Design of an Enhanced Lightweight Security Gateway Protocol for the Edge Layer

A thesis submitted to
Auckland University of Technology
in partial fulfilment of requirements for the degree of
Master of Information Security and Digital Forensics

Supervisor

Professor Jairo Gutierrez

By

MD MASUM REZA

2022
School of Engineering, Computer and Mathematical Sciences

Table of Contents

Abstract

With the rapid expansion of Internet of Thing (IoT) dependency, the necessity of lightweight communication is also increasing due to its constrained capabilities. Focusing on the limited capabilities of the devices, this research presents the design of a novel lightweight weight protocol called Enhanced Lightweight Security Gateway Protocol (ELSGP) based on a developed distributed computation model of the IoT layer. This distributed computational model introduces a new type of node called a Sub-server to assist edge layer servers and IoT devices in computation and act as a primary gateway of dependent IoT nodes. This thesis 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 thesis also presents fault mitigation mechanisms, especially Trust and Priority Impact Relation for Byzantine, Cascading, and Transient faults. To validate the performances, we have simulated the protocol with an example scenario and extracted results for the several performance metrics. Based on the findings from the performance evaluation, further analysis of the protocol and future research directions are outlined. Undoubtedly, ELSGP could be a feasible solution for fulfilling the IoT-dependent system requirements.

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Masum Reza

Acknowledgements

First, I would like to express my heartiest thanks to the Almighty for bringing this work to light.

I wish to express my deepest gratitude to my supervisor Professor Jairo Gutierrez. His perpetual help, enthusiasm, and valuable suggestions helped in every stage to accomplish the work and prepare this report. I would like to thank Associate Professor Nurul Sarkar for his insightful guidance, and support during the coursework. I also like to convey my thanks to my AUT friends and friendly staff.

Last but not the least, I would like to admire my family members, especially my parents, my beloved wife, and my siblings, for their support during the crucial time and encouragement to undertake this work to finish.

List of Tables

List of Figures

List of Acronyms and Symbols

Acronym/Symbol	Meaning
6Lowpan	IPv6 over low power wireless personal area network
ABAC	Attribute Based Access Control
AMQP	Advanced Message Queuing Protocol
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DDS	Data Distribution Service
DSRM	Design Science Research Methodology
DTLS	Datagram Transport Layer Security
ELSGP	Enhanced Lightweight Security Gateway Protocol
GloMoSiM	Global Mobile Information Systems Simulation Library
GNU	GNU's not Unix
GPLv2	General Public License, version 2
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated development environment
IoD	Internet of Drones
IoT	Internet of Things
IIoT	Industrial IoT
IPsec	Internet protocol security
ISP	Internet Service Provider

JSON	JavaScript Object Notation
LoraWAN	Long Range Wide Area Network
LPWAN	Low powered wide area networks
MAC	Media Access Control
MDC	Mobile Data Collector
MQTT	Message Queuing Telemetry Transport
NetSim	Network Simulator
NS2	Network Simulator Version 2
NS3	Network Simulator Version 3
NED	Network Description
NFC	Near Field Communication
OLSR	Optimized Link State Routing
OMNET++	Objective Modular Network Testbed in C++
OPNET	Optimized Network Engineering Tools
OTcl	Object-oriented Tcl
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio Frequency Identification
SDG	Software-defined Gateway
SDN	Software-defined networks
SMAP	Secure Mutual Authentication Protocol
SMRP	Secure Multi-Hop Routing Protocol
SSH	Secure Shell
SSTP	Secure Socket Tunneling Protocol
RAM	Random Access Memory

TLS	Transport Layer Security
VPN	Virtual Private Network
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
UCID	User-Controllable Identification
XOR	Exclusive or
µs	Microservice
AT	Access Token
AT_D	Access Token Distribution function
A_{TAG}	Attribute tag
AT_V	Access Token Validation function
D_{ID}	Device Identifier
DD_{ID}	Dynamic Device Identification
cl	Current load at designated sub-server
CL	Current load at designated edge server
FT	Flooding Traffic
h	Current health check values
h_t	Health check threshold values
G	Group
IO	Interoperability
L	Load threshold value for edge server
l	Load threshold value for sub-server
ls	Available sub-server(s) for load sharing
N	Network

N_{ID}	Network Identifier
O	Object
PT	Priority Traffic
T	Regular Traffic
TQ	Current task queue value of the designated edge server
Tq	Current task queue value of the designated sub-server
TQ_t	Task queue threshold value of an edge server
tq_t	Task queue threshold value of a sub-server
UT	Unknown Traffic
Z	Zone
Z_{ID}	Zone Identifier
$\mathcal{N}$	Total number of nodes
$\mathcal{N}_{SS}$	Number of Sub-servers
$\mathcal{N}_{ES}$	Number of edge server
$\mathcal{N}_I$	Number of IoT devices
$I_{\mathcal{N}_{SS}}$	The numbers of adjacent IoT nodes with each Sub-server
S_S	The number of nodes including the sub-server itself
$\mathcal{E}_S$	The number of nodes including the edge server
$\mathcal{F}_p$	Frequency of generating $\mathcal{DDID}$ in a predefined period
$\mathcal{P}$	Per defined period
t	Every time slot duration
n_s	Number of time slots in a Per defined period
t_n	Duration of n^{th} slot

I_p	The average size of packets associated with the identity information
$\mathcal{T}_{OA}$	Traffic overhead
I_{pex}	Average packet size excluding DD_{ID}
$\mathcal{T}_{IO}$	The volume of traffic generated with the interoperability function
$\mathcal{T}_{ES}$	Traffic overhead due to full encrypted state
$\mathcal{T}_{LS}$	Traffic overhead due to lightweight state
$\mathcal{T}_{PS}$	Traffic overhead due to policy defined state
$\mathcal{T}_{ST}$	Traffic overhead due to secure tunneling
$\mathcal{T}_O$	Total traffic overhead of ELSGP
f	Number of fault nodes
$\mathcal{D}_t$	Degree of trust
P_i	Priority impact factor
p	The power of degree of trust a node
n	Number of nodes
$\mathcal{F}_{Tstart}$	Fault start time
$\mathcal{F}_{Tend}$	Fault end time
$\mathcal{F}_{\Delta T}$	Duration of fault
f_t	Fault event with responding traffic
f_{ut}	Fault event with responding unknown traffic
f_{pt}	Fault event with responding priority traffic
f_{ft}	Fault event with responding flooding traffic
f_{Φ}	Fault event within a period

Glossary of Terms and Definitions

Actuator: An actuator is a device that performs the actions according to the control signal. An actuator needs a source of energy to be operated. In an electronic system, the actuators are controlled with an electric control signal and operated with electrical energy.

Attack surface: In a computing ecosystem, vulnerable nodes from where unauthorized access can perform to interrupt the operational activities or spy on the activities or steal the data from the system. The vulnerable nodes can be recognized as an attack surface.

Cryptography: To secure communication between two ends, encryption is required. The technique or the study of securing the communication with encryption is known as cryptography.

Data Integrity: Data Integrity means the accuracy and validity of sending, receiving or stored data without any modifications and changes.

Data Replication: Data replication is the duplication of data that can be authorized or unauthorized. Unauthorized data replication is a kind of attack which may cause spying on the system or data leakage to an unprotective or threatening environment.

Edge Devices: The devices are operated in an edge layer of architecture are called edge devices.

Edge Layer: Edge layer consists of edge layer IoT devices, emended hardware and software to perform the operations of the edge level.

Edge layer protocol: A protocol refers to guidelines or rules of communication called the communication protocol. The edge layer protocol refers to communication rules among the Edge layer devices.

Edge Server: An edge server is an edge layer device that is an entry or exit point of an edge network. The server act as an exchange point for the edge devices.

Embedded system: Embedded system consists of hardware and software to perform a specific operation or function independently or under a large operational system.

End Nodes: End nodes also refer to the end devices of a system.

Gateway protocol: A gateway protocol is designed to communicate between the devices of two different networks. For the edge layer communication, the gateway protocol defines the communication between the edge layer network and any other network.

IIoT: IIoT stands for Industrial internet of things. It is a subdivision of IoT.

IoT: Billions of devices all over the world that are connected Internet of things and performing dependently or independently to accomplish the operational tasks are known as the Internet of Things or, in short, IoT.

Latency: Latency is usually known as the delay. In a network, data requires time to reach the destination, which is called latency or round-trip delay.

Light Embedded devices: The devices containing hardware and software that are powered with low battery energy and made up for performing light operations.

Lightweight Protocol: lightweight protocols are designed for the data communication within the light operational devices considering limited computational power and limited energy sources of the devices.

DID: DID stands for device identity. UID indicates the identity of a device based on the attributes and performing operations.

Vulnerabilities: vulnerabilities are also known as the weaknesses of a system by which unauthorized access can perform to attack or spy on the system operations.

WSN: WSN stand for wireless sensor network. WSN is a network of several sensing devices. The main objective of the WSN is to sense and monitor the physical environment.

Chapter 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 (Hassija et al., 2019). Numerous research has been conducted to secure the IoT environment, but new vulnerabilities have been found, making it challenging to secure the entire IoT environment (Williams et al., 2017). Industrial IoT (IIoT) is one of the promising domains where sensors and actuators perform sophisticated tasks for automation and job efficiency (Xu et al., 2018). 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 (Kirupakar & Shalinie, 2019). With the development of embedded systems and intelligent technology, the devices have been developed, but still, the devices have constraints in terms of memory, computing power, and energy (Buchanan et al., 2017; Celebi et al., 2019; Iqbal et al., 2020; Khan et al., 2021). Lightweight protocols aim to fulfill the system and security requirements considering the resource constraints. Minimizing CPU usage, consuming low power (especially for battery-powered devices), implementing light overhead, and reducing memory (RAM) usage are the essential requirements (Cherif et al., 2019). However, implementing security for these light embedded devices is become challenging due to the resource constraints because the security operations, including cryptographic operations, authentication, and critical management, increase resource usage (Buchanan et al., 2017). This research 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 (Sha et al., 2020). Due to their wide acceptability and low

latency communication characteristics, many researchers are interested in deploying edge servers (Wang et al., 2020). Edge computing may fulfill the low latency requirement, but security is still a big concern for IoT communication due to the heterogeneity and physically unprotected environment (Hassija et al., 2019). Numerous security threats like stealing identity and generating or injecting false or malicious data can cause disastrous outcomes in the entire communication infrastructure (Minoli, Sohraby, & amp; Kouns, 2017). Therefore, a lightweight threat protection mechanism is crucial for the edge layer. Limited research works have been found where IIoT edge servers have been considered for device management, enhanced security, and lightweight communication schemes for low latency. Therefore, it is easily predicted that a developed protocol based on these requirements can significantly enhance the proficiency of IIoT edge layer functions.

From that inspiration, we propose an enhanced lightweight security getaway protocol expected to enhance the edge layer latency and security performance. In this research work, we have also defined a new terminology called Sub-server. The main objective of the Sub-server is to assist the edge devices for computing. The proposed Sub-server is not the replacement of edge servers but assists in distributing the computation power that can be used for security and individual operations. To perform and evaluate the designed protocol and the proposed model, we have mainly focused on a highly populated IIoT environment where heterogeneous devices perform to accomplish various operational tasks.

The objectives and contributions of the proposed ELSGP are:

1. Reduce attack surface: Edge IoT devices are commonly not continuously operated in a secure environment (Hassija et al., 2019). IoT devices perform various patterns of tasks, which exhibits heterogeneous nature in the performance of these devices (Suman et al., 2019). For example, sensing, computing, actuating, filtering, aggregating, transferring,

analyzing, reading, and so many operational heterogeneities are found in an IoT environment which makes a system more complex and vulnerable to security breaches (Gao et al., 2020; Shakdher et al., 2019). Even though it is not good enough to protect the devices physically, it can be possible to enhance the device security logically (Gebremichael et al., 2019). In our proposed protocol, IoT devices will function according to the defined device identity and categorization based on attributes where the device will perform only a set of permitted tasks.

2. Edge server based IoT network: IoT devices have limited computing power. If the devices are interconnected to each other, there is a chance to propagate malware or infect peer nodes with malicious data, even replicated virtual (stolen identity) devices also can transmit malicious data (Hassija et al., 2019). From that motive, a centralized edge server-based edge network is required where all the traffic generated from the IoT devices is controlled by an edge server (Wang et al., 2020).

3. Secure channel: A short-range communication medium is required to establish communication between Edge nodes and servers (Ma et al., 2020). Numerous wired or wireless mediums can be implemented for this communication. The primary purpose of these secure channel techniques is to securely transfer data from one node to another node without data leakage, replication, or injecting anonymous data (Shakdher et al., 2019). We will discuss some secure channel techniques in the related work section. In the proposed protocol, the objective of the secure channel techniques is to enhance the security operational performance considering real-time and less computing communication.

4. Time-sensitive communication: Real-time communication is required for time-sensitive communication. In real-world industrial applications, where critical data communication is required. To accomplish this requirement, a server near the edge and a lightweight protocol are essential (Cherif et al., 2019).

5. Traffic filtering: Data generated from various IoT devices can have repetition (Gao & Ran, 2019). System bandwidth and processing power will be misused with this repeated traffic which can be the cause of overwhelming the system resources with redundant traffic (Gao & Ran, 2019). In this proposed enhanced lightweight protocol, we use a traffic summarization method to reduce the traffic, which will be explained in chapter 4.

6. Distributed computational model: we will present distributed computational model to assist end nodes and the corresponding edge server on computation.

In summary, we are presenting the literature review in the following chapter 2. Chapter 3 illustrates the adopted methodology with the explanation of selecting the research framework, research design, simulation process, and validation of the findings. In chapter 4, We will present the ELSGP with the designed features based on a developed distributed computational model. Chapter 5 presents the analysis and performance evaluation of the designed protocol with appropriate fault analysis, probability analysis, and comparison of the performance metrics with other relevant protocols. Lastly, the last chapter presents the conclusion and the future directions of this thesis.

Chapter 2. Background and Related Work

With the technological development during the last couple of decades, the term IoT has been revealing new opportunities, which makes it more acceptable. Industrial IoT (IIoT) is a subset of IoT (Kirupakar & Shalinie, 2019), and it is expanding so fast to reduce operational costs, make the industrial processes agile and seamless, and enhance controlling and monitoring (Yang et al., 2020). In a way, this IIoT ecosystem boosts industrial productivity, but on the other way, the attack surface also increases, which is rising security vulnerabilities (Yang et al., 2020). To secure the IIoT ecosystem from anonymous threats and attacks, researchers have developed various specialized security models. This section presents a brief overview of the IIoT system architecture-related terminologies, and the edge IoT-related security research is presented in the related work section 2.4.

2.1 IoT edge architecture

The architecture in the IoT edge layer is equipped with numerous devices and protocols (Wang et al., 2020). The IoT Edge layer architecture is elaborated as below.

2.1.1 IoT devices

IoT devices work as the end nodes, and these depending on their function, work as sensors, actuators, repeaters, integrators, filters, data transmitters, or intelligent devices.

2.1.2 Gateway

A border gateway is required to communicate between the Edge IoT Network and another external network (Zhong et al., 2015). To secure the communication between an external network and the internal edge IoT network, the border gateway applies various traffic filtering rules (Jin et al., 2020). 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 (Kaed et al., 2018).

2.1.3 Edge servers

The edge server works as a central hub of an IoT edge network (Wang et al., 2020) in many applications where time-sensitive communication is required. To fulfill those requirements, the edge server is required to install at the edge of the network. IoT devices transfer data to the network server, and the server process the received data on behalf of the devices as the devices do not have adequate computing power (Davari, 2020). The devices can communicate with each other, but that communication process may have security breaches (Williams et al., 2017). 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) (Zhong et al., 2015). Not like traditional servers, these edge servers should be handy, easy to maintain, and cost-effective (Sha et al., 2020).

2.1.4 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 also like an edge server but not a substitute for the edge server. 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 very locally on the edge devices to reduce the physical and operational distance between an edge server and an end device.

Significance of Sub-Server: In a highly dense (with IoT devices) operational environment like IIoT where edge servers are connected to numerous end devices and work

as end device gateways, the sub-server can be installed there for the below operational benefits:

- Enhance security: Stronger security operations can be implemented on the edge.
- Ensuring high availability: 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 the measure is to reduce the operational tasks of the associated edge server
- Distributional operations: Supports co-functional activities including microservices, adopting SDN functions, protocol translation, transceiving 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 the edge servers are costly, the implementation of sub-servers may reduce the number of operational edge servers (based on the system requirement) which may reduce the implementation cost.

2.2 Communication technologies

End devices can be connected through wired or wireless. In logistic IIoT, a low-range communication medium is needed (Khan et al., 2020). In addition, some other criteria need to be fulfilled, for example, low-powered, secured, ensuring accurate data transfer rates, and the right frequency for the IoT network (Di Pascale et al., 2018). Different types of

communication technologies are found in IoT communications, such as IPv6 over low-power wireless personal area network (6Lowpan), Near Field Communication (NFC), Bluetooth low energy (BLE), RFID, Sigfox, Z wave, Wi-Fi, and Long Range Wide Area Network (LoraWAN) (Lounis & Zulkernine, 2020; Figueroa-Lorenzo et al., 2021).

2.3 Common Challenges for IIoT systems

Security: IIoT Applications are significantly expanding. Due to the wide range of IoT applications, it has become challenging to manage the resources safely and security (Hassija et al., 2019). IIoT devices are not exempt from vulnerabilities and security threats. The chances of security threats like identity stolen, malicious data injection, data leakage, and so on are increasing (Khan et al., 2021). Attackers can get more opportunities to access the system if the number of unprotected devices increases (Khan et al., 2021).

2.3.1 Latency

In a time-critical environment, a low latency application is required. However, the IIoT communication system always faces latency barriers (Hiller et Al., 2018). For example, errors in timestamps, counter fluctuations, responses to the production errors, Synchronization delays, Human-Machine interactions, and so on, where low latency communication is crucial (Celebi et al., 2019).

2.3.2 Low computing power

To comply with all the requirements, the systems are commonly equipped with low-powered embedded devices (Celebi et al., 2019). In many circumstances, the devices are designed with a low powered Lithium battery (Yang et al., 2020). If the application applies advanced encryption techniques and high processing algorithms, it will exhaust the device battery so fast, which is not productive or economical (Buchanan et al., 2017).

2.3.3 Privacy and data protection

Privacy and data protection are also security challenges of IoT communication systems (Jiang et al., 2020). Numerous devices sense and collect data from the environment and perform automated tasks (Yu & Guo, 2019). During this data collection and command execution process, there is a chance of data violation and leakage (Yu & Guo, 2019).

2.3.4 Data filtering

Data generated from various devices of a Wireless Sensor Network (WSN) is required in any of the operations, but it is not necessary to analyze and store all the data generated by these devices (Sofwan et al., 2018). It will consume a large portion of the network resources as well as not be economically effective (Buchanan et al., 2017). To reduce network resource usage, data from the devices need to be efficiently filtered before processing and transmitting it to the cloud or remote server (Sofwan et al., 2018).

2.3.5 Lightweight security protocol

There is a dilemma between security operations and CPU usage (Khan et al., 2021). For example, if advanced encryption applies to transceive data of a sensor, it will consume a more significant portion of the sensor's processing power as well as the battery power (Yang et al., 2020). On the other hand, if the data generated from the sensor is not protected, it will increase the attack surface (Khan et al., 2021).

In recent years, researchers have shown interest in lightweight security protocols that consider various challenges of IoT communication systems. A lightweight security protocol aims to fulfill a system's security requirements and operate under the constraints of low processing power networks and devices (Zhou et al., 2019).

2.4 Related work

Haddadi et al. (2018) proposed a cooperative system called SIOTOME and placed it between the home network gateway and its Internet service provider (ISP) to provide a security solution for IoT security attacks. A machine learning technique was also used to trace traffic to provide privacy-aware security service and isolate the IoT devices from security attacks. This isolation technique can be incorporated into lightweight security schemes.

Chze et at. (2014) proposed a Secure Multi-Hop Routing Protocol (SMRP) for forming a new IoT device network or joining an existing network. In this protocol, several parameters are used, including unique User-Controllable Identification (UCID), the user permitted applications, and user permitted IoT devices. These three unique parameters were used to accomplish light overhead. The protocol is also compared with the well-known Optimized Link State Routing (OLSR) protocol which shows better performance in comparison with the round-trip delay and required bandwidth.

To identify cyber-attacks, an intelligent architectural paradigm is designed (Kirupakar & Shalinie, 2019). Here, a low-power IIoT edge gateway is considered for the architectural paradigm. In this paradigm, a testbed result was presented where the gateway node used a shallow footprint as the authors presented the gateway device as a constraint device. In addition, an algorithm has been developed for randomized cyber-attack mitigation for edge IoT devices.

A secured edge computing mechanism is proposed to facilitate Microservice (μs) for heterogeneous devices in industrial domains (Jin et al., 2020). The experimental result of this research 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.

(Shah & Venkatesan, 2018) 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 have been 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 (Pardeshi & Yuan, 2019). 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.

(Daniel D & Roslin, 2021) presented a data aggregation protocol for a WSN to verify and validate the sensed data. In this protocol, data is encrypted, segmented then signed with a homomorphic MAC tag before forwarding. A comparison of overhead size packet drop rate, energy consumption, and transmission delay among the protocol, the access control and authentication (SDAACA,) and Efficient Integrity-Preserving DA Protocol (EIPDAP) where the protocol showed enhanced performance.

Zhou et al. (2019) proposed a lightweight cryptographic protocol for IoT-based applications where the author presented 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 developed with Raspberry PI 3 Model B. The performance evaluation of this cryptographic protocol shows that the total computational cost of time is improved.

Cherif et al. (2019) proposed a lightweight protocol for a serverless system to collect data from Mobile Data Collectors (MDCs) and transfer it to a trusted third party. Here, time-sensitive communication is 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, which shows that the protocol can fulfill the security requirements, but no information about the credibility (like, Confidence Interval) was found.

Jiang et al. (2020) presented a couple of distinct security vulnerabilities for IIoT devices in the industry 4.0 ecosystem. A testbed system was developed to create an attack scenario to take unauthorized control over the actuators and the platforms to perform hazardous operations. From this demo environment, several variations of an attack (called the Mirai attack) have been explored. However, in this experimental analysis, most of the security risks considered in this testbed system were revealed by OWASP and Kaspersky Lab.

Azevedo et al. (2020) presented a reduced traffic method for a centralized IoT data clustering applied where reduced traffic is needed and in a low-power IoT system. The reduced traffic method applies when it is used in a limited computational power system. In the proposed method, a clustering algorithm was developed to summarize the data representation. The simulation result showed that in the six used datasets, the proportion of data reductions was from 39.59% to 91.22%.

To secure the communication between server and device, Advanced Encryption Standard Constrained Queuing Telemetry Transport Protocol (AESCQTT) has been developed (N.V. & P., 2020). This protocol is designed to block IoT network and application-level vulnerabilities. Constrained Application Protocol (CoAP) functions (over TLS) are applied here to minimize the communication overhead. A comparison table was presented among CoAP, Data Distribution Service (DDS), and AESCQTT, where 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.

Zhang et al. (2020) 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 (Wazid et al., 2019 and Singh et al., 2019) in terms of bandwidth (communication cost) and computation cost. The performance evaluation shows the AKA scheme has the lowest costs among these three schemes.

Daniel and Roslin (2021) proposed an aggregation tree-based protocol to aggregate data fragments by using data validation and integrity verification. This protocol is designed for wireless sensor networks for aggregation, verification, and synchronization of fragmented data blocks by using a homomorphic MAC tag. A better performance finding has been shown for this newly developed Data Validation and Integrity Verification for Trust-based Data Aggregation Protocol (DVIVTDAP) in comparison with the Efficient Integrity and Preserving Data Aggregation Protocol (EIPDAP) protocol (Zhu et al., 2013) and the Secured Data Aggregation using the Access Control and Authentication (SDAACA) protocol

(Razaque & Rizvi, 2017). Here, the performance of DVIVTDAP was evaluated mainly based on overhead size, detection delays, and energy consumption.

In this research (Siddiqui et al., 2019), the authors 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 shows the experimental results in terms of energy and latency, where 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 many extents.

Sachan et al. (2019) presented an enhanced dynamic cipher-based light-weight mutual authentication scheme that is tested in software environments which shows less CPU and power consumption in comparison with the state-of-the-art method. The hash function transfer protocol was used to authenticate the communication between the client and the server, where a dynamic key matrix is used in matrix coordinates to secure the keys transfer process.

Gaurav et al. (2020) reviewed the recent literature on security mechanisms used in Low powered wide area networks (LPWAN). In this paper, the authors also surveyed several recent papers on vulnerabilities and possible attacks on constrained IoT device networks. The outcome of the overall literature review of this paper also exhibited the significance, opportunities, and challenges of software-defined networks (SDN) in LPWAN security.

Several potential recent research has been reviewed where it is identified that either security functions are compromised to reduce the operational tasks of the end devices (Chze et at.,2014; Azevedo et al., 2020), or the constraint of the end devices is not enough

considered to adopt the security functions (Jin et al., 2020; Shah & Venkatesan, 2018); Pardeshi & Yuan, 2019; Jiang et al., 2020). In some extant, responsiveness during time-sensitive communication is limitedly considered (Cherif et al., 2019; Haddadi et al. 2018; Kirupakar & Shalinie, 2019; Sachan et al., 2019)

By considering all the above requirements, It is also identified that limited research (Daniel D & Roslin, 2021; N.V. & P., 2020; Zhang et al., 2020) has been conducted to adopt flexible custom-defined policies according to the system requirements. Adopting improved services (Zhou et al., 2019; Daniel and Roslin, 2021; Haddadi et al., 2018) may produce additional traffic which may not relevant to the functionalities of the end devices. It shows the necessity of adopting improved traffic filtering techniques to prevent unwanted resource utilization.

Any of them is compromised 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 have proposed a distributed computational architecture with enhanced security protocol which is elaborated in chapter 4. The corresponding analysis and evaluation are presented in chapter 5

Chapter 3. Methodology

In chapter 2, we have discussed the existing literature to understand the works have done on security protocols for IoT devices, reveal the limitations of existing research, identify research arguments, and identify the research gap. In this research, we have proposed a novel solution to address the research problems discussed in chapter 2. The purpose of this chapter is to explain the research methodology adopted in this thesis.

This chapter is subdivided into five sections which include the preferred methodology, research design, network simulator selection, Omnet++ simulator, and validation of simulation results. In section 3.1, the adopted research methodology is presented where an appropriate framework is identified to outline the overall research contributions; the following section describes the method to address the problems and the direction to the validation. Section 3.3 describes the network simulator selection process and discusses the justifications for choosing an appropriate simulator IDE. Section 3.4 explains the architecture of the sleeted simulator and its engagement with this research, and the last section of this chapter describes the validation techniques to be used for this research.

3.1 The adopted methodology

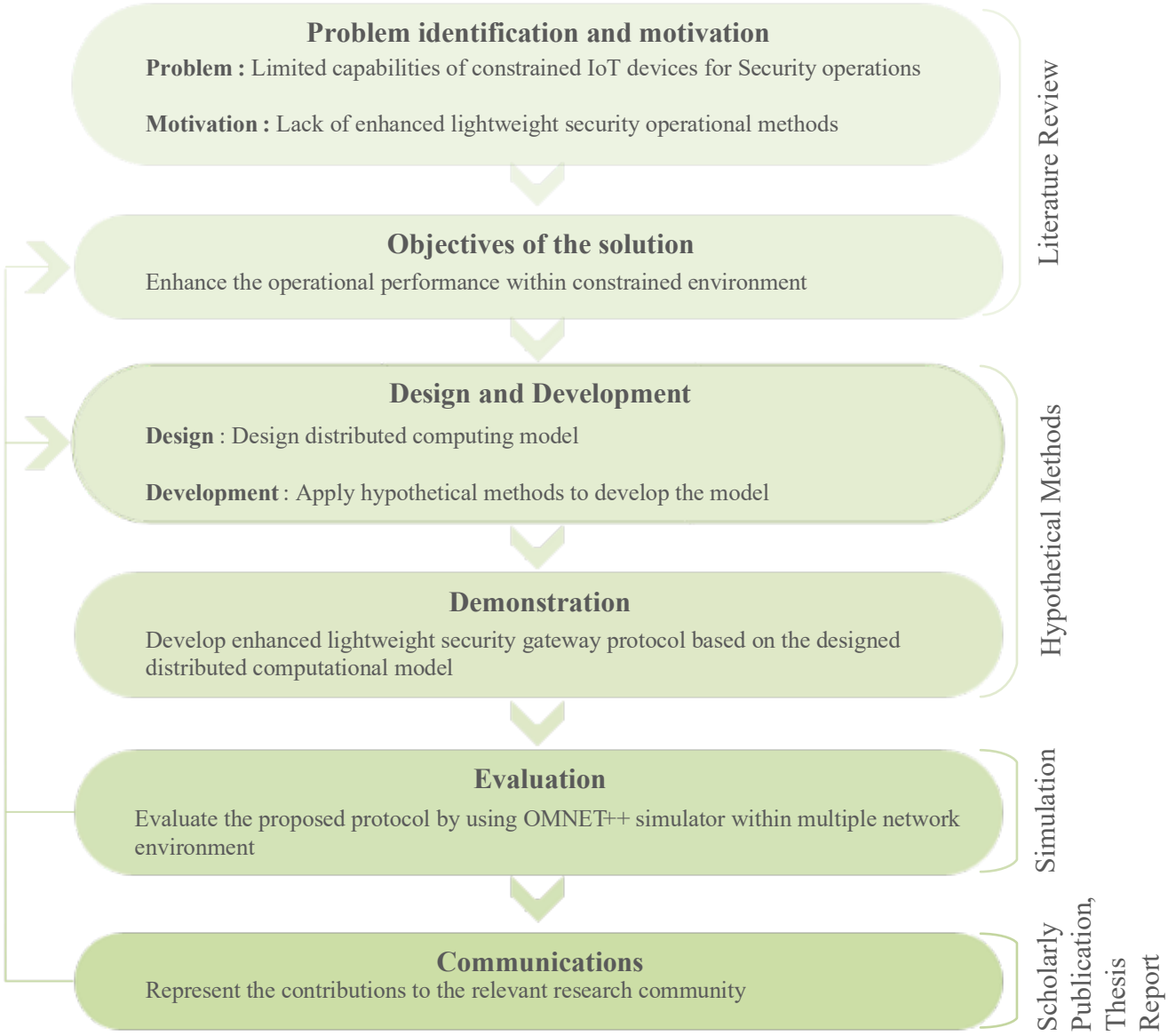
A research process indicates the comprehension of the research area with the appropriate research questions and an explanation of the motivations for the research, and an exploration of the proper design and development methods to address the perceived research questions (Nunamaker et al. 1990). Nunamaker et al. (1990) developed a well-established research framework that indicates a chain research process. The research process contributes to the research method and research domain knowledge. Therefore, using an appropriate research methodology enhances the research life cycle process by contributing knowledge to the research domain.

By identifying and analyzing the nature of research, a widely accepted and well-established research framework has to be adopted. This research has developed an enhanced lightweight security gateway protocol for IoT devices by considering the relevant research problems. In other words, this thesis aims to develop and design a solution to address the research problem by validating the design and comparing design performance with existing solutions. With this process in mind, this research intends to adopt a research framework called Design Science Research Methodology (DSRM).

DSRM is a commonly accepted framework that helps to legitimate and recognize research objects, processes, and outcomes (Peffers et al., 2007). DSRM indicates the process of gaining knowledge on an appropriate domain by undertaking highlighted studies during the literature review and by using methods of analyzing and evaluating the design science research objects like modeling, developing theories, hypotheses, proof of concepts, and validations (Peffers et al., 2007). Figure 3.1 shows how this research adopts the DSRM framework. The framework subdivides the methodology into six steps: problem identification and motivation, solution objectives, design and development, demonstration, evaluation, and communication (Peffers et al., 2007).

Figure 3. 1

Design Science Research Methodology



3.1.1 Problem identification and motivation

The research problem is identified as the limited capabilities of IoT devices to perform the operations. From the literature review, it has been determined that the functions such as authentication, secure handling of generated traffic, prioritizing traffic, and other operational tasks must be performed within the constrained nature of the IoT devices.

However, the existing solutions for this research problem still focus mainly on improving performance within the limited capabilities. The existing solutions like lightweight protocols, lightweight authentication, and low computing encryption techniques still emphasize on enhancing IoT device security and securing the whole network from security breaches, threats, and attacks.

The primary motivation of this research is to develop an enhanced lightweight security gateway protocol for the edge layer of the IoT network where all the requirements of these constraint devices should be considered. Furthermore, a distributed computational model associated with the designed lightweight protocol will be presented in this research aiming to establish secure and efficient communication. The comprehensive literature review shows the deficiency of communication protocols to fulfill an IoT network's operational requirements.

3.1.2 Objectives of the solution

The research should primarily focus on mitigating or, to some extent minimizing the identified problems. The research objectives are as follows:

- To develop a distributed computational model for IoT network
- To propose an enhanced lightweight security gateway protocol based on the developed distributed computational model
- To evaluate and analyze the performance of the developed protocol.

3.1.3 Design and Development

To contribute to the research gap in this domain, we will introduce a novel protocol that has been designed for constrained IoT devices and interconnected networks. The development has been conducted in three phases. Firstly, we proposed a distributed computational model to overcome the limited computational capabilities of constrained IoT

devices. Secondly, based on the designed model, we have developed a lightweight protocol to enhance the overall edge network performance and mitigate security breaches. Furthermore, we have formulated six algorithms that give comprehensive directions to the protocol functioning mechanisms. Finally, the protocol has been simulated with multiple network topologies, and we have analyzed the performance based on the generated simulation results.

3.1.4 Demonstration

In this research, developing algorithms, developing computation model, validation, and reconsidering the performance analysis results to design process are demonstrated as the solution artifacts of the identified problems. The knowledge identified research problem helps shape the initial modeling and the design process finalized through the simulation studies, the performance analysis, and the validation phase.

3.1.5 Evaluation

The performance of the designed model-based protocol has been measured with different metrics such as end-to-end communication delay, throughput, communication overhead, battery power consumption, and packet drop rate. The analysis stage of the thesis has been outlined hypothetically, including designing the model, designing attributes of the protocol, developing pseudocode, and logical analysis of the protocol. Secondly, the validation process of research has been conducted by simulating the hypothetical model and designing protocol attributes with a well-established simulation IDE to measure the protocol's effectiveness. Finally, the design is re-evaluated by focusing on the scope of performance improvement based on simulation results.

3.1.6 Communications

Communication is the way to represent the artifacts and novelty of this research to the scholars or relevant research community to get scholarly feedback for future improvement,

and it can be done by submitting this work to an appropriate high-impact factor journal and presenting it in a renowned technical conference. Moreover, this written master's thesis is another way to present these research contributions, which will also help get the recommendations and further improvement suggestions.

3.2 Research Design

The aim of the research design is to address the research questions (Kothari, 2004). In this section, we will illustrate how we have designed this research to address the gap in this field of study. A literature review is a way of understanding and acquiring knowledge about the relevant field, which is also employed to identify the research problems. Depending on the problem, the methodology, modeling, structuring of the solutions, analysis, and validation techniques have been outlined.

3.2.1 Research design

The performance of the developed solution to the identified problems may or may not work as predicted outcome or performance. There are several ways to verify the predicted outcome or performance. The developed solution needs to be tested through an apparatus, testbed, or a virtual simulation environment to evaluate the performance (Zelkowitz & Wallace, 1998).

We have studied the problems to develop this distributed computational model for IoT edge networks to enhance the network performance compared to the existing solutions. Based on the developed computational model, the lightweight security gateway protocol has been developed. We have elaborated this protocol with appropriate algorithms, mathematical representations, and appropriate validation and performance analysis techniques.

3.2.2 Simulation

Simulation is a widely accepted method used extensively in the telecommunication and computer networking research field (Fruth, 2011). This method facilitates replicating real systems in a virtual environment where a researcher can examine and experiment with new ideas within a cost-effective and time-efficient process (Beese et al., 2019).

A simulator is the virtual representation of a real complex system through which we have tested our proposed protocol. In the production environment, various sensors and actuators are integrated to perform various operations, which will create more complexity in analyzing the performance of the proposed protocol. We used the simulator to verify the predicted performance of the proposed protocol with the simulated outcome. During the event when the predicted performance does not work as the simulated outcome, we have modified the appropriate attributes of the proposed protocol to address the research problem.

3.3 Network simulator selection

Khan et al. (2012) provided a list of enriched simulation tools for telecommunication simulation studies. Numerous potential network simulators have been developed to analyze various networking events. For this research, we are interested in an open-source simulator due to the availability of the resources and the scope of designing the attributes of network devices and events. We found several simulators useful to fulfill this study's simulation requirements, like NS2, NS3, OMNET++, GloMoSiM, OPNET, and NetSim. This section presents a brief overview of these well-recognized simulators, a comparison among the simulators, and a justification for choosing the appropriate simulator for this study.

3.3.1 NS-2

Network Simulator 2, in short NS2, is a discrete event, an open-source simulator that is widely used for computer network simulation in the research community (Issariyakul &

Hossain, 2009). This simulator was written in C++programming language and developed in 1995 under the Virtual Inter Network Testbed project (Issariyakul & Hossain, 2009). As a credible simulator, NS-2 became accepted throughout the research community. With the OTcl scripts, a network topology can be described, and the simulation process is exhibited with a graphical interface along with the executed parameters and defined functions.

3.3.2 NS-3

Network simulator 3 (NS-3) is the developed version of NS-2, which is also a discrete event simulator primarily developed for research and educational purposes (Campanile, 2020). Like NS-2, NS-3 is also an open-source, free network simulator released under the GNU GPLv2 license (Campanile, 2020). In addition to C++, this simulator also supports the Python programming language (Dorathy & Chandrasekaran, 2018). With the dissimilarity with NS-2, this simulator requires C++ programming instead of OTcl scripts to describe the simulating network. In addition, the architecture of NS-3 also supports external resources like packet tracer analyzer tools which give more flexibility to use the simulator IDE (Dorathy & Chandrasekaran, 2018).

3.3.3 OMNET++

OMNET++ is an object-oriented modular discrete event simulator widely used as a network simulation framework (Varga, 2019). It is a comprehensive, modular, and component-based C++ simulation library with several features, including an extensive graphical interface, supporting extensions, model frameworks, and supporting integrated development (Varga, 2019). OMNET++ has been regularly improved; the latest version is 6.0, released on 13th April 2022 for Linux, Windows, and MAC OS (Dorathy & Chandrasekaran, 2018). OMNET++ is well-structured and supported by various frameworks such as INET Simu5G, Veins, and Castalia (Varga, 2019). Additionally, OMNET++ is

released under GNU General Public License in other works; it is free for educational and nonprofit use and easy to learn, giving wide acceptability within the research community (Zarrad & Alsmadi, 2017).

3.3.4 Other Simulators

Additional network simulators also well-known within the research community include GloMoSiM, OPNET, and NetSIM (Zarrad & Alsmadi, 2017). GloMoSiM is often chosen for the large-scale wireless network with heterogeneous links and devices (Dorathy & Chandrasekaran, 2018). NetSIM is generally used for simulating Cisco networking devices for learning purposes (Dorathy & Chandrasekaran, 2018), and OPNET is often used to model communication networks and distributed computing systems within the research community (Zarrad & Alsmadi, 2017).

3.3.5 Justifications for choosing OMNET++

Omnet++ is an advanced simulation IDE with numerous advantages over other network simulators. In this section, we compare OMNET++ with other well-established simulators and provide the justification for choosing OMNET++.

Table 3. 1

Comparison of Network simulators

Simulators	Language Used	License	Upgrades	GUI
NS2	C++, Otcl	Open source (Free)	Stopped	Not Available
NS3	C++, Python	Open source (Free)	Continue	Limited
OMNET++	C++	Public License (Free)	Continue	Available
GloMoSiM	PARSEC (c based)	Open source (Free)	Stopped	Limited
OPNET	C, C++	Commercial	Continue	Available
NetSIM	JAVA	Commercial	Continue	Available

Note. Sources (Zarrad & Alsmadi, 2017) and (Varga & Hornig, 2008)

OMNET++ is a widely used simulator due to its user-friendly architecture, powerful GUI, and compatibility with popular frameworks, which enhances the broader engagement

and acceptability of the research community and educational users. The INET framework has extensive modules for wired or wireless network infrastructure, which can be used to evaluate our proposed protocol. The OMNET++ simulation kernel and library functions are written in C++, which makes the simulator more convenient to use without learning a new programming language. Moreover, the simulator environment is very convenient for developing new prototypes of a device with simple modules by modifying the required parameters, which also influenced us to evaluate our developed distributed computational model with OMNET++. Furthermore, the library functions and predefined Classes reduce the effort and time that would be necessary to develop them from scratch. OMNET++ offers flexibility to define customized messages which are convenient for developing a new protocol.

Table 3. 2

Comparison of Network simulators in terms of performance

Simulators	Memory Use	CPU Usage	Computation time
NS2	Highest	Higher	Highest
NS3	Lowest	Higher	Lowest
OMNET++	Average	Lowest	Low
GloMoSiM	Average	Lowest	Low

Note. Source (Zarrad & Alsmadi, 2017).

Tables 3.1 and table 3.2 show comparisons of several network simulators such as NS2, NS3, OMNET++, GloMoSiM, OPNET, and NetSIM, where it has been recognized that OMNET++ has several benefits over the other simulators in terms of simulation language, License, Upgrades, GUI support and performance metrics.

To sum up, OMNET++ has excellent integrability, debugging capability, flexibility to use predefined functions and classes, customizable architecture, better user engagement, and

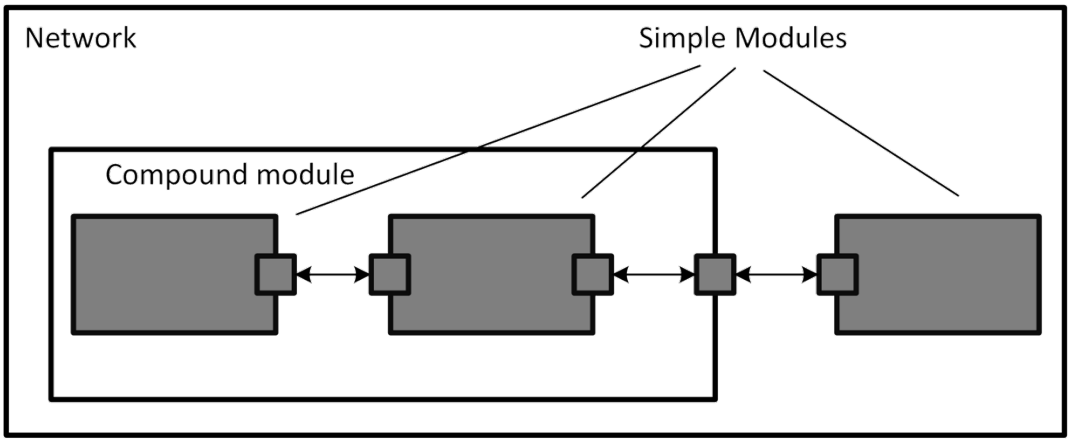
availability of resources making the simulator credible. Moreover, it shows reasonable performance over several identified performance metrics.

3.4 Omnet++ Simulator

A brief overview of the OMNET++ architecture, its associated frameworks, and engagement of the simulator with this thesis will be presented in this section. A network can be modeled in this simulator from a basic module called a Simple module, written in C++ (*OMNeT++ Simulation Manual* 2021). The composition of several Simple modules is called a Compound module. The network itself is a compound module.

Figure 3. 2

OMNET++ simple and compound modules



Note. Source (OMNeT++ Simulation Manual 2021)

OMNET++ has a unique language to describe a network topology called Network Description (NED) language. The modifications or changes of different parameters of the network can also be done from the associated .ned file (*OMNeT++ Simulation Manual* 2021). For the simulation process, some additional files are also required which include .msg, .cc, .h, and .ini files. The prototypes and the fields of a message type are defined with the .msg file, which is translated into full-fledged C++ classes (*OMNeT++ Simulation Manual*

2021). The simple modules, associated algorithms, and the process definitions are developed with a .cc/.h file (*OMNeT++ Simulation Manual* 2021). .ini file is needed to initialize the simulation where the parameters can be assigned. The files are developed in C++ with the related library functions and Classes.

We used OMNET++ IDE version 5.7, the latest version as of 2021 when this thesis simulation was conducted, and installed in a Linux distribution, Ubuntu version 20.04.1. The simple modules were developed with the core properties of the proposed protocol. To some extent, pre-developed INET framework modules were also used to analyze the protocol performance. In the simulation process, firstly, we developed the network with the distributed computational as defined in the research objective. Secondly, the protocol properties (algorithms) are developed in C++ language in three types of required files with .cc, .h, and .ini suffix. Finally, with the generated analysis file (in .anf format), we extracted results from generated grapes (in .svg file), events, and simulation summary information.

3.5 Validation of Simulation results

A simulation is a replicated view of a real system where different entities and characteristics are modeled to represent a real system at a satisfactory level (Sargent, 2015). The conceptual model developed in a simulation may not be accurate in comparison with the actual view (Sargent, 2015). Therefore, the results of the simulated conceptual model need to be validated so that it exhibits trustworthiness.

3.5.1 Simulation results

Primarily performance of the designed protocol is measured with the outcome of simulated results. The simulation results can be extracted, including generated results, graphs, and charts with different applied variables that have been used.

3.5.2 Credibility analysis

To analyze the performance of a hypothetical research design is an ideal step in analyzing the credibility of the research (Pawlikowski et al., 2002). Evaluating the correctness and efficiency and analyzing the credibility of the designed algorithm of the proposed protocol can be an effective way to verify the predicted performance.

A simulation shows only the performance of the network analysis, but it is also essential to demonstrate the accuracy of entire simulation studies. For example, the samples might be lost, and other unattended interruptions can occur during running the simulator (Sarkar & Gutierrez, 2014). In the statical error analysis, this information should be described. In a stochastic process, the samples can be generated and collected randomly to create an identical real environment, illustrate the sampling process, analyze data output, and use a Pseudo-random number generator (Pawlikowski et al., 2002). The statistical index, confidence interval indicates the probability of the actual mean value of a parameter (Pawlikowski et al., 2002).

To validate our simulation result, we considered the confidence interval and statistical error. Considering 95% confidence interval (as this interval is mostly used and accepted (Trafimow, 2018) and the statistical error elimination process, we calculated the parameters set, e.g., resultant variables, standard deviation, sample size, the sample mean, and several simulations run. Based on the findings, we have run the simulation, which projects that the results of our proposed protocol are within the acceptable level.

Chapter 4. Design of an enhanced lightweight security gateway protocol for the edge layer

This lightweight security gateway protocol aims to enhance security operation by ensuring operational latency preferences. To demonstrate the proposed ELSGP protocol, this chapter is divided into three sections. First, in section 4.1, we have explored the challenges of the existing edge server-based edge layer architecture. Then, to mitigate the explored challenges, the proposed IoT edge layer architectural model has been presented (in section 4.2). Finally, following the model, the ELSGP protocol has been presented (in section 4.3).

4.1 Challenges of the existing edge server-based system

In many research and previous IoT computational models, the edge layer servers are proposed to assist the constrained IoT devices. However, in this research, we have investigated the edge server operations where we discovered several challenges of edge-server based IoT edge layer communication systems. The challenges are:

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 requirement 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.

4.2 The proposed model

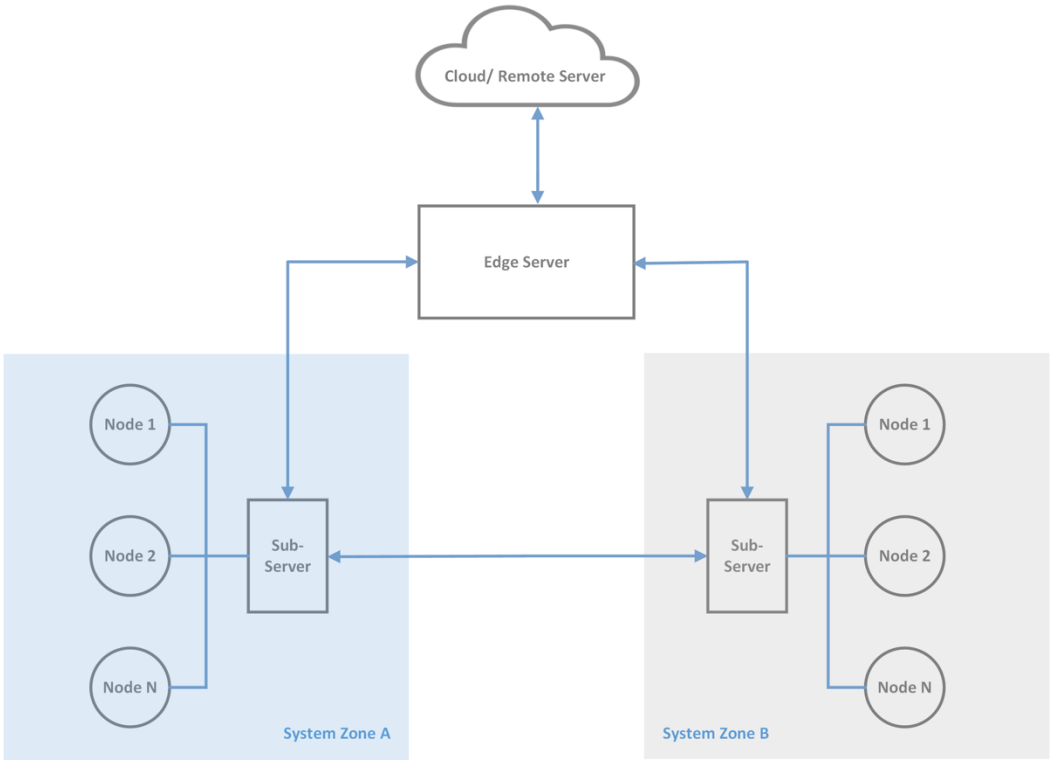
The proposed model is developed to make the edge server-based edge layer architecture more distributed in computation, security, and system operational tasks. In figure 4.1, the proposed edge layer architecture has been presented. The end nodes are considered as the IoT devices, which include edge IoT devices, operational sensors, and actuators.

In this architectural model, we have introduced a new network entity called Sub-server. A Sub-server is a server-like device that is not the replacement of an edge server but has to be designed to enhance the capabilities of the constrained IoT devices. The objectives of implementing the sub-server idea are:

- To support the lightweight communications 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 IoT devices group to reduce attack surfaces.
- To enable a more distributed computational system
- To provide two-layer validations (primary and secondary validations), IoT device to Sub-server and Sub-server to edge server. When the generated data travels from one logical zone to another zone or (to or from) the edge server, this two-stages validation could protect the privacy and isolate the devices of one logical group from other groups of the same system.

Figure 4. 1

The Architecture of IoT edge layer



The end nodes will have client principles, and the Sub-servers will have both server and client principles. This developed concept has been elaborated in section 4.2.4. Participating sensors in the network can act like the typical sensor network where the communication channels can be established in a wired or wireless medium. However, according to this designed architecture, the forming wired/wireless sensor network(s) should follow the developed principles, which are elaborated in section 4.2.1.

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 which are elaborated in sections 4.2.3 and 4.2.4. As with the typical IoT architecture, the edge servers are connected to a cloud-based server or a remote server.

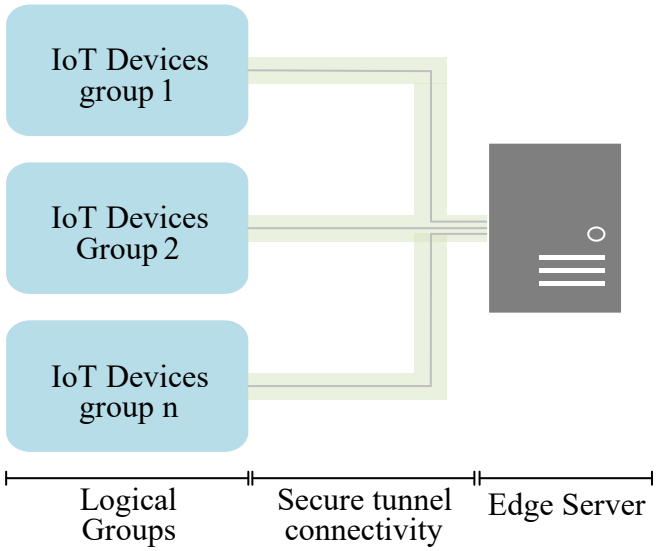
4.2.1 Concept of Zones and Groups

According to this 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 which is populated in a zone should contain the same Zonal ID. Here the zonal ID is denoted as Z_{ID}. According to this 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.

In summary, the key differences between Zone and Group concepts are presented as follows. A zone is the physical segmentation of all operational nodes whereas Group is all about the logical segmentation of the devices. Grouping can be modified by system control command whereas a zone is more resistant to change, a zone can be changed only a device is physically moved to another zone.

Figure 4. 2

Edge layer logical groups of end devices.



In an IoT system, an end device may need to transfer or receive data from other end devices to execute an operation. A 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 together to accomplish operational tasks. 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 this proposed protocol, A logical grouping categorizes the 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 the 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 modifiable with the policy framework or local/remote administration unless the physical changes happen in the Zone. However, The Z_{ID} can be determined by the system admin or through a predetermined group policy.

4.2.2 Implementation of Microservices

In a complex IIoT environment, heterogeneous devices need to be operated where maintaining the quality of service (QoS) of the system is challenging. In various contexts, conventional edge computing cannot fulfill the QoS (Jin et al., 2020). Therefore, researchers

are currently focusing on implementing the μs on IoT systems due to their several properties, including scalability, modularity, resiliency, and compatibility in heterogeneous devices (Jin et al., 2020).

In the proposed model, multiple operations need to be conducted at different levels of a system which could make the system more complex. μs can come into account to resolve the problem and simplify the operational tasks. From protocol translation service to the different kinds of gateway operations in the edge layer, μs can take place 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 requirement of latency where monolithic services can be replaced with μs (Chen et al., 2021). The μs will be deployed in a sub-server and edge server according to the predetermined policy framework.

However, to maintain the dependency relationship within the μs is complex in a large IIoT system, a large amount of data needs to be transferred among the adjacent μs (Chen et al., 2021). In a μs-based system, every service request and response should have a unique identifier called a correlation ID (Stévant et al., 2020). Managing and coordinating these unique request processing identifiers needed additional computational resources.

In this distributed computation-based architecture, to make the μs more simplistic, the μs-based applications are processed in two phases. In phase 1, only the relational services will be deployed and operated, which will be done 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.

4.2.3 Gateway functions at Sub-Server (Phase 1)

The end devices are constrained in nature. These devices have limited computational power, operate in a light-power battery, and have less memory in use. The main purpose of the sub-server is to assist the end devices with processing, memorizing, and optimizing the power consumption. Many lightweight gateway protocols are 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, which may not be done with 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 that may need to be converted to a lightweight protocol like COAP or MQTT (Amaran et al., 2015). A protocol translation may come into operation to convert the conventional protocols to the lightweight protocols or lightweight protocols to the conventional protocols.

Transceiving external requests

Transceiving, executing, and deploying external requests from a different zone, group, or network is required to pass through the various predetermined security policy of a particular system to ensure the security requirement. In this circumstance, implementing lightweight compatibility with end devices become more challenging. The security operations like authentication, authorization, data request validation, and encryption require additional computation. Sub-Servers control downstream communication (with the end IoT devices) as a parent device. As a parent device, a Sub-server should act like the head-point that will

respond to the upstream data request, encrypt the transceiving data, encapsulate, support implementing VPN tunneling, and validate the requests.

Validating transceiving requests

In an unprotected environment, IoT devices can become the entry point of an attacker to gain access to a system which may disrupt the whole system. Therefore, validating the upstream data can be an effective solution to reduce this vulnerability. The 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 correlation ID for µs (if µs is adopted).

Gateway Load balancing

In figure 4.1, we have presented our proposed IoT edge layer architecture where Sub-servers are interconnected as well as connected with the edge server. To deploy the 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 becomes developed on a distributer processing or compurgation architecture.

Adopting microservices

In a heterogeneous IoT system, the researchers show more interest in adopting µs instead of 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 requirement from the end devices, we initialized the concept of a Sub-server. The sub-server manages

every unique µs request and response on behalf of the end devices, which will exempt the end devices from additional data processing or computational resource usage. Moving the existing system solution to a smart solution has always been challenging. The Existing independent system solutions or monolithic Legacy system solutions can still be transformable with the µs (Wolfart et al., 2021).

Adopting Software-defined gateway functions

The idea of a Software-defined gateway (SDG) is designed to focus on industrial interconnection to converge with heterogeneous devices (Jiang et al., 2020). Our effort in adopting the software defined IoT gateway is to automate and dynamically manage the edge network for flexibility and easy programmability (Morabito & Beijar, 2017). In the next sections, we will discuss how the Software-defined gateway functions will be adaptable to our proposed protocol.

4.2.4 Gateway functions at Edge Server (Phase 2)

In this architecture gateway is not just a traditional gateway that is not limited to entry and exit points between the internal and external network. Beyond that, 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 logical groups of internal devices. According to our proposed model, the gateway functions at the edge server are presented as below.

Transceiving external requests

The proposed model is a gateway-centric local network where an edge server is the ultimate edge gateway device for transceiving external requests. In this gateway-centric local network, the dependent devices act as a client. The sub- server itself conveys client

supporting functionality as well as the 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 will be client mode. During executing the edge gateway functionality only to the sub-server, it transforms into client mode. In summary, in our proposed protocol, these three tire devices have three different modes: Server, Transparent, and Clint. Edge gateway and the end devices convey only the Server and Client mode respectively, but the sub-server can perform in two modes: Clint or Transparent.

Manage the encryption decision

In an IoT environment, it is very challenging to implement the robust encryption method through all IoT communication channels. To implement strong security, strong encryption needs to be implemented. Implementation of encryption depends on the security requirement. The decision of executing encryption is from lightweight to strong (Toshihiko, 2017), depending on the security requirement of the policy framework.

Protocol translation

In an IoT network, Different protocols are used to manage, control, execute the command and establish a connection between external and eternal networks. In large and complex heterogeneous IIoT networks various independent protocols are implemented. Synchronization and interoperability within this large variety of protocols are challenging (Derhamy et al., 2017). The interoperability and compatibility with various protocols are elaborated in section 4.3.2.

Managing logical groups

A logical group enables the devices logically isolate from another group of devices. Managing and interconnecting devices within a logical group ensure implementation of the group policy and other relevant policies. By implementing grouping techniques, the devices (including Sub-servers) from different zones can be compelled into a group, enhancing security, manageability, 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 of the policy framework.

Dynamic load distribution and balancing

The motive of the Sub-server is to assist the edge server with operational and computational tasks. Dynamic load distribution is the way to reduce the workload from the edge server. As a hub to the local IoT system of the proposed model, an edge server executes commands of dynamic load distribution and load balancing depending on the availability of the end nodes, traffic transceiving channel preferences, system preference, and deployment policy. An algorithm has been developed for the dynamic load distribution and balancing which is presented in Section 4.3.6.

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 for future analysis and development purposes. Traffic filtering techniques are implemented to optimize the network

traffic to comply with the available network bandwidth. In the proposed protocol, the filtering techniques are developed with categorizing, prioritizing, and processing the generated traffic based on traffic attributes and the generating device. We have developed an algorithm for filtering the generated traffic in section 4.3.3.

Adopting Microservices

At the edge server, the µs are coordinated and deployed to the associated Sub-serves based on the performing operations. Every µs deployment and operation can be identified with a unique correlation ID that determines a processing request. In this IoT service model, µs are included 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

With the evaluation of IoT network and network management features, SDN has become a promising technology to enhance the programmability, resiliency, and security of the IoT network (Morabito et al., 2017). To execute advanced gateway functions, SDN-enabled gateway functions are predicted to be adopted. Control commands of remote or cloud servers are managed at the edge server and Sub-server for the execution at the edge network. In this proposed model, the developed policy framework will define the SDN enabled gateway functions.

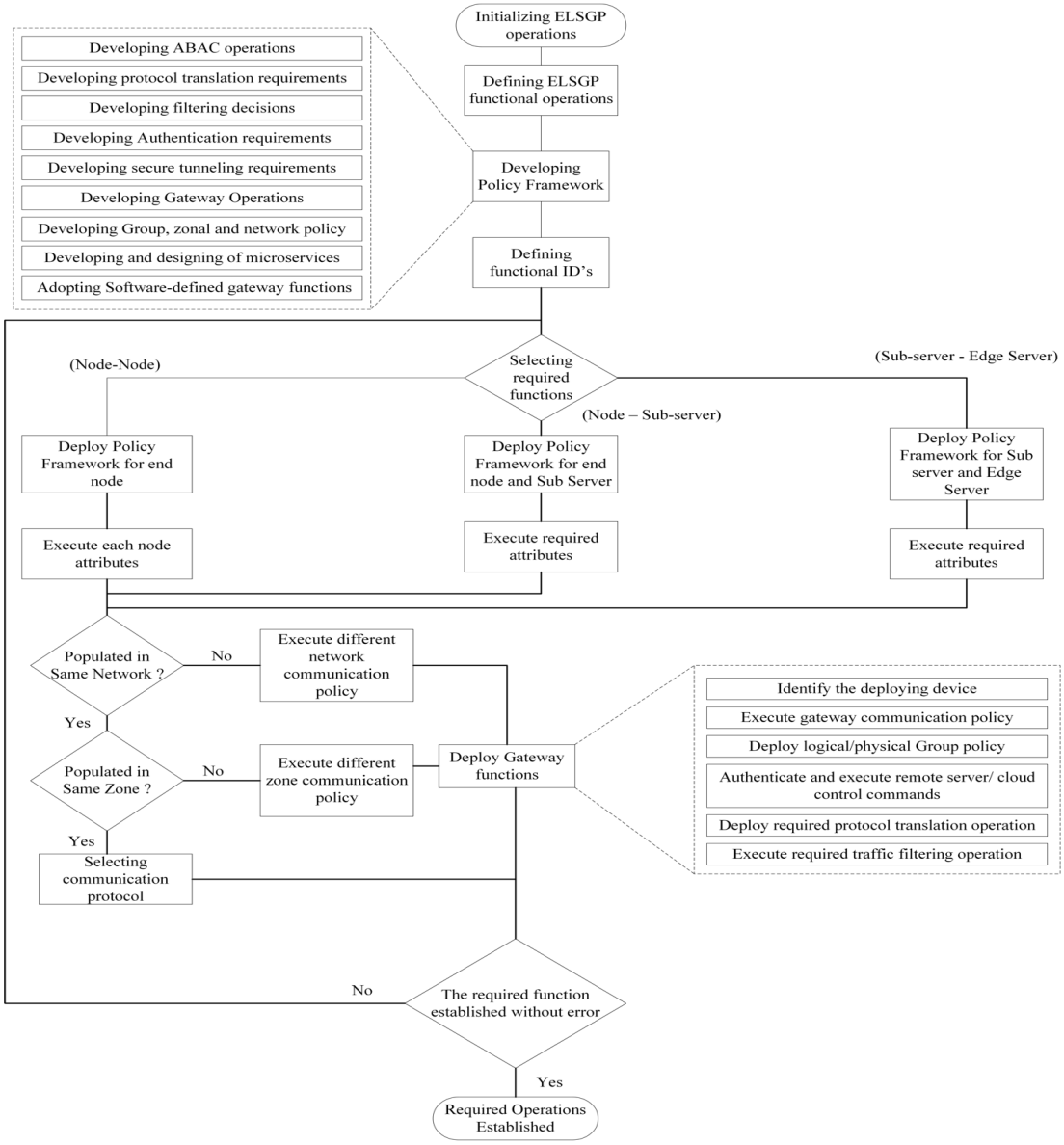
4.2.5 ELSGP operational flow and Policy Framework

Connectivity, privacy, security, adoption with open standards, device and data heterogeneity, and interoperability are the key concerns for the evaluation of IoT domains (Intel, Policy Framework for the internet of things (IoT) 2014). We have developed a prototype of a policy framework based on the key apprehensions of an IoT domain. We have

focused on the IIoT domain for developing the policy framework. To demonstrate and execute the policy framework functions in the proposed model, we have developed the algorithms with the main functions of the designed ELSGP. The developed algorithms are presented in section 4.3.

Figure 4. 3

ELSGP operational flow and Policy Framework



In figure 4.3, we have presented a flowchart for ELSGP operational streams. The ELSGP operations followed the developed policy framework. The policy framework

execution flow is illustrated with several identical operations which include: developing ABAC operations, defining protocol translation requirement and operational flow, transceiving traffic filtering decisions, Authentication requirements, secure tunneling requirement, deployment process of software-defined gateway functions, defining and developing gateway operations (if Software-defined gateway functions are not deployed), Developing Group, Zonal and Network policy (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. In our developed model, we have presented three types of devices at the edge layer: end devices (end IoT devices like sensors and actuators), Sub-server, and edge servers.

Dynamic policy generation, adjustment, and enforcement are under research for deploying various software-defined functionalities (Phung et al., 2017). To adopt the critical challenges with the designed ELSGP protocol, A policy will be dynamically deployed on each node and communication channel based on the predefined modular functions, functional IDs, and device IDs. The transceiving traffic within the nodes will also follow the ELSGP operational functions. The ELSGP functionals are illustrated in the following sections (4.3.1 to 4.3.6).

4.3 Design of ELSGP

The ELSGP is designed with six basic functions: Attribute-based access control, Authentication, Dynamic load balancing and distribution, Dynamic interoperability function, Traffic filtering, and Secure Tunneling. Algorithms will be developed based on the functions of the proposed ELSGP.

4.3.1 Authentication

In the edge server gateway, several security operations need to be processed and validated at the edge layer (Peng et al., 2021). Edge-based authentication will increase data communication security among the nodes (Pardeshi & Yuan, 2019). Untrusted request management is one of the security operations which need to be conducted before processing the generated data from an end node (Peng et al., 2021).

The proposed ELSGP authentication process is designed with three parameters device MAC, D_{ID}, and an access token. In this protocol, to enhance the security and access control, every operating device in an 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}. 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 with MAC, D_{ID} / $DD_{ID,}$ and Access token (determined depending on system security level and lightweight weight performance requirement).

The Authentication process is presented in two algorithms. The first algorithm is for the 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 can't be independently conducted, but it depends on the participating protocol(s) to finalize the authentication process.

Algorithm 1 Access Token Distribution and Validation

```
F(AT_D) // Defining Access Token Distribution function
F(AT_V)// Defining Access Token Validation function
D_ID, Z_ID, N_ID // Defining Device ID, Zone ID, and Network ID
F(DD_ID) //Defining function for Dynamically defined device ID
Get_Source () // Function for Identifying source operating device
Get_Destination ()//Function for Identifying destination operating device
Initialize F(DD_ID) // Operation at Sub-server
      Step A: Get pattern
              Define DD_ID Bitstream format
     Step B: Set DD_ID
              Generate DD_ID
              Allocate DD_ID
     Step C: Reset DD_ID
              Define Reset Period // the reset function triggering interval
              Repeat {Generate DD_ID
                      Allocate DD_ID}
Initialize F(AT_D) // Operation at edge server
     Step A: Get_Source ()
                      Get D_ID / F(DD_ID)
                      Get Z_ID, N_ID and MAC
     Step B: Get_Destination()
                      Get D_ID / F(DD_ID)
                      Get Z_ID, N_ID and MAC
     Step C: Generate AT // Access token (AT) generating operation
              Re-Generate AT
     Step D: Distribute AT // (AT) distributing operation
              Sync regenerated AT
Initialize F(AT_V) // Operation at edge server
     Step A: Verify AT Timestamp
     Step B: Verify AT Bitstream
     Step C: Validate AT
              If
                 Step A AND Step B = True
              Else
                 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 static user-defined or dynamically system-defined. For dynamic Device ID allocation, function $F(DD_{ID})$ is for dynamical allocation of device ID, and it is a Sub-server operation. The second function, $F(AT_D)$ is for Access token Distribution, an edge server operation. The third and the last 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.

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
```

The bitstream of an access token is predefined but should not be identifiable by an end device or an operating Sub-server. This Access token bitstream can be verifiable at the edge server which is expected to enhance the trustworthiness between the transceiving devices without the additional computing requirement at the end device.In algorithm 2, function `F1_Auth` is initialized for establishing authentication and protocol translation functions. The `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. The user authentication will be performed with the used protocol.

4.3.2 Dynamic interoperability function

In a real IIoT environment more than one communication protocol and operational standards have been outlined which raises the interoperability and scalability challenge (Lee et al., 2019). Recent research shows several solutions to the interoperability problem, like protocol proxy, protocol translation, and middleware or MiddleBridge (Derhamy et al., 2017; Akasiadis et al., 2019; Da Cruz et al., 2019).

Example scenario

An IIoT system can be operated with various protocols like CoAP, MQTT, REST, AMQP, HTTP, HTTPS, WebSocket, and so on. In this exemplary 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 B is being operated for the end devices. Various portions of the edge layer of the system are operated

with some legacy solution. An edge server is converting the transceiving traffic between protocols A and B.

The existing problems of the IIoT system are- the edge layer is vulnerable to low-security exposure, requires massive operational tasks for translating the protocol, huge complexity of system management and maintenance (not dynamic), and is very hard to adopt more secure 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 which is one of the limitations of middleware (Derhamy et al., 2017). Protocol proxy is another solution for the interrogatability problem, but the protocol proxy solution is not efficient enough in an IIoT environment where the number of participants is high (Derhamy et al., 2017; Akasiadis et al., 2019). Protocol translation can mitigate the limitations of middleware and protocol proxy. 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 which can convert the proprietary of the entities or the data standard of a protocol to a different protocol (Da Cruz et al., 2019). Executing a large operation every time at the edge layer, securing the seamless interconnection and the configuration compatibility for the heterogeneous end devices can be challenging for the protocol translation.

The developed interoperability Function

The interoperability function should be secure, dynamic, and compatible with various end device configurations (including legacy solutions) with the least operational tasks. 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). Implementation of these security features also depends on the adopting policy framework, or the system security level and the lightweight performance requirement.

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 confutation 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 a deployment JSON file (Vijayma, Azure IoT edge task - azure pipelines 2021). 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 the participating protocol entities. For example, two protocols are participated 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 Exemplary scenario, for the interoperability between protocols A and B, *F(p)* function can be dynamically executed. The associated deployment file can be executed by getting 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 done dynamically.

4.3.3 Attribute-based access control (ABAC)

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

Algorithm 3 Attribute-based access control

```
1. F3_ABAC () // Defining Attribute based access control function
2. A_TAG // Defining Attribute TAG from Attribute policy
3. R_F3 () // function for defining the rules from the policy
4. Begin
5.     Initialize F3_ABAC ()
6.     Initialize R_F3 ()
7.     If R_F3 () = Success
8.           Check   ATAG
9.           If F1_Auth () AND A_TAG = True
10.                Allow traffic flow
11.               Else
12.                  Reject traffic flow
13.          Else
14.                Stop F3_ABAC ()
15.                Acknowledge error initialization
16. end
```

To validate the ABAC operation, an A_{TAG} along with an Authentication function *(F1_Auth)* will authenticate the device access request for joining into a network, communicate with the other network devices, validate the generated or received data, and execute and validate the control command.

4.3.4 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 algorithm.

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
```

In the traffic filtering function, transceiving traffic has been 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- Traffic classifier, Priority function, summary function, and traffic analyzer function. Traffic filtering policy may vary in different IoT systems. The subfunctions execute the rules defined in the traffic filtering policy. The traffic analyzer subfunction categorizes the transceiving traffic as unknown traffic if the traffic cannot be determined with predefined rules.

4.3.5 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 policy will be used to establish a secure tunnel between two nodes of different networks or system zones.

Algorithm 5 Secure Tunneling

```
1.  F5_ST ()// Defining the secure tunneling function
2.  O, G, Z, N //Object, Group, Zone, and network
3.  Encap_Dcap, Enc_Dec // Encapsulation and encryption requirement
    value
4.  Tunnel_sec // the secure tunneling policy
5.  Encap(), Dcap (), Enc_t ()// defining encapsulation, decapsulation,
    encrypted tunneling functions
6.  P() // network path defining function
7.  Begin
8.      Initialize F5_ST ()
9.          Get O, G, Z, N // for Source
10.         Get O, G, Z, N // for destination
11.         Initialize P ()
12.               Get Logical path // between source and destination
13.         If Encap_Dcap= true // at lightweight state
14.             Initialize Encap() // For transmitting traffic
15.             Get keys// for device itself and next-hop device (from
    Source)
16.             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
```

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 device, 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 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 like Secure Shell (SSH), 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 or system-defined the term to an established secure connection with a remote IoT device (Yarali, IOT: Platforms, connectivity, applications, and services 2018).

Lightweight state

At a very low computing stage where a full strong encrypted tunnel can't be created. A Secure encapsulation method can be applied to transceive traffic between the source and destination device. In the algorithm, we presented a method of secure tunneling where 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 another key is for the next-hop device (where the hop count is 1). *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 with this order $O \rightarrow Z \rightarrow G \rightarrow N$. Data decapsulation is for the destination, which is directed with the reverse order, and followed by *XNOR* operation to retrieve the received data frame.

Policy defined state

A policy defined state is also the solution like the lightweight state defined with the position of system security requirement. This arrangement fulfills the requirement of balancing the full strong encryption and lightweight state. *Tunnel_sec* defines the secure tunneling policy which includes the participating protocol and policy applying network segments. The policy deployment file will be defined with a unique deployment ID for future reuse.

4.3.6 Dynamic load distribution and balancing

The function of dynamic load distribution and balancing depends on several parameters of 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-server, load threshold value for each edge server, load threshold value for each sub-server, health check threshold values, task queue threshold value for each edge server, and task queue threshold value of each participating sub-server. The health check value depends on the CPU capacity, Power status, channel status, computing time, and system preference.

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 ≥ L or TQ≥ TQ_t )
           Calculate ΔL // ΔL ←(CL-L)
           Select ls
                If cl <l AND h_t <h AND tq<tq_t
                Endif
           Distribute   ΔL∝ Δl or  ΔTQ∝ Δ tq //ΔTQ←(TQ-TQ_t) and Δtq←(tq_t-tq)
           Update load parameters
           Update status parameters
      Else repeat Step c
```

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

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 the $F6\_dldb()$ 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 $\Delta\ tq$. The load information and status parameters are updated according to the executed modifications.

Chapter 5. Analysis and evaluation

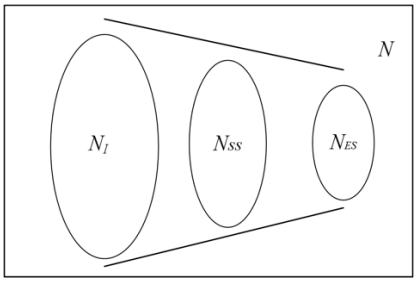
In chapter 4, we outlined the distributed computational model and protocol features. In this chapter, we analyze and evaluate the performance and features of the designed protocol based on simulation results. Section 5.1 presents the mathematical modeling of the distributed computational system. The following section, 5.2, presents the analysis of the developed algorithm. Evaluation of the features with several fault models is presented in section 5.3. We also presented probability analysis, an explanation of the simulation process, and the comparison and evaluation of the findings are presented respectively, in sections 5.4, 5.5, and 5.6.

5.1 Modelling 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 $\mathcal{N}_I$ number of IoT devices, $\mathcal{N}_{SS}$ number of sub-servers, and $\mathcal{N}_{ES}$ number of edge servers. Therefore, the total number of nodes, $\mathcal{N}$ in the edge layer of the system is equal to the summation of the numbers of IoT devices, sub-servers, and edge servers. Therefore $\mathcal{N} = \mathcal{N}_I + \mathcal{N}_{SS} + \mathcal{N}_{ES}$. With this distributed system architecture, the number of the nodes must follow the order as $\mathcal{N}_I > \mathcal{N}_{SS} > \mathcal{N}_{ES}$; figure 5.1 presents the order.

Figure 5. 1

Orientation of the edge layer nodes



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.

$$\mathcal{N}_I = \sum_{1}^{\mathcal{N}ss} (I_{\mathcal{N}_{SS}})$$

Here, $I_{\mathcal{N}_{SS}}$ represents the numbers of adjacent IoT nodes with each Sub-server. If we denote the number of nodes including the sub-server itself as S_S then

$$\mathcal{E}_S = \sum_{1}^{\mathcal{N}_{ES}} (S_{S\mathcal{N}_{ES}} + 1)$$

Similarly, if we denote the number of nodes including the edge server itself as E_S then where each sub-server and IoT device are considered as nodes. In other word, the total number of nodes in the edge layer of a system $\mathcal{N}$ is equal to the accumulation of the numbers of nodes adjacent to each edge server $\mathcal{E}_S$.

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, the modeling presents how every functioning node will be interconnected in the edge layer of a system.

5.2 Analysis of the Algorithm

In chapter 4, we presented the six features of the proposed protocol which are: access token distribution and validation, authentication and dynamic interoperability, attitude-based access control, secure tunneling, and dynamic load distribution and balancing. In this section, 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.

5.2.1 Access token distribution and validation

In a predefined period, the frequency of generating DD_{ID} is denoted as $\mathcal{F}_p$, the period is defined according to the system requirement and policy. The generated DD_{ID} is considered as the periodic traffic which is the subclass of one of the classified traffic types in section 4.3.4. For the frequency, $\mathcal{F}_p$ 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 $\mathcal{F}_p$ can be defined as

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

$$\mathcal{F}_p = \frac{\sum_{1}^{n_s} t_{n_s}}{t} \text{ or } F_p = \frac{p}{t}$$

To calculate the amount of traffic generated due to the distributed access token, the associated parameters need to be determined. According to the developed algorithm in section 4.3.1, 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

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 $\mathcal{T}_{OA}$ As

$$\mathcal{T}_{OA} = I_p$$

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, which includes 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

$$\mathcal{T}_{OA} = I_{pex} + [D_{ID}]^{Fp}$$

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

$$\mathcal{T}_{OA} = [I_{pex} + [D_{ID}]^{Fp}] \,||\, [I_p]$$

5.2.2 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 4.3.1 and 4.3.2, 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 the volume of traffic generated with the interoperability function is $\mathcal{T}_{IO}$.

Secure tunneling creates a logical path between the source and destination device. A logical path can be defined in three ways - a full strong encrypted state, a lightweight state, and a policy-defined state (we introduced those states in section 4.3.5). Based on the system preferences, one of the processes will be executed. Among the three, the full strong encrypted state required the highest consumption of resources, and the lightweight state required the least. The resources include memory, processing unit, communication unit and power sources (Zahoor & Mir, 2021). In section 5.6, we will illustrate different metrics related to resource utilization.

If we consider the average traffic overhead due to establishing a logical path as $\mathcal{T}_{FS}$, $\mathcal{T}_{LS}$, and $\mathcal{T}_{PS}$ for the full strong encrypted state, lightweight state, and policy-defined state, respectively, then the traffic overhead due to secure tunneling, $\mathcal{T}_{ST}$ can be calculated as

$$\mathcal{T}_{ST} = [\mathcal{T}_{FS}] \,||\, [\mathcal{T}_{LS}] \,||\, [\mathcal{T}_{PS}]$$

If we assume $\mathcal{T}_O$ as the total traffic overhead of the proposed ELSGP protocol, then $\mathcal{T}_O$ can be calculated as

$$\mathcal{T}_O = \mathcal{T}_{OA} + \mathcal{T}_{IO} + \mathcal{T}_{FS}$$

5.3 Fault Analysis

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 which include Byzantine fault, Transient fault, and cascading failure. The well-known Byzantine fault model is chosen to interpret the resiliency against fault nodes (due to identified vulnerability, affected with known or unknown attacks, infected with malware, exhibiting malicious behavior, or so on). Transient fault and cascading failure cause the

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

5.3.1 Byzantine Fault

Byzantine fault appeared 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 appearing as malfunctioning is called a Byzantine node. To tolerate the fault, the number of faults should be fewer than 1/3 of the total nodes (Castro et al., 1999). 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 the fault resiliency, we have developed a Trust and Priority Impact relation which is presented in the following section, 5.3.2.

5.3.2 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 have determined different degrees of trust. Here, the lower the value of the degree of trust means higher the trustworthiness. If we consider the degrees of trust as $\mathcal{D}_t$

Table 5. 1

Different degrees of trust

Degree of Trust	Types of nodes
$\mathcal{D}_t^4$	End nodes /IoT devices/ sensors/ actuators
$\mathcal{D}_t^3$	Sub-Server
$\mathcal{D}_t^2$	Edge Server
$\mathcal{D}_t$	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,

$$p_i = \frac{1}{\mathcal{D}_t^p}$$

And if we consider n as the number of nodes, then the priority impact of the faultiness of nodes can be presented as

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

During the fault scenario, other nodes will determine P_i, here higher value of P_i means the higher impact of the faultiness, and lower P_i means the lower impact of the faultiness. If we calculate the degree of trust for n number of devices with

$$\frac{1}{\mathcal{D}_t^p} = \frac{\sum_1^n (p_n f_n)}{n}$$

$$\mathcal{D}_t^p = \frac{n}{\sum_1^n (p_n f_n)}$$

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

Where n number of devices with the same power of degree of trust. If we consider a group of devices with different p then the $\mathcal{D}_t$ of the group of devices can be calculated as the multiplication of $\mathcal{D}_t$ values with different p values.

5.3.3 Transient fault

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 (Lee et al., 2021). There is no prevention mechanism introduced in our distributed computing model. However, we have

adopted the fault mitigation mechanism to resilient the system with this fault. The mitigation process is stepped into four stages which are presented as 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 determined 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 presented in section 4.3.4.

5.3.4 Cascading failure

Cascading failure causes devastating consequences for a system. A cascading event happens with a node and triggers the event in other nodes (Xing, 2021). (Xing, 2021) present a systematic review with fault modeling and analyzes the mitigation process by understanding the cause of the failure. To ensure resiliency and reliability in a distributed computational system, we have developed an isolation mechanism as a part of the access token validation and distribution process which will enhance the security of the system as well as mitigate the cascading failure. In section 4.2.1, 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 failure triggered a domino like a chain effect. Although there is a chance to contain the failure within a portion of the system, the isolation technique will prevent the fault from spreading to a large portion or to the whole system elements. Hence, the mitigation mechanism can ensure resiliency and reliability against failure.

5.4 Probability analysis

In this section, we present a probabilistic analysis of a system during unstable conditions. Among these, when some of the system components behave arbitrarily and shows unpredictive nature, it generates traffic in response to the unstable condition (the categories of the generated traffic are presented in section 4.3.4). 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 the faultiness is a discrete event and happens within a period, $\mathcal{P}$, then the probability of faultiness can be calculated as

$$\mathcal{P}(f_p) = \frac{|\mathcal{T}_{start} - \mathcal{T}_{end}|}{\mathcal{P}}$$

$$\mathcal{P}(f_p) = \frac{\mathcal{T}_{\Delta\mathcal{T}}}{\mathcal{P}}$$

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

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

Here, f_{ut}, f_{pt}, f_{ft} represent the fault event with responding unknown, priority, and flooding traffic accordingly, and $\mathcal{P}(f_{ut})$, $\mathcal{P}(f_{pt})$, $\mathcal{P}(f_{ft})$ presents the probability of getting corresponding traffic within the calculated period. If during the arbitrary or unstable condition any of these three types of traffic is generated, then the probability of faultiness is equal to the probability of getting responding traffic. Hence,

$$\mathcal{P}(f_p) = \mathcal{P}(f_t)$$

5.5 Explanation of Simulation process

The performance of the proposed protocol is evaluated with the OMNET++ simulator by evolving a simulation scenario. We selected several performance metrics to present comparative results with other protocols including CoAP, MQTT and DDS.

5.5.1 Simulation Scenario

In the simulation environment, a sample network topology is created with 19 nodes, including 12 end IoT nodes, 4 Sub-server nodes, 2 Edge server nodes, and one remote server. Table 5.2 shows the chosen simulation parameters, and figure 5.2 shows the OMNET++ simulation topology screenshot.

Table 5. 2

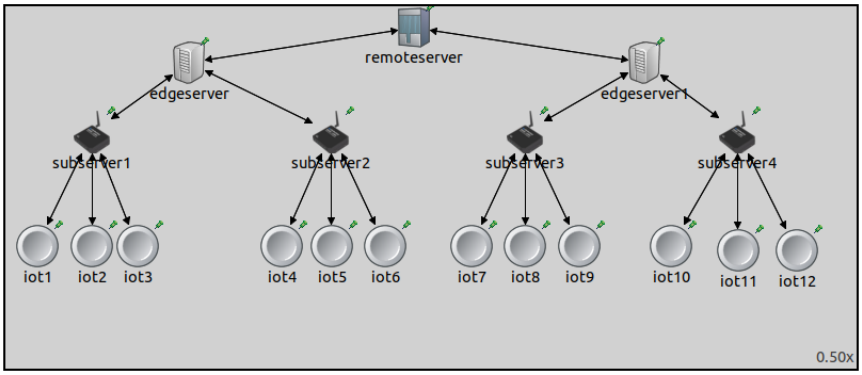
Chosen values for different simulation parameters

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

With this example scenario, the simulation process is conducted 12 times for 100s to achieve 95% confidence interval. With the simulation results, node(s) may not act as faulty which presents results without considering faults. Performance metrics are defined with appropriate parameters to extract and record results. Section 5.5.2 presents several performance metrics used in the simulation.

Figure 5.2

OMNET++ simulating topology screenshot



5.5.2 Performance metrics

Performance metrics provide measurable statistics that help to evaluate the performance of a protocol (Seliya et al., 2009). To evaluate the performance of ELSGP, we select three metrics, end-to-end delay, power consumption and throughput. These metrics are selected based on the acceptability and appropriateness for the performance diagnostics of network elements. In the simulation, end-to-end delay is calculated as the summation of transmission delay, processing delay, and queuing time which defines the required time to get a packet at the destination node from the source node. Power consumption of IoT nodes exhibits the average of power consumed by each node during the simulation run time. Lastly, the throughput metric shows the number of packets successfully transmitted or received by each IoT node.

5.5.3 Simulation results

Figure 5.3 shows the average end-to-end delay and mean end-to-end delay. The graph plots 5551 packets for 100s, and these generated packets travel a maximum of 5 hops to arrive at the destination node. For the first 30s, the graph shows an unstable end-to-end delay line which is because of uneven traffic overhead, but subsequently it becomes more stable. The mean value of end-to-end delay is recorded as 0.01099s which is equal to 10.99 ms. Here

to add that the result of end-to-end delay does not consider propagation delay as the delay is very insignificant.

Figure 5.3

Graph of End-end-delay and end-to-end delay (mean)

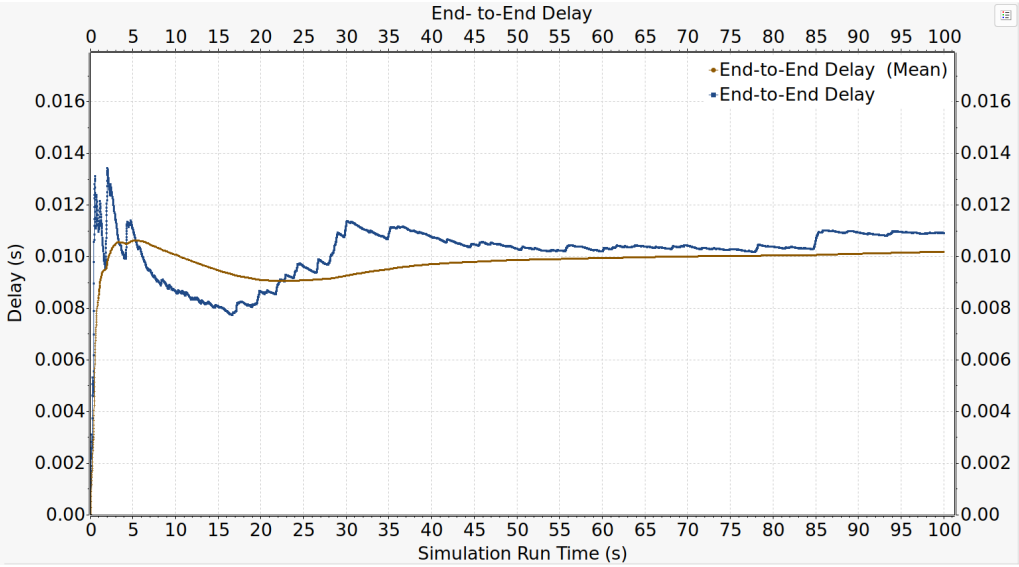


Figure 5.4 presents the proportion of power consumption by each end node over time. It is clearly observable that the plotted line shows mostly steady state conditions, although there is some instability due to initial data processing on each node for the first 30 seconds. The mean value of the proportion of power consumption is noted as 0.00894 of 100J (energy storage module of INET used in each end node). In another world, the energy consumption rate is 0.894J/s.

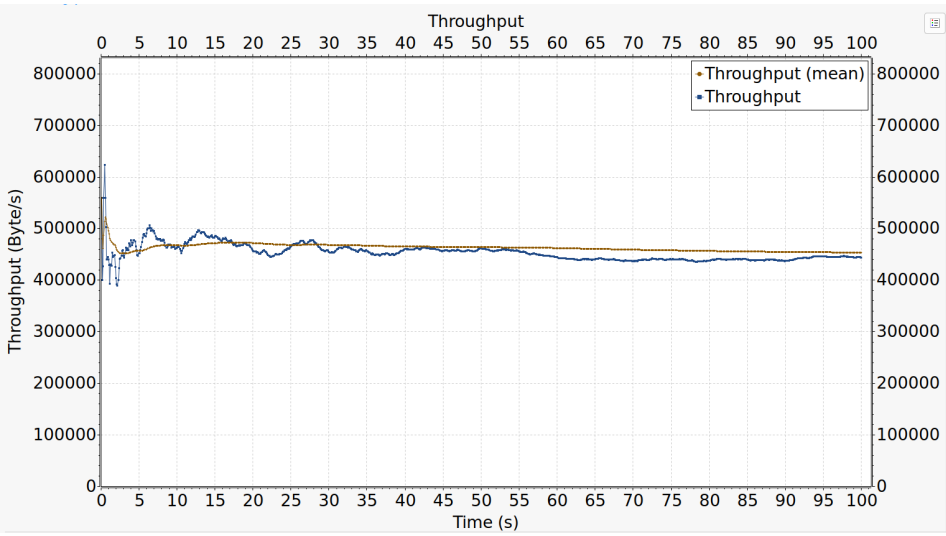
Figure 5.4

Graph of proportion of power consumption per IoT node.

Figure 5.5 illustrates the average throughput per end node over time. It is clearly observed that the average throughput is noted as a non-steady state condition for the first few seconds which is due to uneven packet volume and processing time.

Figure 5.5

Graph of throughput and throughput mean.



The simulation result shows that the mean throughput value for end nodes is 444080.0 bytes per second which is equivalent to 3.553Mbps. Figure 5.5 shows the plotted values for 5551 packets transceived across different end nodes for 100s time. Here, the parameter of the probability of faultiness is taken randomly between 0.0 and 0.9 where a lower value means less likelihood of being faulty and a higher value means a high 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.

5.6 Comparison and evaluation of the findings

In this section, we show the comparison of performance metrics of ELSGP with other comparative protocols including CoAP, MQTT and DDS based on the simulation results and the performance evaluated in several sources (Guaman et al., 2020; N.V. & P., 2020; Bansal & Priya, 2020). Section 5.6.1 presents the comparison and evaluation of the performance metrics and section 5.6.2 limitations of the performance evaluation.

5.6.1 Comparison and evaluation

To measure the performance of the comparative protocols, throughput, end-to-end delay, and power consumption are also recognized as valuable metrics. Figure 5.6 compares the measured throughput of ELSGP with comparative protocols. This comparison only shows the achieved throughput for IoT nodes which is measured as average 3.553Mbps per node, whereas CoAP, DDS and MQTT show 2.02, 1.62 and 2.38 Mbps respectively. In other words, the throughput of IoT node with ELSGP shows 75.89% higher than CoAP, 49.28% higher than MQTT and more than double of the throughput of DDS.

Figure 5.6

Throughput Comparison

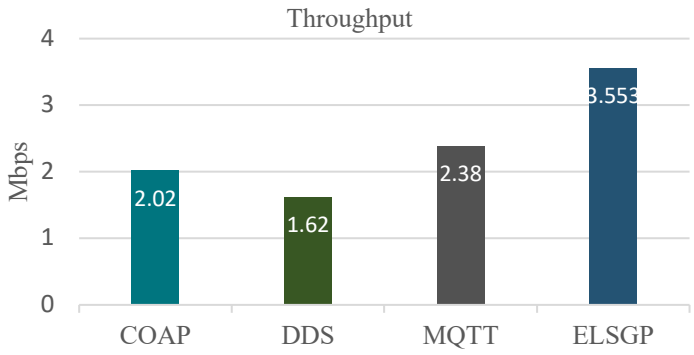
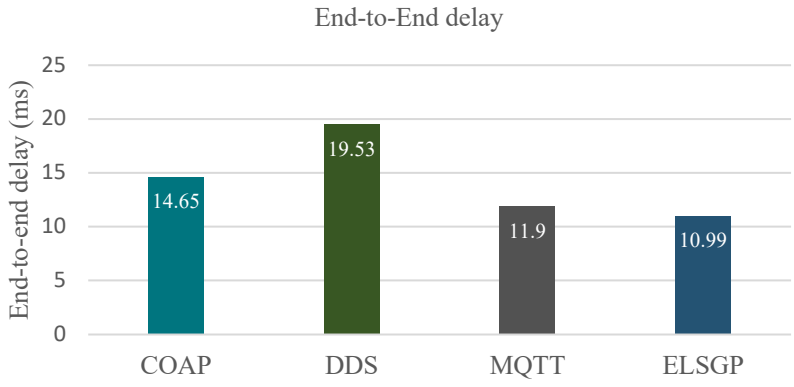


Figure 5.7 shows the comparison among the protocols in terms of End-to-End delay. The graph with the result shows that the lowest delay is for ELSGP at 10.99ms. The main objective of distributed computation is to enhance the computing capabilities of edge layer nodes to achieve high performance for real-time operations. Based on figure 5.7, ELSGP achieves minimum delay within the existing lightweight protocols.

Figure 5.7

Comparison of end-to-end delay



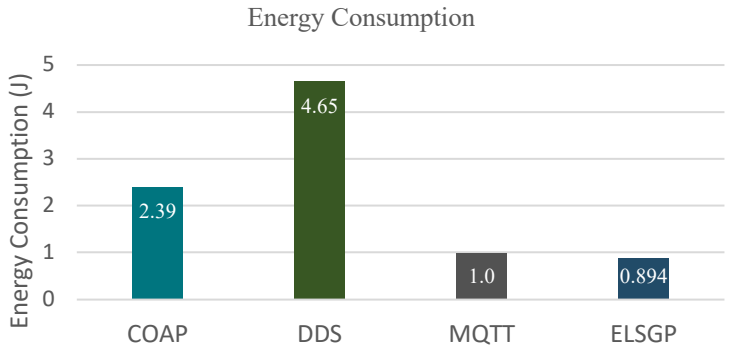
Obviously, low traffic overhead reduces energy consumption, and it gives leverage to utilize the energy for power-constrained devices. Figure 5.8 illustrates that the average power consumption rate per IoT node with ELSGP is 0.894 J/s which is more than two times and

more than five times less in comparison with the rate of CoAP and DDS, respectively. The result also shows the power consumption rate with ELSGP is 10.6% lower than the rate with MQTT.

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

Figure 5.8

Comparison of energy consumption



5.6.2 Limitation of performance evaluation

The proposed protocol is designed based on the developed distributed computational model whereas the comparative protocols are developed for traditional edge layer architecture. Hence, the comparison result 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. The nature of faultiness is considered as we presented in section 5.3. However, the fault is defined only with a set of predetermined random probabilistic values.

Chapter 6. Conclusion and Future work

This chapter is split into three sections. In section 6.1, we summarize the background of the study. In section 6.2, we cover the contributions of the study, and in section 6.3, the possible future directions would be presented.

6.1 Background of the study

With the rapid expansion of IoT and IoT-dependent systems, the chances of security breaches and numbers of vulnerabilities and threats are also rising. Due to the constrained resources, including 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.

6.2 Summary of the contributions

This research proposed an enhanced lightweight security gateway protocol based on a developed distributed computational model. The summary of the contributions is outlined below.

By scrutinizing the characteristics of IoT devices and system requirements, this research developed a distributed computational model for the edge layer (described in sections 4.2.3, 4.2.4, and 5.1). This model introduces a new term for a device called a sub-server which functions as an edge server but not as the substitute of an edge server. 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 the 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 securely establishing connectivity and transmitting traffic between two nodes. In a heterogeneous system, devices are operated with more than one communication protocol and operational standards, and the interoperability function enables to be compatible with various end device configurations (including legacy solutions) (discussed in 4.3.1 and 4.3.2). The ABAC operations are also a part of Authentication which enables (using attribute tags) to authenticate the device access request for joining into a network, communicate with the other network devices, validate the generated or received data, and execute and validate the control command (discussed in section 4.3.3). The traffic filtering function facilitates the deployment of traffic filtering policies to reduce the volume of unwanted traffic (as shown in section 4.3.4). Among the features of ELSGP, a secure tunnel enables to establish a logical path between the sending and receiving ends. ELSGP defines three different states; these are full encrypted state, lightweight state and policy defined state (as presented in section 4.3.5). Lastly, a dynamic load distribution and balancing function is developed with two different types of parameters- load information and load status parameters which enables to balance and distribute loads within policy defined sub-server(s) and edge server(s) (as described in section 4.3.6).

Considering the variability of system requirements, ELSGP adopts the policy framework which can define the protocol features according to the requirements which also includes microservice deployment and software-define operations. This study also showed fault mitigation mechanisms for byzantine, cascading, and transient faults (in sections 5.3 and 5.4).

Evaluation and comparison result shows that ELSGP shows enhanced performance in terms of throughput, end-to-end delay end 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 much adaptable to improve the performance of time-critical real-world scenario which includes manufacturing plants, process plants, industrial inspection and vision processing, highly IoT-intensified warehouses, healthcare (surgical centers), and anywhere where real-time data processing, local decision-making and enhance security for edge environment would be required.

6.3 Future scope

This research was the first step to develop a lightweight protocol under a distributed computational model. In future work, we would be focused on investigating the open research problems. The future directions can be summarized as follows

We would like to investigate further on the performance analysis of the proposed protocol. In this study, we have evaluated the performance under different conditions. Still, further analysis can be done by re-evaluating the protocol performance with other relevant metrics and, if required, identifying and rectifying the required modifications of the protocol design. Performance may vary with different sizes of a network, mobility patterns, and various system requirements. We simulated and evaluated the protocol performance under predetermined fault conditions and a known environment, but the findings may vary under unknown conditions. Mobility of the operating devices and characteristics of communication channels also may impact on the performance. Future investigations for performance analysis would consider these conditions.

High performing and cost-effective solution of the sub-server to execute this protocol (like the transformability of Raspberry pi 4, an Arduino board, or a newly designed device) is beyond the limit of this thesis scope but could be an interesting direction to the future research. As this study mainly focused on an environment with a large number of edge devices like IIoT, further studies can be done on the adaptability of the protocol in IIoT.

References

Akasiadis, C., Pitsilis, V., & Spyropoulos, C. D. (2019). A multi-protocol IOT platform based on open-source frameworks. Sensors, 19(19), 4217. https://doi.org/10.3390/s19194217

Amaran, M. H., Noh, N. A., Rohmad, M. S., & Hashim, H. (2015). A comparison of lightweight communication protocols in robotic applications. Procedia Computer Science, 76, 400–405. https://doi.org/10.1016/j.procs.2015.12.318

Azevedo, R. D., Machado, G. R., Goldschmidt, R. R., & Choren, R. (2021). A Reduced Network Traffic Method for IoT Data Clustering. *ACM Transactions on Knowledge Discovery from Data*, *15*(1), 1–23. https://doi.org/10.1145/3423139

Bansal, M., & Priya. (2020). Performance comparison of MQTT and CoAP protocols in different simulation environments. Lecture Notes in Networks and Systems, 549–560. https://doi.org/10.1007/978-981-15-7345-3_47

Beese, J., Haki, M. K., Aier, S., & Winter, R. (2019). Simulation-based research in information systems. Business & Information Systems Engineering, 61(4), 503-521.

Buchanan, W. J., Li, S., & Asif, R. (2017). Lightweight cryptography methods. Journal of Cyber Security Technology, 1(3-4), 187–201. https://doi.org/10.1080/23742917.2017.1384917

Campanile, L., Gribaudo, M., Iacono, M., Marulli, F., & Mastroianni, M. (2020). Computer network simulation with ns-3: A systematic literature review. Electronics, 9(2), 272.

Castro, M., & Liskov, B. (1999, February). Practical byzantine fault tolerance. In OsDI (Vol. 99, No. 1999, pp. 173-186).

Celebi, H. B., Pitarokoilis, A., & Skoglund, M. (2019). Low-Latency Communication with Computational Complexity Constraints. 2019 16th International Symposium on Wireless Communication Systems (ISWCS). https://doi.org/10.1109/iswcs.2019.8877142

Chen, L., Xu, Y., Lu, Z., Wu, J., Gai, K., Hung, P. C., & Qiu, M. (2021). IOT μs deployment in edge-cloud hybrid environment using reinforcement learning. IEEE Internet of Things Journal, 8(16), 12610–12622. https://doi.org/10.1109/jiot.2020.3014970

Cherif, A., Belkadi, M., & Sauveron, D. (2019). A Lightweight and Secure Data Collection Serverless Protocol Demonstrated in an Active RFIDs Scenario. *ACM Transactions on Embedded Computing Systems*, *18*(3), 1–27. https://doi.org/10.1145/3274667

Chze, P. L. R., & Leong, K. S. (2014). A secure multi-hop routing for IoT communication. *2014 IEEE World Forum on Internet of Things (WF-IoT)*. https://doi.org/10.1109/wf-iot.2014.6803204

Da Cruz, M. A. A., Rodrigues, J. J. P. C., Lorenz, P., Solic, P., Al-Muhtadi, J., & Albuquerque, V. H. (2019). A proposal for Bridging Application Layer Protocols to HTTP on IOT Solutions. Future Generation Computer Systems, 97, 145–152. https://doi.org/10.1016/j.future.2019.02.009

Daniel D, A., & Roslin, S. E. (2021). Data validation and integrity verification for trust-based data aggregation protocol in WSN. Microprocessors and Microsystems, 80, 103354. https://doi.org/10.1016/j.micpro.2020.103354

Derhamy, H., Eliasson, J., & Delsing, J. (2017). IOT interoperability—on-demand and low latency transparent multiprotocol translator. IEEE Internet of Things Journal, 4(5), 1754–1763. https://doi.org/10.1109/jiot.2017.2697718

Di Pascale, E., Macaluso, I., Nag, A., Kelly, M., & Doyle, L. (2018). The Network as a Computer: A Framework for Distributed Computing Over IoT Mesh Networks. IEEE Internet of Things Journal, 5(3), 2107–2119. https://doi.org/10.1109/jiot.2018.2823978

Dorathy, I., & Chandrasekaran, M. (2018). Simulation tools for mobile ad hoc networks: a survey. Journal of applied research and technology, 16(5), 437-445.

Figueroa-Lorenzo, S., Añorga, J., & Arrizabalaga, S. (2021). A Survey of IIoT Protocols: A Measure of Vulnerability Risk Analysis Based on CVSS. ACM Computing Surveys, 53(2), 1–53. https://doi.org/10.1145/3381038

Fruth, M. (2011). Formal methods for the analysis of wireless network protocols (Doctoral dissertation, University of Oxford).

Gao, Y., Zhang, J., Guan, G., & Dong, W. (2020). LinkLab: A Scalable and Heterogeneous Testbed for Remotely Developing and Experimenting IoT Applications. 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI). https://doi.org/10.1109/iotdi49375.2020.00025

Gao, Y., & Ran, L. (2019). Collaborative Filtering Recommendation Algorithm for Heterogeneous Data Mining on the Internet of Things. IEEE Access, 7, 123583–123591. https://doi.org/10.1109/access.2019.2935224

Gebremichael, T., Jennehag, U., & Gidlund, M. (2019). Lightweight IoT Group Key Establishment Scheme from the One Time Pad. 2019 7th IEEE International

Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). https://doi.org/10.1109/mobilecloud.2019.00021

Guaman, Y., Ninahualpa, G., Salazar, G., & Guarda, T. (2020). Comparative performance analysis between MQTT and CoAP protocols for IOT with raspberry pi 3 in IEEE 802.11 environments. 2020 15th Iberian Conference on Information Systems and Technologies (CISTI). https://doi.org/10.23919/cisti49556.2020.9140905

Haddadi, H., & Christophidesy, V. (2018). SIOTOME: An Edge-ISP Collaborative Architecture for IoT Security. *1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec)*.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., & Sikdar, B. (2019). A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. IEEE Access, 7, 82721-82743. doi:10.1109/access.2019.2924045

Hiller, J., Henze, M., Serror, M., Wagner, E., Richter, J. N., & Wehrle, K. (2018). Secure Low Latency Communication for Constrained Industrial IoT Scenarios. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. Published. https://doi.org/10.1109/lcn.2018.8638027

Intel. (2014). Policy Framework for the internet of things (IOT). Intel. Retrieved January 4, 2022, from https://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/policy-iot-framework.pdf

Issariyakul, T., & Hossain, E. (2009). Introduction to network simulator 2 (NS2). In Introduction to network simulator NS2 (pp. 1-18). Springer, Boston, MA.

Iqbal, W., Abbas, H., Daneshmand, M., Rauf, B., & Bangash, Y. A. (2020). An in-depth analysis of iot security requirements, challenges, and their countermeasures via software-defined security. IEEE Internet of Things Journal, 7(10), 10250–10276. https://doi.org/10.1109/jiot.2020.2997651

Jiang, X., Lora, M., & Chattopadhyay, S. (2020). An Experimental Analysis of Security Vulnerabilities in Industrial IoT Devices. *ACM Transactions on Internet Technology*, *20*(2), 1–24. https://doi.org/10.1145/3379542

Jin, W., Xu, R., You, T., Hong, Y., & Kim, D. (2020). Secure Edge Computing Management Based on Independent μs Providers for Gateway-Centric IoT Networks. *IEEE Access, 8*, 187975-187990. doi:10.1109/access.2020.3030297

Jiang, Z., Chang, Y., & Liu, X. (2020). Design of software-defined gateway for Industrial Interconnection. Journal of Industrial Information Integration, 18, 100130. https://doi.org/10.1016/j.jii.2020.100130

Kaed, C. E., Khan, I., Van Den Berg, A., Hossayni, H., & Saint-Marcel, C. (2018). SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways. IEEE Transactions on Industrial Informatics, 14(2), 715–724. https://doi.org/10.1109/tii.2017.2769001

Khan, A. R., Bilal, S. M., & Othman, M. (2012, November). A performance comparison of open source network simulators for wireless networks. In 2012 IEEE international conference on control system, computing and engineering (pp. 34-38). IEEE.

Khan, M. N., Rao, A., & Camtepe, S. (2021). Lightweight Cryptographic Protocols for IoT-Constrained Devices: A Survey. IEEE Internet of Things Journal, 8(6), 4132–4156. https://doi.org/10.1109/jiot.2020.3026493

Khan, W. Z., Rehman, M. H., Zangoti, H. M., Afzal, M. K., Armi, N., & Salah, K. (2020). Industrial internet of things: Recent advances, enabling technologies and open challenges. Computers & Electrical Engineering, 81, 106522. https://doi.org/10.1016/j.compeleceng.2019.106522

Kirupakar, J., & Shalinie, S. M. (2019). Situation Aware Intrusion Detection System Design for Industrial IoT Gateways. *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*. doi:10.1109/iccids.2019.8862038

Kothari, C. R. (2004). Research methodology: Methods and techniques. New Age International.

Lee, C.-H., Wu, Z.-L., Chiu, Y.-T., & Chen, V.-S. (2019). Heterogeneous industrial IOT integration for manufacturing production. 2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS). https://doi.org/10.1109/ispacs48206.2019.8986308

Lee, Y.-L., Arizky, S. N., Chen, Y.-R., Liang, D., & Wang, W.-J. (2021). High-availability computing platform with Sensor Fault Resilience. Sensors, 21(2), 542. https://doi.org/10.3390/s21020542

Lounis, K., & Zulkernine, M. (2020). Attacks and Defenses in Short-Range Wireless Technologies for IoT. IEEE Access, 8, 88892–88932. https://doi.org/10.1109/access.2020.2993553

Ma, D., Lan, G., Hassan, M., Hu, W., & Das, S. K. (2020). Sensing, Computing, and Communications for Energy Harvesting IoTs: A Survey. IEEE Communications Surveys & Tutorials, 22(2), 1222–1250. https://doi.org/10.1109/comst.2019.2962526

Minoli, D., Sohraby, K., & Kouns, J. (2017). IoT security (IoTSec) considerations, requirements, and architectures. *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. doi:10.1109/ccnc.2017.7983271

Morabito, R., & Beijar, N. (2017). A framework based on SDN and containers for dynamic service chains on IOT Gateways. Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems. https://doi.org/10.1145/3094405.3094413

N.V., R. K., & P., M. K. (2020). Application of SDN for Secure Communication in IOT environment. Computer Communications, 151, 60–65. https://doi.org/10.1016/j.comcom.2019.12.046

Nunamaker Jr, J. F., Chen, M., & Purdin, T. D. (1990). Systems development in information systems research. Journal of management information systems, 7(3), 89-106.

Pardeshi, M. S., & Yuan, S. M. (2019). SMAP Fog/Edge: A Secure Mutual Authentication Protocol for Fog/Edge. *IEEE Access*, *7*, 101327–101335. https://doi.org/10.1109/access.2019.2930814

Pawlikowski, K., Jeong, H.-D. J., & Lee, J.-S. R. (2002). On credibility of simulation studies of telecommunication networks. IEEE Communications Magazine, 40(1), 132–139. https://doi.org/10.1109/35.978060

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of management information systems, 24(3), 45-77.

Peng, C., Chen, J., Vijayakumar, P., Kumar, N., & He, D. (2021). Efficient Distributed Decryption Scheme for IoT Gateway-based Applications. ACM Transactions on Internet Technology, 21(1), 1–23. https://doi.org/10.1145/3414475

Phung, P. H., Truong, H.-L., & Yasoju, D. T. (2017). P4SINC - an execution policy framework for IOT services in the edge. 2017 IEEE International Congress on Internet of Things (ICIOT). https://doi.org/10.1109/ieee.iciot.2017.23

Razaque, A., & Rizvi, S. S. (2017). Secure data aggregation using access control and authentication for wireless sensor networks. *Computers & Security*, *70*, 532–545. https://doi.org/10.1016/j.cose.2017.07.001

Sachan, A., Kumar, D. N., & Adwiteeya, A. (2019). Light Weighted Mutual Authentication and Dynamic Key Encryption for IoT Devices Applications. 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT). https://doi.org/10.1109/icict46931.2019.8977672

Sargent, R. G. (2015). An introductory tutorial on verification and validation of simulation models. 2015 Winter Simulation Conference (WSC). https://doi.org/10.1109/wsc.2015.7408291

Sarkar, N. I., & Gutierrez, J. A. (2014). Revisiting the issue of the credibility of simulation studies in telecommunication networks: Highlighting the results of a comprehensive

survey of IEEE publications. IEEE Communications Magazine, 52(5), 218–224. https://doi.org/10.1109/mcom.2014.6815915

Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2009). A study on the relationships of classifier performance metrics. 2009 21st IEEE International Conference on Tools with Artificial Intelligence. https://doi.org/10.1109/ictai.2009.25

Sha, K., Yang, T. A., Wei, W., & Davari, S. (2020). A survey of edge computing-based designs for IoT security. Digital Communications and Networks, 6(2), 195-202. doi:10.1016/j.dcan.2019.08.006

Shah, T., & Venkatesan, S. (2018). Authentication of IoT Device and IoT Server Using Secure Vaults. *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)*. Published. https://doi.org/10.1109/trustcom/bigdatase.2018.00117

Shakdher, A., Agrawal, S., & Yang, B. (2019). Security Vulnerabilities in Consumer IoT Applications. 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). https://doi.org/10.1109/bigdatasecurity-hpsc-ids.2019.00012

Siddiqui, F., Beley, J., Zeadally, S., & Braught, G. (2019). Secure and lightweight communication in heterogeneous IoT environments. *Internet of Things*, 100093. https://doi.org/10.1016/j.iot.2019.100093

Singh, J., Gimekar, A., & Venkatesan, S. (2019). An efficient lightweight authentication scheme for human-centered industrial Internet of Things. *International Journal of Communication Systems*, e4189. https://doi.org/10.1002/dac.4189

Stévant, B., Pazat, J.-L., & Blanc, A. (2020). QoS-aware autonomic adaptation of microservices placement on Edge Devices. Proceedings of the 10th International Conference on Cloud Computing and Services Science. https://doi.org/10.5220/0009319902370244

Suman, S., Perumal, T., Mustapha, N., & Yaakob, R. (2019). Device Verification and Compatibility for Heterogeneous Semantic IoT Systems. 2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE). https://doi.org/10.1109/icraie47735.2019.9037767

Toshihiko, O. (2017). Lightweight Cryptography Applicable to Various IoT Devices. NEC Technical Journal／Vol.12 No.1／Special Issue on IoT That Supports Digital Businesses.

Trafimow, D. (2018). Confidence intervals, precision and confounding. New Ideas in Psychology, 50, 48–53. https://doi.org/10.1016/j.newideapsych.2018.04.005

Vijayma. (2021). Azure iot edge task - azure pipelines. Azure Pipelines | Microsoft Docs. Retrieved January 4, 2022, from https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/build/azure-iot-edge?view=azure-devops

Wang, J., Liu, K., & Pan, J. (2020). Online UAV-Mounted Edge Server Dispatching for Mobile-to-Mobile Edge Computing. IEEE Internet of Things Journal, 7(2), 1375–1386. https://doi.org/10.1109/jiot.2019.2954798

Wang, Y., Tang, M., Zhou, S., Tan, G., Zhang, Z., & Zhan, J. (2020). Performance Analysis of Heterogeneous Mobile Edge Computing Networks with Multi-core Server. 2020 IEEE 20th International Conference on Communication Technology (ICCT). https://doi.org/10.1109/icct50939.2020.9295920

Wazid, M., Das, A. K., Kumar, N., Vasilakos, A. V., & Rodrigues, J. J. P. C. (2019). Design and Analysis of Secure Lightweight Remote User Authentication and Key Agreement Scheme in Internet of Drones Deployment. *IEEE Internet of Things Journal*, *6*(2), 3572–3584. https://doi.org/10.1109/jiot.2018.2888821

Williams, R., McMahon, E., Samtani, S., Patton, M., & Chen, H. (2017). Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach. 2017 IEEE International Conference on Intelligence and Security Informatics (ISI). https://doi.org/10.1109/isi.2017.8004904

Wolfart, D., Assunção, W. K., da Silva, I. F., Domingos, D. C., Schmeing, E., Villaca, G. L., & Paza, D. do. (2021). Modernizing legacy systems with µs: A roadmap. Evaluation and Assessment in Software Engineering. https://doi.org/10.1145/3463274.3463334

Xing, L. (2021). Cascading failures in internet of things: Review and Perspectives on Reliability and Resilience. IEEE Internet of Things Journal, 8(1), 44–64. https://doi.org/10.1109/jiot.2020.3018687

Xu, H., Yu, W., Griffith, D., & Golmie, N. (2018). A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective. IEEE Access, 6, 78238–78259. https://doi.org/10.1109/access.2018.2884906

Yang, H., Alphones, A., Zhong, W.-D., Chen, C., & Xie, X. (2020). Learning-Based Energy-Efficient Resource Management by Heterogeneous RF/VLC for Ultra-Reliable Low-Latency Industrial IoT Networks. IEEE Transactions on Industrial Informatics, 16(8), 5565–5576. https://doi.org/10.1109/tii.2019.2933867

Yarali, A. (2018). IOT: Platforms, connectivity, applications and services. Amazon. Retrieved January 4, 2022, from https://docs.aws.amazon.com/iot/latest/developerguide/secure-tunneling.html

Yu, X., & Guo, H. (2019). A Survey on IIoT Security. 2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS). https://doi.org/10.1109/vts-apwcs.2019.8851679

Zahoor, S., & Mir, R. N. (2021). Resource Management in pervasive internet of things: A survey. Journal of King Saud University - Computer and Information Sciences, 33(8), 921–935. https://doi.org/10.1016/j.jksuci.2018.08.014

Zarrad, A., & Alsmadi, I. (2017). Evaluating network test scenarios for network simulators systems. International Journal of Distributed Sensor Networks, 13(10), 155014771773821. https://doi.org/10.1177/1550147717738216

Zelkowitz, M. V., & Wallace, D. R. (1998). Experimental models for validating technology. Computer, 31(5), 23-31.

Zhang, Y., He, D., Li, L., & Chen, B. (2020). A lightweight authentication and key agreement scheme for Internet of Drones. *Computer Communications*, *154*, 455–464. https://doi.org/10.1016/j.comcom.2020.02.067

Zhong, C.-L., Zhu, Z., & Huang, R.-G. (2015). Study on the IOT Architecture and Gateway Technology. 2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES). https://doi.org/10.1109/dcabes.2015.56

Zhou, L., Su, C., & Yeh, K. H. (2019). A Lightweight Cryptographic Protocol with Certificateless Signature for the Internet of Things. *ACM Transactions on Embedded Computing Systems*, *18*(3), 1–10. https://doi.org/10.1145/3301306

Zhu, L., Yang, Z., Li, M., & Liu, D. (2013). An Efficient Data Aggregation Protocol Concentrated on Data Integrity in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks*, *9*(5), 256852. https://doi.org/10.1155/2013/256852