration information of an IIoT edge layer, the μs deployment will vary. All the interoperability functions will be deployed as one or many μs. Every different μs deployment should have a unique deployment ID. By sharing the system configuration information, the master server will request the deploying file, which should be stored in a remote/cloud server. For example, in the Azure cloud service, the deployment ID is denoted as a parameter named deploymentid, which is needed to execute deploy the JSON file [46]. Once the system environment is changed, the interoperability function executes different deployment files with the associated μs. The master server deploys the μs to a designated device (according to the proposed model, the designated device is a sub-server), and the designated device executes μs to all the associated end devices.

In Algorithm 2, *F2_IO* is denoted as the interoperability function. The steps of initializing *F2_IO* are getting the system configuration information and requesting the deploying μs.

If the μs discover the protocol translation requirement, the function $F(P)$ is executed to find the operating protocol. If different protocols participate in completing the communication, then the protocol translation function $F(P)$ requires translating theparticipating protocol entities. For example, two protocols participate to establish communication where one protocol needs to be transformed into another. The protocol translating function translates the entities of one protocol into another protocol's entities. If one protocol establishes the communication, then the protocol translation function cannot be executed.

Referring to the above sample scenario, for the interoperability between protocols A and B, the $F(P)$ function can be dynamically executed. The associated deployment file can be executed by obtaining the system configuration information, including the legacy solution. The deployment can be recorded with deployment ID for similar use. Repeated system management and maintenance tasks can be performed dynamically.

3.3.3. Attribute-Based Access Control (ABAC)

Every operating device can be identified with its individual identification (ID) depending on the attribute of the device, which is called the attribute tag (A_{TAG}), to control the device's access to a system network. The A_{TAG} must incur a light overhead to fulfill the network bandwidth requirement(s) and the round-trip delay.

To validate the ABAC operation, an A_{TAG} along, with an Authentication function (*F1_Auth*), will authenticate the device access request to join onto a network as shown in Algorithm 3, communicating with the other network devices, validating the generated or received data, and executing and validating the control command.

Algorithm 3 Attribute-based access control

```
F3_ABAC () // Defining Attribute based access control function
A_TAG // Defining Attribute TAG from Attribute policy
R_F3 () // function for defining the rules from the policy
Begin
    Initialize F3_ABAC ()
    Initialize R_F3 ()
    If R_F3 () = Success
        Check ATAG
        If F1_Auth () AND A_TAG = True
            Allow traffic flow
          Else
            Reject traffic flow
     Else
        Stop F3_ABAC ()
        Acknowledge error initialization
end
```

3.3.4. Traffic Filtering

The traffic filtering function of the designed protocol is to optimize the network traffic to comply with the bandwidth requirement. Categorizing, prioritizing, and processing the generated data based on the attributes and filtering the repeated traffic are the key functions of the traffic filtering process shown in Algorithm 4.

In the traffic filtering function, transmitted traffic is categorized into four types: regular traffic, priority traffic, flooding traffic, and unknown traffic, denoted as T, PT, FT, and UT, respectively. Periodic traffic can be defined as regular traffic. At a certain interval, periodic traffic indicates the status of known events. The traffic filtering function F4_TF () is developed with four subfunctions: a traffic classifier, priority function, summary function, and traffic analyzer function. Traffic filtering policies may vary in different IoT systems. The subfunctions operate according to the rules defined in the traffic filtering policy. The

traffic analyzer subfunction categorizes the transmitted traffic as unknown traffic if the nature of the traffic cannot be determined with predefined rules.

Algorithm 4 Traffic Filtering

```
T, PT, FT, UT //regular traffic, priority traffic, flooding traffic, unknown traffic
F4_TF ()// Defining the traffic filtering function
Traffic_Filtering_Policy // defining traffic filtering policy
TC ()// defining Traffic classifier function
PF ()// defining priority function
SF () // defining summery function
TA ()// Traffic analyzer function
Begin
    Initialize F4_TF ()
    Initialize TC ()
    If TC ()→ T = true
            Deploy Traffic_Filtering_Policy
     Elseif   TC ()→ PT = true
            Execute PF () AND
            Deploy Traffic_Filtering_Policy
     Elseif TC ()→ FT = True
            Execute SF () AND
            Deploy Traffic_Filtering_Policy
       Else TC ()→ UT =True
            Stop UT // stop unknown traffic flow
            Execute TA ()
                        If TA () =False //Traffic analyzer function return
                Block UT
                Update Traffic_Filtering_Policy
              Else
                Allow UT
End
```

3.3.5. Secure Tunneling

The proposed protocol aims to validate internal communication (within a system zone) and external communication (with devices or servers out of a system zone). Device identity parameters, encapsulating function, and secure tunneling policies will be used to establish a secure tunnel between two nodes of different networks or system zones.

In Algorithm 5, the secure tunneling function is denoted as *F5_ST ()*. The subfunction *P ()* is initialized to establish a logical path between the source and destination devices, where several identification parameters are used for accomplishing the functional operation. Depending on the level of the secure tunneling preferences, a secure communication tunnel can be formed in three ways: a full strong encrypted state, a lightweight state, and a policy-defined state.

Full strong encrypted state: A full strong encrypted state is formed when the source and destination devices are connected with a strongly encrypted tunnel. *Enc ()* is required to outline the steps of forming a secure tunnel where the parameter *Enc_Dec* is needed to define which protocol will be used for the strongly encrypted tunneling protocol, such as Secure Shell (SSH) or Secure Socket Tunneling Protocol (SSTP), an internet protocol security (IPsec). However, in an IoT environment, these protocols are not used to establish a secure connection between two devices due to the constrained nature of the IoT devices. These protocols are only used temporarily to connect with devices to execute control commands. For example, Amazon Web Services (AWS) IoT secure tunneling is formed for a short-term to set up a secure connection with a remote IoT device [47].

Algorithm 5 Secure Tunneling

```
F5_ST ()// Defining the secure tunneling function
O, G, Z, N //Object, Group, Zone, and network
Encap_Dcap, Enc_Dec // Encapsulation and encryption requirement value
Tunnel_sec // the secure tunneling policy
Encap(), Dcap (), Enc_t ()// defining encapsulation, decapsulation, encrypted tunneling functions
P() // network path defining function
Begin
  Initialize F5_ST ()
  Get O, G, Z, N // for Source
  Get O, G, Z, N // for destination
  Initialize P ()
  Get Logical path // between source and destination
  If Encap_Dcap= true // at lightweight state
      Initialize Encap() // For transmitting traffic
      Get keys// for device itself and next-hop device (from Source)
      Set XOR // XOR operation between the key and data
          O → Z → G → N //Defining encapsulation order
          Initialize Dcap ()// For retrieving traffic
          O← Z← G ← N // //Defining decapsulation order
          Get keys // for device itself and next-hop device (from Destination)
          Set XNOR // XNOR operation between the key and received data, for retrieving transmitted traffic
        Else
          Stop Encap()
          Stop Dcap ()
        If Enc_Dec= true
          Get protocol
          Initialize Enc_t ()
        End if
        If Tunnel_sec =true // at lightweight state
          Execute Tunnel_sec
        End if
End
```

Lightweight state: At a very low computing stage in which a full strong encrypted tunnel cannot be created, a secure encapsulation method can be applied to transmit traffic between a source and a destination device. In the above-listed algorithm, we presented a method of secure tunneling wherein the parameter *Encap_Dcap* is required to identify the lightweight state. If the channel is required to form with a lightweight state, then the parameter *Encap_Dcap* returns True and the subfunction *Encap()* is initialized. In the lightweight state, two secret keys are exchanged within the source and destination end. One key is for the source device itself and the other key is for the next-hop device (where the hop count is 1). An *XOR* operation is set between the exchanged key and the device-generated data frame. This process is repeated at the next-hop device but with different keys. Data encapsulation is formed using this order $O \rightarrow Z \rightarrow G \rightarrow N$. Data decapsulation is for the destination, which is directed using the reverse order, followed by an *XNOR* operation to retrieve the received data frame.

Policy-defined state: A policy-defined state is also part of the solution, this arrangement fulfills the requirement of balancing full strong encryption and a lightweight state. *Tunnel_sec* defines the secure tunneling policy, which includes the participating protocol and policy that applies to the configured network segments. The policy deployment file will be defined with a unique deployment ID for future reuse.

3.3.6. Dynamic Load Distribution and Balancing

The function of dynamic load distribution and balancing depends on several parameters of the edge servers and sub-servers. The parameters are divided into two categories: load information parameters and load status parameters. The load information parameters include the number of operating sub-servers, the load threshold values for each edge server, the load threshold values for each sub-server, the health check threshold values, the task queue threshold values for each edge server, and the task queue threshold values of each participating sub-server. The health check value depends on the CPU capacity, power status, channel status, computing time, and system preference.

The load status parameters include health check current values, current load at the designated sub-server, the current task queue value of the designated edge server, the current task queue value of the designated sub-server, and identifying the sub-server(s) available for load sharing. In Algorithm 6, the dynamic load distribution and balancing function is split into three steps: obtaining the load information parameter, obtaining the load status, and defining the load balancing and distribution function.

Algorithm 6 Dynamic load distribution and balancing

F6_dldb ()// Defining the Dynamic load distribution and balancing function
Step A Get load information parameters
    N_{ss} //number of sub-servers
    L //load threshold value for edge server
    l //load threshold value for sub-server
    h_t //health check threshold values
    TQ_t //Task queue threshold value of an edge server
    tq_t //Task queue threshold value of a sub-server
Step B Get load status parameters
    h //current health check values
    cl //current load at designated sub-server
    CL //current load at designated edge server
    TQ //Current task queue value of the designated edge server
    tq// Current task queue value of the designated sub-server
    ls //available sub-server(s) for load sharing
Step C Balance load
    Initialized *F6_dldb()*
    if ($CL \geq L$ *or* $TQ \geq TQt$)
        Calculate ΔL // $\Delta L \leftarrow (CL\text{-}L)$
        Select *ls*
            If $cl < l$ ***AND*** $h_t < h$ ***AND*** $tq < tq_t$
            Endif
        Distribute $\Delta L \propto \Delta l$ *or* $\Delta TQ \propto \Delta\ tq$ //$\Delta TQ \leftarrow (TQ\text{-}TQt)$ *and* $\Delta tq \leftarrow (tqt\text{–}tq)$
        Update load parameters
        Update status parameters
    Else *repeat* **Step c**

In step C, the load balancing and distribution function is defined. If a server exceeds its load threshold value or task queue threshold value, then *F6_dldb ()* is initialized. A sub-server is selected for load sharing if the current load value is less than the threshold value AND health check values are better than the threshold value AND the current task queue value is less than the threshold value. The exceeded load of an edge server ΔL distribution is proportional to the Δl (difference between the threshold load and the current load of a sub-server), and ΔTQ is distributed with the proportion of Δtq. The load information and status parameters are updated according to the executed modifications.

4. Analysis and Evaluation

In this section, we analyze and evaluate the performance and features of the designed protocol based on simulation studies. Section 4.1 presents the mathematical modeling of the distributed computational system. Section 4.2 presents our analysis of the developed algorithm. An evaluation of the features with several fault models is presented in Section 4.3. We also present a probability analysis, an explanation of the simulation process, and a comparison and evaluation of the findings, respectively, in Sections 4.4–4.6.

4.1. Modeling of the Distributed Computational System

A distributed model is required to enhance the performance of a system by reducing the load from the centralized servers. Assume that an edge layer of a system has N_I number of IoT devices, N_{SS} number of sub-servers, and N_{ES} number of edge servers. Therefore, the total number of nodes, N, in the edge layer of the system is equal to the sum of the numbers of IoT devices, sub-servers, and edge servers. Therefore $N = N_I + N_{SS} + N_{ES}$. With this distributed system architecture, the number of the nodes must follow the following order: $N_I > N_{SS} > N_{ES}$.

The IoT nodes are directly connected with an adjacent sub-server, which accumulates the number of IoT nodes under each sub-server. Therefore, the number of adjacent IoT nodes with the sub-servers will be as follows:

$$NI = \sum_{1}^{Nss} (I_{N_{SS}})$$

Here, I_{NSS} represents the number of adjacent IoT nodes with each sub-server. If we denote the number of nodes including the sub-server itself as S_S, then

$$E_S = \sum_{1}^{N_{ES}} (S_{SN_{ES}} + 1)$$

Similarly, if we denote the number of nodes including the edge server itself as E_S, then all sub-servers and IoT devices are considered to be nodes. In other words, the total number of nodes in the edge layer of a system N is equal to the accumulation of the numbers of nodes adjacent to each edge server E_S.

In a nutshell, the hierarchical modeling and topological definition of the distributed commutation system are presented here. In an edge layer, all the operations need to be performed with resiliency, adaptability, stability, reliability, and effectiveness. Considering all these metrics, modeling presents how every functioning node will be interconnected in the edge layer of a system.

4.2. Analysis of the Algorithm

We present the formulation of the protocol features relating to the computational complexity of IoT devices. Here, computational complexity determines the resources required to execute the algorithm and dependent functions.

4.2.1. Access Token Distribution and Validation

In a predefined period, the frequency of generating DD_{ID} is denoted as F_p; the period is defined according to the system requirements and policies. The generated DD_{ID} is considered to be the periodic traffic, which is the subclass of one of the classified traffic types in Section 3.3.4. For the frequency, F_p of DD_{ID} allocation per defined period p, we assume n_s number of time slots, where every time slot duration is t. The p and F_P can be defined as follows:

$$p = \sum_{1}^{n_s} t_{n_s}$$

$$F_p = \frac{\sum_1^{n_s} t_{n_s}}{t} \text{ or } F_p = \frac{p}{t}$$

To calculate the amount of traffic generated due to the distributed access token, the associated parameters need to be determined. According to the algorithm developed in Section 3.3.1, four different types of identification information are associated with access token distribution and validation: D_{ID}, MAC, Z_{ID}, and N_{ID}. We assume the average size of the packets associated with the identity information as I_p, which contains the information of D_{ID}, MAC, Z_{ID}, and N_{ID}. Then, the size traffic overhead T_{OA} is as follows:

$$T_{OA} = I_p$$

However, if the D_{ID} is generated dynamically after a predetermined period, the packet size will be $[D_{ID}]^{Fp}$. During the communication initializing time, other identification information is required, including MAC, Z_{ID}, and N_{ID}. The packet size is denoted as $I_{pex,}$ which contains the information excluding DD_{ID}. The amount of traffic overhead can be calculated as follows:

$$T_{OA} = I_{pex} + [D_{ID}]^{Fp}$$

Therefore, the traffic overhead due to the access token distribution and validation can be calculated as follows:

$$T_{OA} = [I_{pex} + [D_{ID}]^{Fp}\ ] \mid\mid [I_p]$$

4.2.2. Dynamic Interoperability and Secure Tunneling

The authentication function is completed with the access token validation result. The authentication function is initialized if the validation result returns True. In Sections 3.3.1 and 3.3.2, we presented the authentication and dynamic interoperability mechanism. The interoperability function requires system configuration information from every associated node, which initiates the associated end IoT nodes to generate packets in response to the request. We assume that the volume of traffic generated with the interoperability function is T_{IO}.

Secure tunneling creates a logical path between the source and destination devices. A logical path can be defined in three ways, namely, a full strong encrypted state, a lightweight state, and a policy-defined state (we introduced these states in Section 3.3.5). Based on the system preferences, one of these processes will be executed. Among the three, the full strong encrypted state consumes the most resources, and the lightweight state requires the least. The resources include memory, processing unit, communication unit, and power sources [48].

If we consider the average traffic overhead due to establishing a logical path as T_{ES}, T_{LS}, and T_{PS} for the full strong encrypted state, lightweight state, and policy-defined state, respectively, then the traffic overhead due to secure tunneling, T_{ST}, can be calculated as follows:

$$T_{ST} = [T_{ES}] \mid\mid [T_{LS}] \mid\mid [T_{PS}]$$

If we assume T_O as the total traffic overhead of the proposed ELSGP protocol, then T_O can be calculated as follows:

$$T_{O} = T_{OA} + T_{IO} + T_{ES}$$

4.3. Fault Analysis

The failure of one or multiple critical devices may cause substantial losses, and the impact may spread throughout the system. To analyze faults, we considered several well-recognized fault models, including the Byzantine fault model, the Transient fault model, and cascading failure. The well-known Byzantine fault model was chosen to interpret the resiliency against fault nodes (due to identified vulnerability, affected with known or unknown attacks, infected with malware, exhibiting malicious behavior, and so on). Transient faults

and cascading failure cause the instability of a system. We considered both faults as these faults because both may cause tremendous impacts throughout a system. All these three fault models and the remediation processes against the faults are elaborated on below.

4.3.1. Byzantine Fault

Byzantine faults derive from the Byzantine General Problem (Castro et al., 1999). With this problem, in a distributed system, a component fails to perform or appears with imperfect information. The node that is seemingly malfunctioning is called a Byzantine node. To tolerate the fault, the number of faults should be fewer than 1/3 of the total nodes [49]. In other words, if a system has f number of fault nodes, there will be $(1 + 3f)$ number of nodes to tolerate the fault. This fault tolerance is essential because of the increasing attacks and arbitrary behavior of malicious nodes. To enhance fault resiliency, we developed a Trust and Priority Impact relation, which is presented in the following sub-section (Section 4.3.2).

4.3.2. Trust and Priority Impact Relation

Depending on the significance of the tasks, the viability of resources for security operations, and the threats and vulnerability protection mechanisms, we determined different degrees of trust, as shown in Table 1. Here, the lower the value of the degree of trust, the higher the trustworthiness, and we consider the degrees of trust as D_t.

Table 1. Different degrees of trust.

Degree of Trust	Types of Nodes
D_t^4	End nodes/IoT devices/sensors/actuators
D_t^3	Sub-server
D_t^2	Edge server
D_t	Remote server/control server/cloud

Priority impact factor: The priority impact factor is the inverse of the degree of trust, which presents the impact of the faultiness of a node. We assume the priority impact factor as P_i and p as the power of degree of trust a node; therefore,

$$p_i = \frac{1}{D_t^p}$$

If we consider n as the number of nodes, then the priority impact of the faultiness of nodes can be presented as follows:

$$p_i = \frac{\sum_0^n (p_n f_n)}{n}$$

During the fault scenario, other nodes will determine P_i; here, a higher value of P_i means the higher the impact of the faultiness, and a lower P_i means the lower the impact of the faultiness. We calculate the degree of trust for n number of devices using the following:

$$\frac{1}{{D_t}^p} = \frac{\sum_1^n (p_n f_n)}{n}$$

$${D_t}^p = \frac{n}{\sum_1^n (p_n f_n)}$$

$$D_t = \sqrt[p]{\frac{n}{\sum_1^n (p_n f_n)}}$$

where n is the number of devices with the same degree of trust. If we consider a group of devices with different p, then the D_t of the group of devices can be calculated as the multiplication of the D_t values with different p values.

4.3.3. Transient Faults

Transient faults are very hard to prevent because they occur for a limited duration due to many reasons (including malfunctioning, power outage, a network being busy, and so on), and the system comes back to normal when the fault disappears [50]. No prevention mechanism was introduced into our distributed computing model. However, we have adopted a fault mitigation mechanism to provide resilience to the system. The mitigation process is made up of four stages, all of which are presented below.

1. Get: exchange the system log among the non-faulty nodes.
2. Diagnose: diagnose the log to identify the cause of the fault.
3. Organize: plan a fault resolution process.
4. Commit: execute the predetermined resolution plan and acknowledge the fault resolution status from the faulty node(s).

With these four stages, a system will mitigate the transient fault. During the Get process, the packets for the logs are categorized as priority traffic, as defined in Section 3.3.4.

4.3.4. Cascading Failures

Cascading failures cause devastating consequences for a system. A cascading event happens with a node and triggers the event in other nodes [51]. Ref. [51] conducted a systematic review of fault modeling and analyzed the mitigation process by considering and understanding the cause of the failure. To ensure resiliency and reliability in the distributed computational system, we developed an isolation mechanism as a part of the access token validation and distribution process, which will enhance the security of the system and mitigate the cascading failure. In Section 3.2.1, we presented the concept of zones and groups, which will isolate a physical or logical group from others. By observing the nature of the failure, cascading failures trigger a domino-like chain effect. Although there is a chance that the failure can be contained within a portion of the system, the isolation technique will prevent the fault from spreading to a larger system portion or to the whole system elements. Hence, the mitigation mechanism can ensure reliable resiliency against failures.

4.4. Probability Analysis

When some of the system components behave arbitrarily and show unpredictable behavior, it generates traffic in response to the unstable condition (the categories of the generated traffic were presented in Section 3.3.4). With this uncontrolled scenario, the components are recognized as faulty. We assume that the probability of f number of faulty devices in a system is equal to p(f). Hence, the probability of non-faulty devices is equal to $(1 - p(f))$. If we assume that faultiness is a discrete event that happens within a period, P, then the probability of faultiness can be calculated as follows:

$$P(f_P) = \frac{|F_{Tstart} - F_{Tend}|}{P}$$

$$P(f_P) = \frac{F_{\Delta T}}{P}$$

where F_{Tstart} is presented as the fault start time, F_{Tend} as the fault end time, and $F_{\Delta T}$ as the duration of the fault. If we assume a fault event with responding traffic f_t and the probability of generating traffic while responding to the faultiness as $P(f_t)$, then it can be calculated as the sum of the probability of each type of traffic. Therefore,

$$P(f_t) = P(f_{ut}) + P(f_{pt}) + P\left(f_{ft}\right)$$

Here, f_{ut}, f_{pt}, f_{ft} represent the fault event with responding unknown, priority, and flooding traffic accordingly, and $P(f_{ut})$, $P(f_{pt})$, $P\left(f_{ft}\right)$ presents the probability of obtaining corresponding traffic within the calculated period. During the arbitrary or unstable

condition, if any of these three types of traffic is generated, then the probability of faultiness is equal to the probability of obtaining responding traffic. Hence,

$$P(f_p) = P(f_t)$$

4.5. Explanation of the Simulation Process

The performance of the proposed protocol was evaluated with the OMNET++ simulator by evolving a simulation scenario. We selected several performance metrics to compare the protocol's results with that of other protocols, including CoAP, MQTT, and DDS.

4.5.1. Simulation Scenario

Regarding the simulation environment, a sample network topology with 19 nodes, including 12 end IoT nodes, 4 sub-server nodes, 2 edge server nodes, and 1 remote server, was created. Table 2 shows the chosen simulation parameters.

Table 2. Chosen values for different simulation parameters.

Parameter Name	Value
Number of nodes	19
Packet count	5551
Link layer protocol	Address Resolution Protocol (ARP)
Simulation time	100 s
Module used	Simple and Compound
Framework	INET
Power modules	INET Energy modules
Mobility	Static
Link	Eth100M

Using this example scenario, the simulation process was conducted 12 times for 100 s to achieve a 95% confidence interval. With the simulation results, node(s) may not act as faulty, leading to the generation of results without considering faults. Performance metrics were defined using appropriate parameters to extract and record results.

4.5.2. Simulation Results

Figure 4 shows the average end-to-end delay and mean end-to-end delay. The graph plots 5551 packets for 100 s, and these generated packets travel a maximum of five hops to arrive at the destination node. For the first 30 s, the graph shows an unstable end-to-end delay line, which is due to the uneven traffic overhead, but subsequently, it becomes more stable. The mean value of end-to-end delay is recorded as 0.01099 s, which is equal to 10.99 ms. Note: the result of end-to-end delay does not consider propagation delay as the delay is very insignificant.

Figure 5 presents the proportion of power consumption by each end node over time. It is clearly observable that the plotted line shows mostly steady state conditions, although there is some instability due to the initial data processing on each node for the first 30 s. The mean value of the proportion of power consumption is noted as 0.00894 of 100 J (energy storage module of INET used in each end node). In other words, the energy consumption rate is 0.894 J/s.

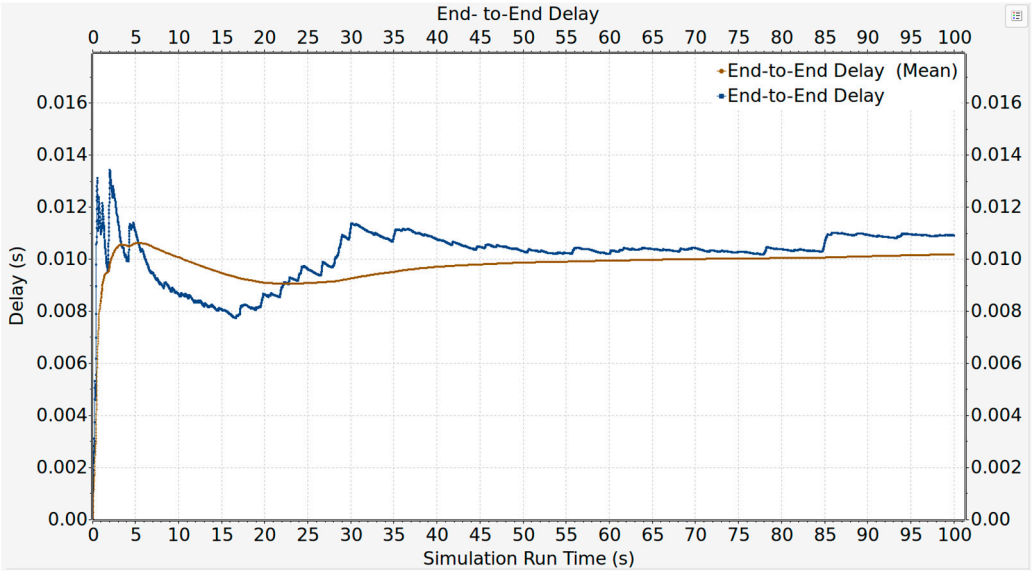

Figure 4. Graph of end-to-end-delay and end-to-end delay (mean).

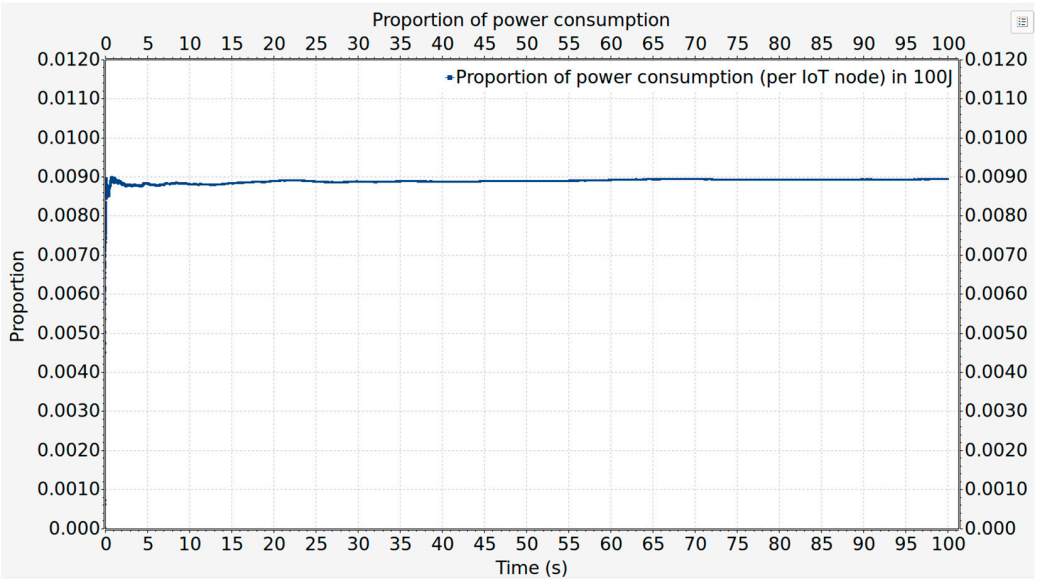

Figure 5. Graph of proportion of power consumption per IoT node.

Figure 6 illustrates the average throughput per end node over time. It can clearly be observed that the average throughput is noted as a non-steady state condition for the first few seconds, which is due to the uneven packet volume and processing time.

The simulation results show that the mean throughput value for the end nodes is 444,080.0 bytes per second, which is equivalent to 3.553 Mbps. Figure 5 shows the plotted values for the 5551 packets transmitted across the different end nodes for 100 s. Here, the parameter of the probability of faultiness is taken randomly between 0.0 and 0.9, where a lower value corresponds to a lower likelihood of being faulty and a higher value corresponds to a higher probability of being faulty. Because of a node failure, the model generates some corresponding traffic to mitigate the faults, which causes the simulated throughput results to be uneven.

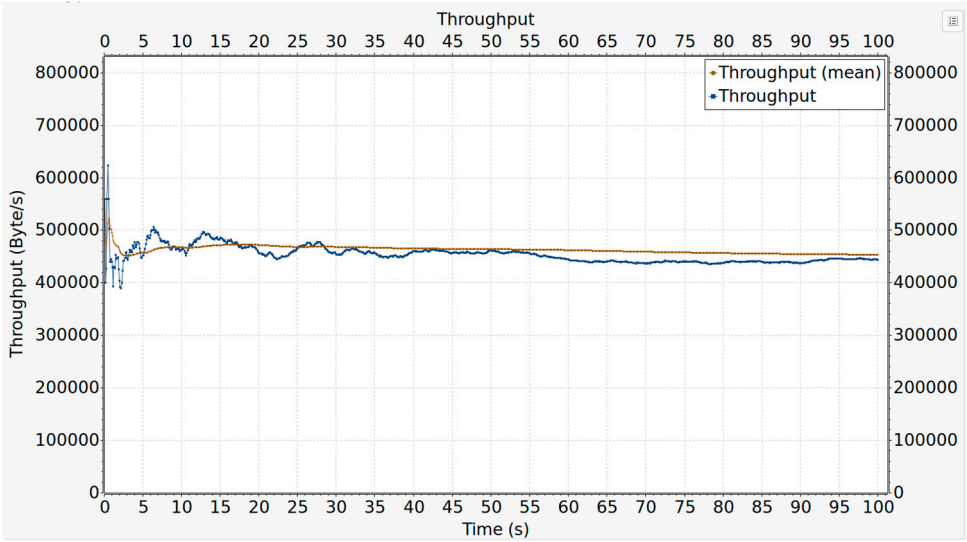

Figure 6. Graph of throughput and mean throughput.

4.6. Comparison and Evaluation of the Findings

Our comparison of the performance metrics of the ELSGP with other competing protocols, including CoAP, MQTT, and DDS, were based on our simulation results and the performance results reported in several sources in the literature [21,52,53]; the limitations of the performance evaluation are also presented in this section.

4.6.1. Comparison and Evaluation

To measure the performance of the comparable protocols, throughput, end-to-end delay, and power consumption were considered valuable metrics. Figure 7 compares the measured throughput of ELSGP against the other protocols. This comparison only shows the achieved throughput for the IoT nodes, the average of which was 3.553 Mbps per node, whereas CoAP, DDS, and MQTT show averages of 2.02, 1.62, and 2.38 Mbps, respectively. In other words, the throughput of the IoT node with ELSGP shows a value that is 75.89% higher than that of CoAP, 49.28% higher than that of MQTT, and more than double of the throughput of DDS.

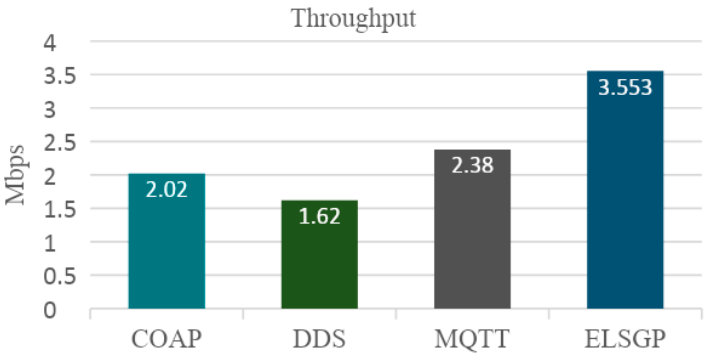

Figure 7. Throughput comparison.

Figure 8 shows a comparison among the protocols in terms of end-to-end delay. The graph shows that the lowest delay is for ELSGP at 10.99 ms. The main objective of distributed computation is to enhance the computing capabilities of edge layer nodes to achieve high performance for real-time operations. As shown in Figure 8, ELSGP achieves minimum delay within the existing lightweight protocols.

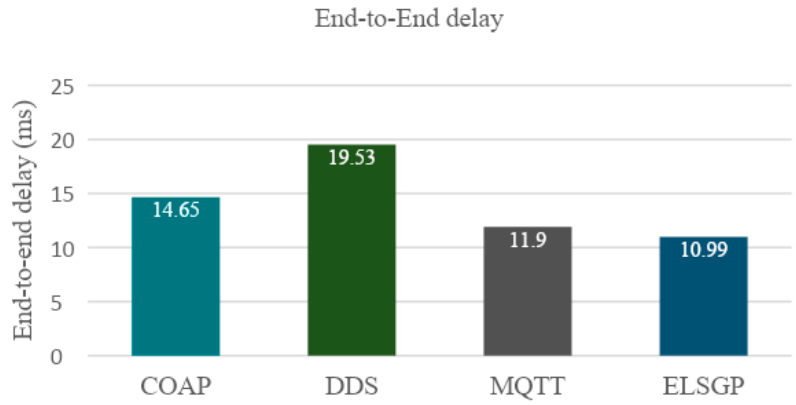

Figure 8. Comparison of end-to-end delay.

Obviously, low traffic overhead reduces energy consumption, and it gives leverage to utilize the energy for power-constrained devices. Figure 9 illustrates that the average power consumption rate per IoT node using ELSGP is 0.894 J/s, which is more than two times and more than five times less in comparison to the rate of CoAP and DDS, respectively. The results also show that the power consumption rate using ELSGP is 10.6% lower than the rate using MQTT.

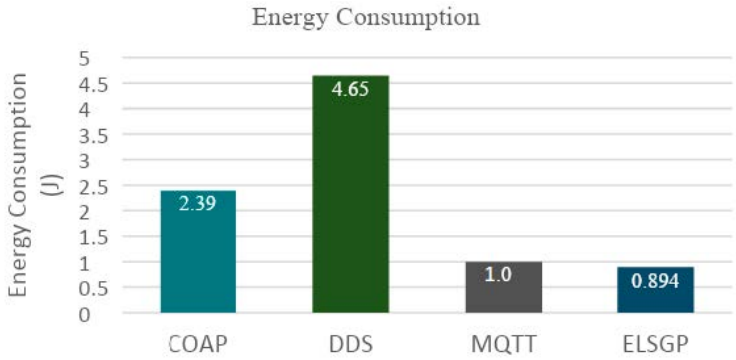

Figure 9. Comparison of energy consumption.

To sum up, based on the observed performance comparisons, ELSGP shows robust performance in terms of throughput, end-to-end delay, and energy consumption.

4.6.2. Limitations of Performance Evaluation

The proposed protocol was designed based on the aforementioned developed distributed computational model, whereas the comparative protocols were developed for a traditional edge layer architecture. Hence, the comparison results may not truly reflect the performance of each protocol, and the observed results may be higher or lower than the actual results.

We have presented three different types of faultiness. In this study, the nature of faultiness was considered to be as presented in Section 4.3. However, a fault can only be defined with a set of predetermined random probabilistic values.

5. Conclusions

With the rapid expansion of IoT and IoT-dependent systems, risks associated with security breaches, vulnerabilities, and threats are also rising. Due to constrained resources such as memory, processing unit, energy source, and bandwidth, IoT devices have limited capabilities to perform security operations like authentication, encryption, incident response, and other critical operations. This open research problem highlights the significance of adopting a lightweight security protocol based on a distributed computational architecture that can function within the limited capabilities of IoT devices.

5.1. Summary of the Contributions

This research article proposed an enhanced lightweight security gateway protocol based on a developed distributed computational model. A summary of the contributions of this study is provided below.

By scrutinizing the characteristics of IoT devices and system requirements, we developed a distributed computational model for the edge layer (described in Sections 3.2.3, 3.2.4 and 4.1). This model introduces a new type of node called a sub-server, which functions as an edge server but does not substitute it. The purpose of a sub-server is to distribute the computation power to perform the functions of the edge devices more efficiently and enhance the security operations. Along with the sub-server, the end nodes, edge server, and remote server are also members of this hierarchical model.

The proposed protocol has six operational features: access token distribution and validation, authentication and dynamic interoperability, attribute-based access control, traffic filtering, secure tunneling, and dynamic load distribution and balancing. The access token distribution and validation features demonstrate how different levels of identifications and validations are required to establish a trust relationship between nodes. Authentication enables the securely establishment of connectivity and allows for the transmission of traffic between two nodes. In a heterogeneous system, devices are operated with more than one communication protocol and operational standard, and the interoperability function enables compatibility with various end device configurations (including legacy solutions) (as discussed in Sections 4.3.1 and 4.3.2). The ABAC operations are also a part of authentication, which enables (through the use of attribute tags) the authentication of the device access request for joining onto a network, communicating with the other network devices, validating the generated or received data, and executing and validating the control command (as discussed in Section 3.3.3). The traffic filtering function facilitates the deployment of traffic filtering policies to reduce the volume of unwanted traffic (as shown in Section 3.3.4). Among the features of ELSGP, a secure tunnel enables the establishment of a logical path between the sending and receiving ends. The ELSGP consists of three different states: the full encrypted state, the lightweight state, and the policy-defined state (as presented in Section 3.3.5). Lastly, a dynamic load distribution and balancing function was developed with two different types of parameters—load information and load status parameters—which allow for the balancing and distribution of loads within policy-defined sub-servers and edge servers (as described in Section 3.3.6).

Considering the variability of system requirements, ELSGP adopts a policy framework that can define the protocol features according to the requirements, which also includes microservice deployment and software-defined operations. This study also showed fault mitigation mechanisms for Byzantine, cascading, and transient faults (as noted in Sections 4.3 and 4.4).

The evaluation and comparison results show that ELSGP shows enhanced performance in terms of throughput, end-to-end delay, and power consumption. In a nutshell, by studying the variability of system requirements, operational performance, and fault resiliency, it can be said that ELSGP could be a viable solution for the IoT edge layer.

Our proposed protocol is very adaptable; thus, it can be used to improve performance in time-critical real-world scenarios in the manufacture of plants, the processing of plants, industrial inspection and vision processing, highly IoT-intensified warehouses, healthcare (surgical centers), and anywhere where real-time data processing, local decision-making, and enhanced security for edge environments would be required.

5.2. Future Scope

This research article was the first step to develop a lightweight protocol under a distributed computational model. In future work, we will focus on investigating a number of open research problems. The notable future research directions can be summarized as follows:

We would like to further investigate the performance of the proposed protocol. For this study, we evaluated its performance under different conditions. Still, further analysis could be performed by re-evaluating the protocol performance via comparisons with other relevant metrics and, if required, identifying and rectifying the required modifications of the protocol design. Performance may vary with different network sizes, mobility patterns, and system requirements. We simulated and evaluated the proposed protocol's performance under predetermined fault conditions and a known environment, but the findings may vary under unknown conditions. The mobility of the operating devices and the characteristics of the communication channels may also have an impact on the performance. Our future investigations regarding performance analysis will consider these conditions.

The development of a high-performance and cost-effective sub-server, such as the one that was implemented into our proposed protocol (e.g., a sub-server adapted from a Raspberry pi 4, an Arduino board, or a newly designed device), is beyond the scope of this paper; however, this could be an interesting direction for future research articles to explore.

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

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Hassija, V.; Chamola, V.; Saxena, V.; Jain, D.; Goyal, P.; Sikdar, B. A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. *IEEE Access* **2019**, *7*, 82721–82743. [CrossRef]
2. Williams, R.; McMahon, E.; Samtani, S.; Patton, M.; Chen, H. Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach. In Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, China, 22–24 July 2017. [CrossRef]
3. Xu, H.; Yu, W.; Griffith, D.; Golmie, N. A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective. *IEEE Access* **2018**, *6*, 78238–78259. [CrossRef] [PubMed]
4. Kirupakar, J.; Shalinie, S.M. Situation Aware Intrusion Detection System Design for Industrial IoT Gateways. In Proceedings of the 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 21–23 February 2019.
5. Buchanan, W.J.; Li, S.; Asif, R. Lightweight cryptography methods. *J. Cyber Secur. Technol.* **2017**, *1*, 187–201. [CrossRef]
6. Celebi, H.B.; Pitarokoilis, A.; Skoglund, M. Low-Latency Communication with Computational Complexity Constraints. In Proceedings of the 2019 16th International Symposium on Wireless Communication Systems (ISWCS), Oulu, Finland, 27–30 August 2019. [CrossRef]
7. Iqbal, W.; Abbas, H.; Daneshmand, M.; Rauf, B.; Bangash, Y.A. An in-depth analysis of IoT security requirements, challenges, and their countermeasures via software-defined security. *IEEE Internet Things J.* **2020**, *7*, 10250–10276. [CrossRef]
8. Khan, M.N.; Rao, A.; Camtepe, S. Lightweight Cryptographic Protocols for IoT-Constrained Devices: A Survey. *IEEE Internet Things J.* **2021**, *8*, 4132–4156. [CrossRef]
9. Cherif, A.; Belkadi, M.; Sauveron, D. A Lightweight and Secure Data Collection Serverless Protocol Demonstrated in an Active RFIDs Scenario. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–27. [CrossRef]
10. Sha, K.; Yang, T.A.; Wei, W.; Davari, S. A survey of edge computing-based designs for IoT security. *Digit. Commun. Netw.* **2020**, *6*, 195–202. [CrossRef]
11. Wang, Y.; Tang, M.; Zhou, S.; Tan, G.; Zhang, Z.; Zhan, J. Performance Analysis of Heterogeneous Mobile Edge Computing Networks with Multi-core Server. In Proceedings of the 2020 IEEE 20th International Conference on Communication Technology (ICCT), Nanning, China, 28–31 October 2020. [CrossRef]
12. Minoli, D.; Sohraby, K.; Kouns, J. IoT security (IoTSec) considerations, requirements, and architectures. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017. [CrossRef]

13. Yang, H.; Alphones, A.; Zhong, W.-D.; Chen, C.; Xie, X. Learning-Based Energy-Efficient Resource Management by Heterogeneous RF/VLC for Ultra-Reliable Low-Latency Industrial IoT Networks. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5565–5576. [CrossRef]
14. Zhong, C.L.; Zhu, Z.; Huang, R.G. Study on the IOT Architecture and Gateway Technology. In Proceedings of the 2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), Guiyang, China, 18–24 August 2015. [CrossRef]
15. Jin, W.; Xu, R.; You, T.; Hong, Y.-G.; Kim, D. Secure Edge Computing Management Based on Independent μs Providers for Gateway-Centric IoT Networks. *IEEE Access* **2020**, *8*, 187975–187990. [CrossRef]
16. El Kaed, C.; Khan, I.; Berg, A.V.D.; Hossayni, H.; Saint-Marcel, C. SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways. *IEEE Trans. Ind. Inform.* **2018**, *14*, 715–724. [CrossRef]
17. Shah, T.; Venkatesan, S. Authentication of IoT Device and IoT Server Using Secure Vaults. In Proceedings of the 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018. [CrossRef]
18. Pardeshi, M.S.; Yuan, S.-M. SMAP Fog/Edge: A Secure Mutual Authentication Protocol for Fog/Edge. *IEEE Access* **2019**, *7*, 101327–101335. [CrossRef]
19. Daniel, D.A.; Roslin, S.E. Data validation and integrity verification for trust-based data aggregation protocol in WSN. *Microprocess. Microsyst.* **2021**, *80*, 103354. [CrossRef]
20. Zhou, L.; Su, C.; Yeh, K.-H. A Lightweight Cryptographic Protocol with Certificateless Signature for the Internet of Things. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–10. [CrossRef]
21. Rajeesh Kumar, N.V.; Mohan Kumar, P. Application of SDN for Secure Communication in IOT environment. *Comput. Commun.* **2020**, *151*, 60–65. [CrossRef]
22. Zhang, Y.; He, D.; Li, L.; Chen, B. A lightweight authentication and key agreement scheme for Internet of Drones. *Comput. Commun.* **2020**, *154*, 455–464. [CrossRef]
23. Wazid, M.; Das, A.K.; Kumar, N.; Vasilakos, A.V.; Rodrigues, J.J.P.C. Design and Analysis of Secure Lightweight Remote User Authentication and Key Agreement Scheme in Internet of Drones Deployment. *IEEE Internet Things J.* **2019**, *6*, 3572–3584. [CrossRef]
24. Singh, J.; Gimekar, A.; Venkatesan, S. An efficient lightweight authentication scheme for human-centered industrial Internet of Things. *Int. J. Commun. Syst.* **2019**, *36*, e4189. [CrossRef]
25. Zhu, L.; Yang, Z.; Li, M.; Liu, D. An Efficient Data Aggregation Protocol Concentrated on Data Integrity in Wireless Sensor Networks. *Int. J. Distrib. Sens. Netw.* **2013**, *9*, 256852. [CrossRef]
26. Razaque, A.; Rizvi, S.S. Secure data aggregation using access control and authentication for wireless sensor networks. *Comput. Secur.* **2017**, *70*, 532–545. [CrossRef]
27. Siddiqui, F.; Beley, J.; Zeadally, S.; Braught, G. Secure and lightweight communication in heterogeneous IoT environments. *Internet Things* **2021**, *14*, 100093. [CrossRef]
28. Chze, P.L.R.; Leong, K.S. A secure multi-hop routing for IoT communication. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Republic of Korea, 6–8 March 2014. [CrossRef]
29. De Azevedo, R.; Machado, G.R.; Goldschmidt, R.R.; Choren, R. A Reduced Network Traffic Method for IoT Data Clustering. *ACM Trans. Knowl. Discov. Data* **2021**, *15*, 1–23. [CrossRef]
30. Jiang, X.; Lora, M.; Chattopadhyay, S. An Experimental Analysis of Security Vulnerabilities in Industrial IoT Devices. *ACM Trans. Internet Technol.* **2020**, *20*, 1–24. [CrossRef]
31. Haddadi, H.; Christophidesy, V. SIOTOME: An Edge-ISP Collaborative Architecture for IoT Security. In Proceedings of the 1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec), Orlando, FL, USA, 17–20 April 2018.
32. Sachan, A.; Kumar, N.; Adwiteeya, A. Light Weighted Mutual Authentication and Dynamic Key Encryption for IoT Devices Applications. In Proceedings of the 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, India, 27–28 September 2019. [CrossRef]
33. Chen, L.; Xu, Y.; Lu, Z.; Wu, J.; Gai, K.; Hung, P.C.K.; Qiu, M. IOT μs deployment in edge-cloud hybrid environment using reinforcement learning. *IEEE Internet Things J.* **2021**, *8*, 12610–12622. [CrossRef]
34. Stévant, B.; Pazat, J.-L.; Blanc, A. QoS-aware autonomic adaptation of microservices placement on Edge Devices. In Proceedings of the 10th International Conference on Cloud Computing and Services Science, Prague, Czech Republic, 7–9 May 2020. [CrossRef]
35. Amaran, M.H.; Noh, N.A.M.; Rohmad, M.S.; Hashim, H. A comparison of lightweight communication protocols in robotic applications. *Procedia Comput. Sci.* **2015**, *76*, 400–405. [CrossRef]
36. Wolfart, D.; Assunção, W.K.G.; da Silva, I.F.; Domingos, D.C.P.; Schmeing, E.; Villaca, G.L.D.; Paza, D.D.N. Modernizing legacy systems with μs: A roadmap. In Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 21–23 June 2021. [CrossRef]
37. Morabito, R.; Beijar, N. A framework based on SDN and containers for dynamic service chains on IOT Gateways. In Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems, Los Angeles, CA, USA, 25 August 2017. [CrossRef]
38. Toshihiko, O. Lightweight Cryptography Applicable to Various IoT Devices. *NEC Tech. J.* **2017**, *12*, 67–71.
39. Derhamy, H.; Eliasson, J.; Delsing, J. IOT interoperability—On-demand and low latency transparent multiprotocol translator. *IEEE Internet Things J.* **2017**, *4*, 1754–1763. [CrossRef]

40. Intel. Policy Framework for the Internet of Things (IOT). Intel. 2014. Available online: https://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/policy-iot-framework.pdf (accessed on 4 January 2022).
41. Phung, P.H.; Truong, H.-L.; Yasoju, D.T. P4SINC—An execution policy framework for IOT services in the edge. In Proceedings of the 2017 IEEE International Congress on Internet of Things (ICIOT), Honolulu, HI, USA, 25–30 June 2017. [CrossRef]
42. Peng, C.; Chen, J.; Vijayakumar, P.; Kumar, N.; He, D. Efficient Distributed Decryption Scheme for IoT Gateway-based Applications. *ACM Trans. Internet Technol.* **2021**, *21*, 1–23. [CrossRef]
43. Lee, C.-H.; Wu, Z.-L.; Chiu, Y.-T.; Chen, V.-S. Heterogeneous industrial IOT integration for manufacturing production. In Proceedings of the 2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Taipei, Taiwan, 3–6 December 2019. [CrossRef]
44. Akasiadis, C.; Pitsilis, V.; Spyropoulos, C.D. A multi-protocol IOT platform based on open-source frameworks. *Sensors* **2019**, *19*, 4217. [CrossRef]
45. da Cruz, M.A.; Rodrigues, J.J.; Lorenz, P.; Solic, P.; Al-Muhtadi, J.; Albuquerque, V.H.C. A proposal for Bridging Application Layer Protocols to HTTP on IOT Solutions. *Future Gener. Comput. Syst.* **2019**, *97*, 145–152. [CrossRef]
46. Vijayma. Azure IoT Edge Task—Azure Pipelines. Azure Pipelines | Microsoft Docs. 2021. Available online: https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/build/azure-iot-edge?view=azure-devops (accessed on 4 January 2022).
47. Yarali, A. IOT: Platforms, Connectivity, Applications and Services. Amazon. 2018. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/secure-tunneling.html (accessed on 4 January 2022).
48. Zahoor, S.; Mir, R.N. Resource Management in pervasive internet of things: A survey. *J. King Saud Univ. Comput. Inf. Sci.* **2021**, *33*, 921–935. [CrossRef]
49. Castro, M.; Liskov, B. Practical byzantine fault tolerance. In *OSDI '99: Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, LA, USA, 22 February 1999*; USENIX Association: Berkeley, CA, USA, 1999; Volume 99, pp. 173–186.
50. Lee, Y.-L.; Arizky, S.N.; Chen, Y.-R.; Liang, D.; Wang, W.-J. High-availability computing platform with Sensor Fault Resilience. *Sensors* **2021**, *21*, 542. [CrossRef] [PubMed]
51. Xing, L. Cascading failures in internet of things: Review and Perspectives on Reliability and Resilience. *IEEE Internet Things J.* **2021**, *8*, 44–64. [CrossRef]
52. Guaman, Y.; Ninahualpa, G.; Salazar, G.; Guarda, T. Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry PI 3 in IEEE 802.11 Environments. In Proceedings of the 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), Seville, Spain, 24–27 June 2020. [CrossRef]
53. Bansal, M.; Priya. Performance comparison of MQTT and CoAP protocols in different simulation environments. In *Inventive Communication and Computational Technologies*; Lecture Notes in Networks and Systems; Springer: Berlin/Heidelberg, Germany, 2020; pp. 549–560. [CrossRef]